PENGEMBANGAN KITCHEN MANAGEMENT SYSTEM PADA POINT OF SALE UNTUK USAHA KULINER



Disusun Oleh:

N a m a : Naufal Rafi Hidayat
NIM : 22523285

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2026

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGEMBANGAN KITCHEN MANAGEMENT SYSTEM PADA POINT OF SALE UNTUK USAHA KULINER

TUGAS AKHIR JALUR MAGANG

Disusun Oleh:

N a m a : Naufal Rafi Hidayat
NIM : 22523285

Yogyakarta, 28 Juni 2026

Pembimbing,

(Dr. Sri Kusumadewi, S.Si., M.T.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGEMBANGAN KITCHEN MANAGEMENT SYSTEM PADA POINT OF SALE UNTUK USAHA KULINER

TUGAS AKHIR JALUR MAGANG

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 24 Juli 2026

Tim Penguji

Prof., Dr. Sri Kusumadewi, S.Si., M.T.

Anggota 1

Elyza Gustri Wahyuni, S.T., M.Cs.

Anggota 2

Rahadian Kurniawan, S.Kom., M.Kom.

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Ir. Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. IPM., ASEAN Eng.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Naufal Rafi Hidayat

NIM : 22523285

Tugas akhir dengan judul:

PENGEMBANGAN KITCHEN MANAGEMENT SYSTEM PADA POINT OF SALE UNTUK USAHA KULINER

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 28 Juni 2026



(Naufal Rafi Hidayat)

HALAMAN PERSEMBAHAN

Puji syukur ke hadirat Allah Subhanahu wa Ta'ala atas segala rahmat, hidayah, dan kemudahan yang telah diberikan sehingga tugas akhir ini dapat diselesaikan dengan baik.

Karya sederhana ini kupersembahkan untuk:

1. **Kedua orang tuaku, Bapak dan Ibu tercinta,**

   Yang telah memberikan segalanya kasih sayang, doa, pengorbanan, dan dukungan yang tak pernah putus. Setiap tetes keringat dan setiap doa yang dipanjatkan menjadi kekuatan bagiku untuk terus melangkah dan menyelesaikan perjalanan ini. Terima kasih telah menjadi rumah pertama dan terbaik bagi hidupku.

2. **Ibu Dr. Sri Kusumadewi, S.Si., M.T.,**

   Selaku dosen pembimbing yang dengan sabar telah meluangkan waktu, tenaga, dan pikiran untuk membimbing, mengarahkan, serta memberikan masukan yang sangat berharga sejak awal hingga terselesaikannya tugas akhir ini. Setiap arahan dan motivasi yang diberikan menjadi cahaya dalam setiap langkah penyusunan karya ini. Terima kasih atas segala ilmu, kesabaran, dan ketulusan yang telah ibu berikan.

3. **Bapak dan Ibu dosen Program Studi Informatika,**

   Serta para dosen penguji yang telah memberikan kritik dan saran membangun demi kesempurnaan tugas akhir ini.

4. **Seluruh keluarga besar dan sahabat,**

   Yang telah memberikan dukungan moral, semangat, serta doa tulus yang mengiringi setiap langkahku.

5. **PT. Vastu Cipta Persada,**

   Khususnya tim pengembang CAVAA, yang telah memberikan kesempatan untuk belajar dan berkontribusi secara langsung di dunia industri.

6. **Dan untuk diriku sendiri,**

   Yang telah bertahan, berusaha, dan tidak pernah menyerah melewati setiap tantangan dalam proses belajar dan berkarya. Terima kasih telah percaya bahwa setiap usaha akan membuahkan hasil pada waktu yang tepat.

HALAMAN MOTO

وَجَزٰٓؤُا سَيِّئَةٍ سَيِّئَةٌ مِّثْلُهَاۚ فَمَنْ عَفَا وَاَصْلَحَ فَاَجْرُهٗ عَلَى اللّٰهِۗ اِنَّهٗ لَا يُحِبُّ الظّٰلِمِيْنَ ﴿٤٠﴾

"Balasan suatu keburukan adalah keburukan yang setimpal. Akan tetapi, siapa yang memaafkan dan berbuat baik (kepada orang yang berbuat jahat), maka pahalanya dari Allah. Sesungguhnya Dia tidak menyukai orang-orang zalim."

QS. Asy-Syura ayat 40.

"Aku mulai lelah dan aku butuh tempat untuk memulai. Dan jika kamu punya waktu sebentar, mengapa kita tidak pergi? Mari kita bicarakan hal ini di tempat yang hanya kita berdua tahu. Ini bisa jadi akhir dari segalanya. Jadi, kenapa kita tidak pergi saja? Tempat yang hanya kita yang tahu?"

Keane, Somewhere only we know.

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah Subhanahu wa Ta'ala atas segala rahmat dan karunia-Nya sehingga laporan tugas akhir jalur magang ini dapat diselesaikan dengan baik. Laporan berjudul "Pengembangan Kitchen Management System pada Point of Sale untuk Usaha Kuliner" ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika -- Program Sarjana, Fakultas Teknologi Industri, Universitas Islam Indonesia.

Selama pelaksanaan magang dan penyusunan laporan, penulis menghadapi berbagai tantangan, antara lain menyeimbangkan tuntutan pekerjaan pengembangan sistem dengan kewajiban akademik, memahami alur bisnis operasional restoran, serta mengimplementasikan teknik optimasi pada Kitchen Dashboard. Namun berkat bimbingan dan dukungan berbagai pihak, seluruh kendala tersebut dapat teratasi.

Pada kesempatan ini, penulis menyampaikan terima kasih kepada:

1. **Ibu Dr. Sri Kusumadewi, S.Si., M.T**. selaku dosen pembimbing yang telah meluangkan waktu dan pikiran dalam membimbing penulis;
2. **PT. Vastu Cipta Persada**, khususnya tim pengembang CAVAA, atas kesempatan dan pengalaman berharga selama magang;
3. **Kedua orang tua dan keluarga** tercinta atas doa dan dukungan yang tiada henti;
4. **Seluruh dosen, staf, dan teman-teman Program Studi Informatika** yang telah memberikan ilmu dan dukungan selama perkuliahan.

Penulis berharap laporan ini dapat memberikan manfaat bagi pengembangan ilmu pengetahuan dan menjadi referensi bagi pengembangan sistem serupa di masa mendatang. Kritik dan saran yang membangun sangat penulis harapkan.

Yogyakarta, 28 Juni 2026



(Naufal Rafi Hidayat)

SARI

Industri restoran menghadapi tantangan koordinasi antara kasir dan dapur yang masih dilakukan secara manual melalui media cetak atau komunikasi langsung. Metode tersebut memiliki keterbatasan, terutama saat jumlah pesanan meningkat, sehingga berisiko menyebabkan keterlambatan penyajian, kesalahan informasi, dan menurunnya kepuasan pelanggan. Berdasarkan permasalahan tersebut, dikembangkan *Kitchen Dashboard* sebagai bagian dari sistem *Point of Sale* (POS) restoran berbasis web untuk membantu *staff* dapur memantau dan mengelola pesanan secara *real-time*.

Laporan ini disusun berdasarkan hasil magang selama enam bulan di PT. Vastu Cipta Persada pada proyek CAVAA. Penulis dilibatkan dalam perancangan, implementasi, dan pengujian modul *Kitchen Dashboard*. Sistem dibangun menggunakan Laravel pada sisi *backend* dengan arsitektur MVC, serta JavaScript dan Tailwind CSS pada sisi *frontend*. Komunikasi antar lapisan menggunakan REST API dengan format JSON.

Kitchen Dashboard menerapkan *polling* API interval 3 detik untuk pembaruan data otomatis. Optimasi dilakukan melalui *hash comparison* untuk membatasi render ulang, *Page Visibility* API untuk menghentikan *polling* saat tab tidak aktif, dan *debounce* pada fitur pencarian. Antarmuka dirancang responsif menggunakan CSS *Grid* untuk mendukung berbagai perangkat.

Hasil pengembangan menunjukkan *dashboard* menampilkan tiga panel utama: antrian, aktif, dan selesai. Fitur pengelolaan status pesanan (*pickup*, *cancel*, *serve*) berfungsi dengan baik dan langsung tercermin pada tampilan tanpa penyegaran manual. Penerapan teknik optimasi terbukti meningkatkan efisiensi sumber daya sistem. Integrasi *frontend* dan *backend* melalui REST API berjalan sesuai rancangan dengan konsistensi data yang terjaga pada basis data.

Kata kunci: *Kitchen Dashboard*, POS, REST API, Laravel, JavaScript, *polling*, optimasi sistem.

GLOSARIUM

Frontend	Bagian dari aplikasi yang berinteraksi langsung dengan pengguna, menampilkan antarmuka dan menangani interaksi melalui browser.
Backend	Bagian dari aplikasi yang berjalan di sisi server dan bertanggung jawab mengelola logika bisnis, basis data, serta menyediakan layanan melalui API.
Blade	*Template engine* bawaan framework Laravel yang digunakan untuk membangun tampilan antarmuka pengguna dengan sintaks yang sederhana dan terintegrasi langsung dengan PHP.
Debounce	Teknik optimasi yang menunda eksekusi suatu fungsi hingga pengguna berhenti melakukan aktivitas tertentu dalam periode waktu yang ditentukan.
Eloquent ORM	Fitur bawaan Laravel yang menyediakan cara interaksi dengan basis data menggunakan objek dan model, sehingga *query database* dapat ditulis dengan sintaks yang lebih ekspresif dan aman.
Endpoint URL	spesifik pada REST API yang digunakan untuk mengakses sumber daya tertentu, seperti data pesanan atau status pesanan.
Hash Comparison	Teknik membandingkan dua kondisi data menggunakan nilai *hash* untuk mendeteksi perubahan dan menentukan apakah proses render ulang diperlukan.
JSON	*JavaScript Object Notation*, format pertukaran data yang ringan dan mudah dibaca, umum digunakan dalam komunikasi REST API.
Kitchen Dashboard	Antarmuka berbasis web yang digunakan oleh *staff* dapur untuk memantau dan mengelola pesanan pelanggan secara *real-time*.
Middleware	Lapisan pada aplikasi Laravel yang berfungsi sebagai filter permintaan HTTP sebelum mencapai *controller*, digunakan untuk autentikasi dan otorisasi.
MVC	*Model-View-Controller*, pola arsitektur perangkat lunak yang memisahkan aplikasi menjadi tiga komponen: *Model* (data), *View* (tampilan), dan *Controller* (logika).
Page Visibility API	API pada browser yang memungkinkan aplikasi mendeteksi apakah halaman web sedang aktif atau tidak, digunakan untuk mengoptimalkan penggunaan sumber daya.

Polling	Teknik pengambilan data dari server secara berkala dalam interval waktu tertentu untuk memperoleh informasi terbaru tanpa interaksi manual.
REST API	*Representational State Transfer Application Programming Interface*, gaya arsitektur komunikasi antar aplikasi menggunakan protokol HTTP dengan format data JSON.

DAFTAR ISI

HALAMAN JUDUL i
HALAMAN PENGESAHAN DOSEN PEMBIMBING ii
HALAMAN PENGESAHAN DOSEN PENGUJI iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR iv
HALAMAN PERSEMBAHAN v
HALAMAN MOTO vi
KATA PENGANTAR vii
SARI viii
GLOSARIUM ix
DAFTAR ISI xi
DAFTAR TABEL xiii
DAFTAR GAMBAR xiv
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Ruang Lingkup 2
1.3 Tujuan 3
1.4 Manfaat 4
1.5 Sistematika Penulisan 5
BAB II LANDASAN TEORI DAN TINJAUAN PUSTAKA 7
2.1 *Point of Sale* (POS) 7
2.2 *Kitchen Management System* 7
2.3 REST API 8
2.4 JavaScript 8
2.5 Laravel 9
2.6 *Polling* 9
2.7 *Hash Comparison* 9
2.8 *Page Visibility API* 10
2.9 *Debounce* 10
2.10 IDOR 10
2.11 Tinjauan Pustaka 11
BAB III PELAKSANAAN MAGANG 13

3.1 Aktivitas Magang 13
3.1.1 *On-boarding* Magang 13
3.1.2 Penugasan dan Ruang Lingkup Magang 14
3.1.3 Metodologi Pengembangan Agile 17
3.2 Analisis dan Perancangan Sistem Kitchen Dashboard 20
3.2.1 Analisis Kebutuhan Sistem 20
3.2.2 Analisis Kebutuhan Fungsional 20
3.2.3 Analisis Kebutuhan Non-Fungsional 23
3.2.4 Perancangan Arsitektur Sistem 23
3.2.5 Perancangan Antarmuka Pengguna 27
3.3 Realisasi dan Pengembangan Sistem *Kitchen Dashboard* 28
3.3.1 Implementasi *Frontend* Menggunakan Blade dan Tailwind CSS 28
3.3.2 Implementasi Backend Menggunakan Laravel 36
3.3.3 Pengembangan REST API 40
3.3.4 Pengembangan Struktur Frontend Modular 42
3.3.5 Implementasi Pengelolaan Status Pesanan 44
3.4 Optimalisasi Mekanisme Pembaruan Data 48
3.4.1 Implementasi Mekanisme Polling API 49
3.4.2 Implementasi *Hash Comparison* 52
3.4.3 Implementasi *Page Visibility API* 59
3.4.4 Implementasi Debounce 64
3.5 Evaluasi Sistem 65
BAB IV REFLEKSI PELAKSANAAN MAGANG 73
4.1 Relevansi Akademik 73
4.2 Pembelajaran Magang 75
BAB V PENUTUP 78
5.1 Kesimpulan 78
5.2 Saran.. 79
DAFTAR PUSTAKA 81
LAMPIRAN 84

DAFTAR TABEL

Tabel 3. 1 Aktivitas Magang. 13
Tabel 3. 2 *Tools* yang Digunakan Selama Pengembangan. 15
Tabel 3. 3 Hasil Analisis Kebutuhan Pengguna. 20
Tabel 3. 4 Pemetaan Alur Status Pesanan. 21
Tabel 3. 5 Kebutuhan Non-Fungsional. 23
Tabel 3. 6 *Endpoint* REST API *Kitchen Dashboard.* 25
Tabel 3. 7 Tabel *Utility* Tailwind CSS 31
Tabel 3. 8 Perbandingan Jumlah *Render*. 57
Tabel 3. 9 Perbandingan Jumlah *Request*. 63
Tabel 3.10 Hasil Kuesioner 67
Tabel 3.10 Hasil Pengukuran Waktu Respons Endpoint. 69

DAFTAR GAMBAR

Gambar 3. 1 Diagram MVC.....14
Gambar 3. 2 Tampilan *branch* Rafi-UII-Team pada repositori proyek sebagai media pengumpulan hasil pekerjaan magang. 16
Gambar 3. 3 *Use Case Diagram Kitchen Dashboard*.....22
Gambar 3. 4 Flowchart Alur Pemesanan *Kitchen Dashboard.*.....23
Gambar 3. 5 Diagram Arsitektur Sistem *Kitchen Dashboard*. 27
Gambar 3. 6 Hasil Desain Figma *Kitchen Dashboard*. 28
Gambar 3. 7 Potongan Kode *Layout* Blade. 30
Gambar 3. 8 Potongan Kode *Grid.* 30
Gambar 3. 9 Tampilan Desktop *Kitchen Dashboard.*.....32
Gambar 3. 10 Tampilan Mobile *Kitchen Dashboard*. 33
Gambar 3. 11 Tampilan Modal Detail Pesanan pada Antrian. 34
Gambar 3. 12 Tampilan Modal Detail Pesanan Aktif.....35
Gambar 3. 13 Tampilan Modal Detail Pesanan Selesai.....36
Gambar 3. 14 Potongan Kode *Controller Kitchen Dashboard*.....37
Gambar 3. 15 Hasil verifikasi penyimpanan data pesanan pada basis data menggunakan TablePlus (Data sampel sistem).....39
Gambar 3. 16 Potongan Kode *Method _fetch*(). 40
Gambar 3. 17 Potongan Kode *_fetchWithCache*().....41
Gambar 3. 18 Hasil Respon JSON dari Setiap *Endpoint*.....42
Gambar 3. 19 Potongan Kode *createQueueItem*().....43
Gambar 3. 20 Potongan Kode *createQueueItem*().....43
Gambar 3. 21 Potongan Kode *pickUpOrder.*.....45
Gambar 3. 22 Potongan Kode *cancelOrder.*.....46
Gambar 3. 23 Potongan Kode *markAsServed.*.....47
Gambar 3. 24 Visualisasi Data Status Pesanan yang Tersimpan Menggunakan TablePlus. ...48
Gambar 3. 25 *Sequence Diagram* Mekanisme *Polling* dan *Hash Comparison*.....50
Gambar 3. 26 Potongan Kode *startRealtimePolling*.....51
Gambar 3. 27 Data Sebelum Masuk. 52
Gambar 3. 28 Data Langsung Masuk 52
Gambar 3. 29 Potongan Kode *checkForUpdates.* 54
Gambar 3. 30 Hasil Pengujian Jumlah *Render* Sebelum Penerapan *Hash Comparison* 55

Gambar 3. 31 Hasil Pengujian Jumlah *Render* Setelah Penerapan *Hash Comparison*. 56
Gambar 3. 32 Alur Kerja Optimasi *Hash Comparison*.....59
Gambar 3. 33 Potongan Kode *Page Visibility API.* 60
Gambar 3. 34 Hasil Pengujian Jumlah Request Sebelum Penerapan *Page Visibility API*. 61
Gambar 3. 35 Hasil Pengujian Jumlah *Request* Setelah Penerapan *Page Visibility* API. 62
Gambar 3. 36 Potongan Kode *Debounce*.....65
Gambar 3. 37 Fitur Pencarian Data Antrian Pesanan. 65
Gambar 3. 38 Hasil Pengamatan *Request* dan Waktu *Respons Endpoint* melalui *Chrome DevTools* 69

BAB I
PENDAHULUAN

1.1 Latar Belakang

Salah satu teknologi yang banyak digunakan pada industri restoran adalah sistem *Point of Sale* (POS). Sistem POS berfungsi untuk mengelola transaksi penjualan, pencatatan pesanan pelanggan, pengelolaan data produk, serta penyajian laporan operasional. Dengan adanya sistem POS, proses pelayanan dapat dilakukan secara lebih terstruktur karena seluruh data transaksi tersimpan dan dikelola dalam satu sistem yang terintegrasi (Putu et al., 2018).

Selama pelaksanaan kegiatan magang, penulis terlibat dalam pengembangan sistem POS restoran berbasis web yang dikembangkan menggunakan framework Laravel. Sistem tersebut terdiri dari beberapa modul yang mendukung kebutuhan operasional restoran, mulai dari pengelolaan pesanan hingga pengelolaan transaksi penjualan. Dari berbagai modul yang dikembangkan, *Kitchen Dashboard* menjadi salah satu modul yang memiliki peran penting karena digunakan secara langsung oleh *staff* dapur dalam mengelola pesanan pelanggan.

Pada proses operasional restoran, koordinasi antara kasir dan dapur menjadi faktor yang sangat menentukan kualitas pelayanan kepada pelanggan. Setelah pesanan dicatat oleh kasir, informasi tersebut harus segera diteruskan kepada *staff* dapur agar proses penyajian makanan dapat dilakukan tanpa keterlambatan. Apabila proses komunikasi tidak berjalan dengan baik, maka dapat terjadi kesalahan pesanan, keterlambatan penyajian, maupun ketidaksesuaian informasi yang berdampak pada kepuasan.

Pada beberapa restoran, proses penyampaian informasi pesanan masih dilakukan secara manual melalui media cetak maupun komunikasi langsung antar pegawai. Metode tersebut memiliki berbagai keterbatasan, terutama ketika jumlah pesanan meningkat pada jam operasional yang sibuk. Kondisi tersebut dapat menyebabkan informasi pesanan terlambat diterima oleh dapur sehingga menghambat proses pelayanan kepada pelanggan.

Untuk mengatasi permasalahan tersebut, perlu dikembangkan *Kitchen Dashboard* sebagai antarmuka berbasis web yang mampu menampilkan informasi pesanan secara *real-time*. *Kitchen Dashboard* harus terintegrasi dengan sistem POS sehingga setiap pesanan yang masuk dapat langsung ditampilkan pada layar dapur. Dengan adanya sistem ini, *staff* dapur dapat memantau dan mengelola pesanan secara lebih cepat, akurat, dan terorganisir.

Kitchen Dashboard yang dikembangkan pada sistem POS ini memiliki sejumlah kelebihan dibandingkan dengan sistem dashboard konvensional. Dari sisi integrasi, *dashboard* terhubung secara *real-time* dengan sistem POS melalui REST API sehingga setiap pesanan yang dimasukkan kasir langsung muncul pada antrian dapur tanpa jeda. Transisi status pesanan dari *paid*, *processed*, hingga *served* tercatat dengan akurat dan dapat diaudit, yang membantu *staff* dapur dalam memantau beban kerja serta memprioritaskan pesanan secara lebih terstruktur.

Dari segi antarmuka, dashboard menyajikan tiga panel terpisah untuk antrian, pesanan aktif, dan pesanan selesai sehingga *staff* dapur memiliki visibilitas penuh terhadap seluruh kondisi operasional. Kartu pesanan dirancang ringkas dengan indikator visual seperti warna dan efek *opacity* untuk memudahkan pemindaian cepat, sementara modal detail yang terstruktur menampilkan informasi lengkap termasuk daftar menu dan catatan khusus tanpa mengganggu tampilan utama.

Kelebihan lain terletak pada fleksibilitas akses karena sistem berbasis web sehingga dapat diakses melalui berbagai perangkat seperti desktop, tablet, maupun ponsel tanpa memerlukan instalasi khusus. Dari sisi teknis, arsitektur modular dengan pemisahan kelas pada *frontend* memudahkan pemeliharaan dan pengembangan fitur baru di masa mendatang. Selain itu, penerapan teknik optimasi seperti *hash comparison* untuk mencegah render ulang yang tidak perlu, *Page Visibility* API untuk menghentikan *polling* saat tab tidak aktif, serta *debounce* pada fitur pencarian menjadikan sistem ini lebih efisien dalam penggunaan sumber daya dibandingkan *dashboard* lainnya.

Berdasarkan latar belakang tersebut, laporan tugas akhir ini berfokus pada pengembangan *Kitchen Dashboard* sebagai bagian dari sistem POS restoran. Pembahasan difokuskan pada implementasi *frontend* yang terintegrasi dengan *backend* Laravel melalui *REST API* untuk mendukung pengelolaan pesanan dapur secara *real-time*. Melalui pengembangan ini diharapkan proses koordinasi antara kasir dan dapur dapat berlangsung lebih efektif sehingga kualitas pelayanan restoran dapat meningkat.

1.2 Ruang Lingkup

Kegiatan magang dilaksanakan pada pengembangan sistem *Point of Sale* (POS) restoran berbasis web. Sistem tersebut terdiri dari beberapa modul yang mendukung operasional restoran, namun laporan tugas akhir ini hanya membahas modul *Kitchen Dashboard* yang digunakan oleh *staff* dapur. Pembatasan ruang lingkup dilakukan agar pembahasan lebih terfokus pada proses pengembangan dan implementasi *Kitchen Dashboard*.

Laporan tugas akhir ini berfokus pada pengembangan Kitchen Dashboard sebagai bagian dari sistem *Point of Sale* (POS) restoran berbasis web. Sistem POS secara keseluruhan terdiri dari berbagai modul yang mendukung operasional restoran, seperti modul transaksi kasir, pengelolaan menu dan produk, pengaturan meja, manajemen pelanggan, serta pembuatan laporan penjualan. Namun demikian, laporan ini hanya membahas salah satu modul, yaitu *Kitchen Dashboard* yang digunakan oleh *staff* dapur dalam memantau dan mengelola pesanan pelanggan.

Pembatasan ruang lingkup dilakukan dengan pertimbangan bahwa modul *Kitchen Dashboard* memiliki peran strategis dalam kelancaran operasional restoran, khususnya dalam menjembatani komunikasi antara bagian kasir dan bagian dapur. Fokus pembahasan diarahkan pada proses pengembangan sistem yang meliputi tahap analisis kebutuhan, perancangan arsitektur, implementasi antarmuka dan *backend*, serta pengujian fungsionalitas sistem. Aspek-aspek lain di luar modul *Kitchen Dashboard*, seperti proses transaksi pembayaran, pencatatan keuangan, dan laporan manajerial, tidak dibahas dalam laporan ini karena berada di luar cakupan pengembangan yang menjadi tanggung jawab penulis selama kegiatan magang.

Dari sisi teknis, pembahasan difokuskan pada implementasi antarmuka pengguna berbasis JavaScript yang terintegrasi dengan *backend* Laravel melalui REST API, serta penerapan berbagai teknik optimasi untuk meningkatkan performa dan efisiensi sistem. Mekanisme pembaruan data secara *real-time* melalui *polling* API, pengelolaan status pesanan, dan perancangan tampilan responsif untuk berbagai perangkat juga menjadi bagian dari cakupan pembahasan. Sementara itu, aspek infrastruktur seperti konfigurasi server, manajemen basis data, dan keamanan jaringan tidak dibahas secara mendalam karena dianggap sebagai bagian dari dukungan teknis yang telah disediakan oleh perusahaan tempat magang.

Dengan pembatasan ruang lingkup tersebut, tugas akhir ini diharapkan dapat memberikan pemahaman yang komprehensif mengenai proses pengembangan *Kitchen Dashboard* secara terfokus dan mendalam, mulai dari identifikasi kebutuhan pengguna hingga implementasi solusi yang siap digunakan dalam operasional restoran.

1.3 Tujuan

Pengembangan *Kitchen Dashboard* pada sistem POS restoran memiliki beberapa tujuan yang ingin dicapai selama pelaksanaan kegiatan magang. Tujuan tersebut difokuskan pada penyediaan sistem yang mampu membantu *staff* dapur dalam mengelola pesanan secara efektif dan efisien. Adapun tujuan dari pengembangan *Kitchen Dashboard* adalah sebagai berikut:

a. Membangun antarmuka dashboard dapur berbasis web yang mampu menampilkan data pesanan secara *real-time* melalui mekanisme *polling API*.
b. Mengintegrasikan *Kitchen Dashboard* dengan *backend* Laravel menggunakan *REST API* untuk mendukung pertukaran data pesanan antara sistem POS dan dapur.
c. Mengimplementasikan teknik *hash comparison* guna mengoptimalkan proses render antarmuka sehingga pembaruan tampilan hanya dilakukan ketika mendapat perubahan data.
d. Mengimplementasikan *Page Visibility API* untuk meningkatkan efisiensi pengguna sumber daya sistem serta mengurangi beban server.
e. Menyediakan fitur pengelolaan status pesanan yang meliputi proses pengambilan pesanan dari antrian, penyelesaian pesanan dan pembatalan pesanan aktif.

1.4 Manfaat

Pengembangan *Kitchen Dashboard* memberikan manfaat bagi pengguna maupun pihak yang terlibat dalam pengoperasian sistem restoran. Manfaat tersebut berkaitan dengan peningkatan efisiensi kerja, kemudahan pengelolaan pesanan, serta optimalisasi penggunaan sumber daya sistem. Adapun manfaat yang diperoleh dari pengembangan *Kitchen Dashboard* adalah sebagai berikut:

a. Membantu *staff* dapur dalam memantau dan mengelola pesanan secara *real-time* sehingga proses pengerjaan pesanan dapat dilakukan dengan lebih cepat dan terorganisir.
b. Meningkatkan koordinasi antara kasir dan dapur melalui penyampaian informasi pesanan yang lebih cepat, akurat, dan terintegrasi.
c. Mengurangi penggunaan sumber daya sistem melalui penerapan *hash comparison* yang membatasi proses render ulang hanya ketika terjadi perubahan data.
d. Mengurangi konsumsi *bandwidth* dan beban server melalui penghentian *polling* otomatis saat tab browser tidak aktif serta pembatasan permintaan pencarian menggunakan teknik *debounce*.
e. Mempermudah pemantauan status pesanan mulai dari antrian, proses pengerjaan, hingga pesanan selesai disajikan kepada pelanggan.
f. Memberikan fleksibilitas penggunaan sistem pada berbagai perangkat melalui desain antarmuka yang responsif berbasis CSS *Grid*.

1.5 Sistematika Penulisan

Laporan tugas akhir ini disusun secara sistematis agar pembahasan mengenai pengembangan *Kitchen Dashboard* dapat dipahami dengan mudah. Setiap bab memiliki fokus pembahasan yang berbeda namun saling berkaitan satu sama lain. Adapun sistematika penulisan laporan tugas akhir ini adalah sebagai berikut.

BAB I PENDAHULUAN

Bab ini menjelaskan latar belakang pengembangan Kitchen Dashboard sebagai bagian dari sistem POS restoran, yang dilatarbelakangi oleh permasalahan koordinasi manual antara kasir dan dapur. Bab ini juga memuat ruang lingkup pembahasan yang berfokus pada modul Kitchen Dashboard, tujuan pengembangan yang mencakup implementasi antarmuka berbasis web, integrasi REST API, serta penerapan teknik optimasi, dan manfaat yang diperoleh bagi operasional restoran. Melalui bab ini, pembaca dapat memahami gambaran umum mengenai permasalahan dan solusi yang dikembangkan.

BAB II LANDASAN TEORI

Bab ini berisi teori-teori yang digunakan sebagai dasar dalam pengembangan sistem. Materi yang dibahas meliputi konsep *Point of Sale* (POS), *Kitchen Management System*, REST API, JavaScript, Laravel, *polling*, *hash comparison*, *debounce*, serta *Page Visibility* API. Selain itu, bab ini juga memuat tinjauan pustaka yang membandingkan penelitian-penelitian terdahulu dengan pengembangan yang dilakukan, sekaligus mengidentifikasi kebaruan dari sistem yang dikembangkan, terutama dalam hal optimasi performa *frontend*. Landasan teori dan tinjauan pustaka digunakan untuk mendukung proses analisis dan implementasi sistem yang dilakukan.

BAB III PEMBAHASAN

Bab ini berisi tentang tahapan pelaksanaan pengembangan *Kitchen Dashboard* pada aplikasi CAVAA yang dilakukan oleh penulis sebagai *fullstack* developer.

BAB IV REFLEKSI PELAKSANAAN MAGANG

Bab ini berisi refleksi penulis terhadap pengalaman magang yang telah dilaksanakan. Refleksi mencakup analisis relevansi akademik yang membandingkan teori-teori yang dipelajari di bangku kuliah (seperti REST API, optimasi kinerja *frontend*, dan pengujian sistem) dengan praktik implementasi di lapangan, serta mengidentifikasi kesenjangan yang terjadi, khususnya dalam pelaksanaan *User Acceptance Testing* (UAT). Bab ini juga memaparkan pembelajaran yang diperoleh selama magang, baik dari sisi teknis seperti penguasaan Laravel, JavaScript, dan konsep *real-time*, maupun dari sisi non-teknis seperti komunikasi, kolaborasi tim, dan

manajemen waktu. Refleksi ini diharapkan dapat memberikan gambaran tentang keterkaitan antara pendidikan akademik dan dunia industri.

BAB V PENUTUP

Bab ini berisi kesimpulan yang diperoleh dari keseluruhan proses pengembangan *Kitchen Dashboard*, yang mencakup keberhasilan sistem dalam memenuhi kebutuhan operasional dapur, integrasi REST API yang berjalan dengan baik, serta penerapan teknik optimasi yang terbukti meningkatkan efisiensi sistem. Bab ini juga memuat saran yang dapat digunakan sebagai acuan untuk pengembangan sistem pada masa yang akan datang, baik bagi perusahaan terkait pelibatan pengguna akhir dalam pengujian dan eksplorasi teknologi *WebSocket*, maupun bagi pengembang selanjutnya terkait pendalaman arsitektur sistem dan pengembangan fitur analitik. Kesimpulan dan saran diharapkan dapat memberikan gambaran mengenai pencapaian serta peluang pengembangan lebih lanjut.

BAB II
LANDASAN TEORI DAN TINJAUAN PUSTAKA

2.1 *Point of Sale* (POS)

Point of Sale (POS) merupakan sistem yang digunakan untuk mendukung proses transaksi penjualan dan pengelolaan operasional bisnis. Pada industri restoran, POS tidak hanya digunakan untuk mencatat transaksi pembayaran, tetapi juga berfungsi sebagai pusat pengelolaan pesanan, data menu, laporan penjualan, dan komunikasi dengan bagian dapur. Sistem POS modern memungkinkan setiap pesanan yang dimasukkan oleh kasir dapat langsung diteruskan ke bagian dapur melalui jaringan komputer sehingga proses pelayanan menjadi lebih cepat dan terintegrasi.

Penerapan sistem POS memberikan berbagai manfaat bagi operasional restoran. Selain meningkatkan akurasi pencatatan transaksi, sistem ini juga mampu mengurangi kesalahan komunikasi antara bagian pelayanan dan dapur. Integrasi antara POS dan *Kitchen Dashboard* memungkinkan informasi pesanan diterima secara *real-time* sehingga proses penyajian makanan dapat dilakukan dengan lebih efisien. Pada pengembangan sistem yang dibahas dalam laporan ini, *Kitchen Dashboard* merupakan bagian dari sistem POS yang bertugas menerima dan mengelola pesanan yang telah dicatat oleh kasir. Integrasi tersebut memungkinkan *staff* dapur memperoleh informasi pesanan secara langsung tanpa memerlukan media komunikasi manual (Putu et al., 2018).

2.2 *Kitchen Management System*

Kitchen Management System (KMS) merupakan sistem yang digunakan untuk membantu pengelolaan aktivitas dapur dalam usaha kuliner. Sistem ini berfungsi sebagai penghubung antara bagian pelayanan dan bagian dapur sehingga informasi pesanan dapat diterima secara cepat dan akurat. Dengan adanya sistem tersebut, proses koordinasi antarbagian dapat dilakukan secara lebih efektif dibandingkan menggunakan metode manual.

Salah satu bentuk implementasi *Kitchen Management System* adalah *Kitchen Display System* (KDS) atau *Kitchen Dashboard*. Sistem ini menampilkan informasi pesanan pada layar digital yang ditempatkan di area dapur. Informasi yang ditampilkan umumnya meliputi nomor pesanan, nomor meja, daftar menu, catatan khusus pelanggan, dan status pengerjaan pesanan. Penggunaan *Kitchen Dashboard* mampu meningkatkan efisiensi operasional dapur karena pesanan dapat dipantau secara *real-time*. Penggunaan sistem digital juga dapat mengurangi

risiko kehilangan tiket pesanan dan kesalahan pencatatan yang sering terjadi pada sistem berbasis kertas.

2.3 REST API

Representational State Transfer Application Programming Interface (REST API) merupakan gaya arsitektur yang digunakan untuk membangun layanan web sehingga aplikasi yang berbeda dapat saling berkomunikasi melalui protokol *HTTP*. REST API memungkinkan pertukaran data dilakukan secara fleksibel menggunakan format yang ringan seperti *JSON*. Pendekatan ini menjadi salah satu standar yang paling banyak digunakan dalam pengembangan aplikasi web modern.

REST API memanfaatkan beberapa metode *HTTP* seperti *GET*, *POST*, *PUT*, *PATCH*, dan *DELETE* untuk mengelola data. Metode *GET* digunakan untuk mengambil data dari server, sedangkan *POST* digunakan untuk mengirimkan data baru. *PUT* atau *PATCH* digunakan untuk memperbarui data, sementara *DELETE* digunakan untuk menghapus data yang sudah tersimpan pada server. Pada *Kitchen Dashboard*, REST API digunakan sebagai media komunikasi antara *frontend* berbasis JavaScript dengan *backend* Laravel. Seluruh informasi pesanan diperoleh melalui *endpoint* API sehingga data dapat diperbarui secara *real-time* sesuai kondisi operasional restoran (Pangestu, Kurniadi, & Septiana, 2022).

2.4 JavaScript

JavaScript merupakan bahasa pemrograman yang digunakan untuk mengembangkan aplikasi web pada sisi *client*. Bahasa pemrograman ini memungkinkan halaman web menampilkan konten secara dinamis tanpa memerlukan proses pemuatan ulang halaman secara keseluruhan. Kemampuan tersebut menjadikan JavaScript sebagai salah satu teknologi utama dalam pengembangan antarmuka web modern. JavaScript mendukung berbagai konsep pemrograman seperti *object-oriented programming*, *asynchronous programming*, *event handling*, dan manipulasi *Document Object Model* (DOM). Fitur-fitur tersebut memungkinkan pengembang membangun aplikasi yang responsif dan interaktif. Pada *Kitchen Dashboard*, JavaScript digunakan untuk mengelola komunikasi dengan REST API, memperbarui data pesanan, serta menangani interaksi pengguna terhadap sistem (Fajrizal, Handayani, Taslim, Walhidayat, & Syahril, 2025).

2.5 Laravel

Laravel merupakan framework PHP yang menerapkan pola arsitektur *Model View Controller* (MVC). Framework ini menyediakan berbagai fitur yang mempermudah pengembangan aplikasi web seperti *routing*, *middleware*, *authentication*, *migration*, dan API *resource*. Penggunaan Laravel membantu pengembang dalam membangun *backend* yang lebih terstruktur dan mudah dipelihara. Laravel juga mendukung pengembangan REST API yang dapat digunakan oleh berbagai jenis aplikasi *frontend*. Pada sistem yang dikembangkan, Laravel digunakan untuk mengelola data pesanan serta menyediakan *endpoint* API yang digunakan oleh *Kitchen Dashboard* (Rahmawati & Sumarsono, 2024).

2.6 *Polling*

Polling merupakan teknik pengambilan data dari server yang dilakukan secara berkala dalam interval waktu tertentu. Teknik ini digunakan untuk memperoleh informasi terbaru tanpa memerlukan tindakan manual dari pengguna. *Polling* sering digunakan pada aplikasi yang membutuhkan pembaruan data secara mendekati *real-time*. Mekanisme *polling* bekerja dengan mengirimkan permintaan ke server secara berulang. Setelah data diterima, aplikasi akan memperbarui tampilan berdasarkan informasi terbaru yang diperoleh. Proses tersebut dilakukan terus menerus selama aplikasi aktif digunakan. Pada *Kitchen Dashboard*, *polling* digunakan untuk mengambil data pesanan setiap tiga detik. Dengan mekanisme tersebut, *staff* dapur dapat memperoleh informasi pesanan terbaru tanpa perlu melakukan penyegaran halaman secara manual (“Telkom University Open Library,” n.d.).

2.7 *Hash Comparison*

Hash comparison merupakan teknik yang digunakan untuk membandingkan dua kondisi data menggunakan nilai hash yang dihasilkan dari data tersebut. Apabila nilai hash yang dibandingkan berbeda, maka dapat disimpulkan bahwa terjadi perubahan data. Teknik ini sering digunakan untuk meningkatkan efisiensi proses pemrosesan data. Pada aplikasi berbasis *dashboard*, *hash comparison* dapat digunakan untuk menentukan apakah proses render ulang perlu dilakukan atau tidak. Aplikasi dapat menghindari proses render yang sebenarnya tidak diperlukan. Pendekatan tersebut membantu mengurangi penggunaan sumber daya browser dan meningkatkan performa aplikasi. Pada *Kitchen Dashboard*, *hash comparison* digunakan untuk mendeteksi perubahan data pesanan yang diperoleh melalui *polling API*. Apabila tidak

ditemukan perubahan data, sistem tidak melakukan render ulang sehingga penggunaan sumber daya menjadi lebih efisien (Septiani, Sediyono, & Rochman, 2022).

2.8 *Page Visibility API*

Page Visibility API merupakan API pada *web browser* yang memungkinkan aplikasi mengetahui apakah suatu halaman sedang dapat dilihat oleh pengguna atau berada dalam kondisi tersembunyi. API ini menyediakan properti seperti `hidden` dan `visibilityState` untuk mendeteksi status visibilitas dokumen, serta peristiwa `visibilitychange` yang dipicu ketika status visibilitas halaman berubah. Informasi tersebut memungkinkan aplikasi menyesuaikan perilakunya ketika halaman tidak sedang dilihat oleh pengguna, misalnya dengan memperlambat siklus pembaruan data atau menghentikan aktivitas tertentu hingga halaman kembali terlihat. Penggunaan *Page Visibility API* juga dapat membantu aplikasi menggunakan sumber daya perangkat secara lebih efisien karena aktivitas yang tidak diperlukan ketika halaman tersembunyi dapat dikurangi (Delgado, 2010). Penerapan *Page Visibility API* pada Kitchen Dashboard digunakan untuk mengoptimalkan aktivitas *polling* pada sisi klien ketika halaman tidak sedang digunakan, sehingga dapat mengurangi jumlah *request* yang dikirimkan selama kondisi halaman tersembunyi.

2.9 *Debounce*

Debounce merupakan teknik optimasi yang digunakan untuk membatasi frekuensi pemanggilan suatu fungsi ketika suatu peristiwa terjadi secara berulang dalam waktu singkat. Teknik ini umumnya diterapkan pada fitur pencarian, validasi input, maupun pemrosesan data secara *real-time*. Prinsip kerja *debounce* adalah menunda eksekusi fungsi hingga pengguna berhenti melakukan aktivitas tertentu selama periode waktu yang telah ditentukan (Ahyar, Wibisono, Khusnuliawati, & Charolina, 2024). Dengan pendekatan tersebut, jumlah proses yang dijalankan sistem dapat dikurangi secara signifikan. Pada *Kitchen Dashboard*, *debounce* digunakan pada fitur pencarian pesanan. Teknik ini membantu mengurangi jumlah permintaan ke server ketika pengguna mengetik kata kunci pencarian secara berulang.

2.10 IDOR

Insecure Direct Object Reference (IDOR) merupakan kerentanan keamanan yang terjadi ketika aplikasi menggunakan *identifier* yang dikirimkan pengguna untuk mengakses suatu *resource* tanpa melakukan pemeriksaan otorisasi yang memadai di sisi *backend*, sehingga

pengguna dapat mengakses atau memanipulasi *resource* yang bukan menjadi haknya hanya dengan mengubah nilai *identifier* pada *URL*, parameter, maupun *body request*. Pada Kitchen Dashboard, *identifier* pesanan digunakan dalam berbagai proses, seperti mengambil, membatalkan, dan mengubah status pesanan, sehingga setiap permintaan yang menggunakan *identifier* tersebut perlu diverifikasi oleh *backend* untuk memastikan bahwa pesanan yang diakses berada dalam lingkup kewenangan pihak yang melakukan permintaan. Validasi dilakukan pada sisi *backend* dengan memeriksa kepemilikan atau hak akses terhadap *resource*, sehingga perubahan nilai *identifier* oleh pengguna tidak secara otomatis memberikan akses terhadap pesanan lain. Penerapan mekanisme tersebut bertujuan untuk mencegah akses dan manipulasi data pesanan secara tidak sah serta menjaga kerahasiaan dan integritas *resource* pada Kitchen Dashboard (Putra, Kautsar, Hindarto, & Sumarno, 2023).

2.11 Tinjauan Pustaka

Beberapa penelitian menunjukkan bahwa penerapan *Kitchen Display System* mampu meningkatkan efisiensi komunikasi antara kasir dan dapur. Sistem yang dikembangkan umumnya memanfaatkan teknologi web untuk menampilkan daftar pesanan secara digital sehingga proses penyampaian informasi dapat dilakukan secara lebih cepat dibandingkan penggunaan tiket pesanan berbasis kertas. Selain meningkatkan kecepatan penyampaian informasi, penggunaan *dashboard* digital juga membantu mengurangi risiko kehilangan data pesanan dan kesalahan komunikasi antarbagian. Penelitian lain yang berkaitan dengan sistem *Point of Sale* berbasis web menunjukkan bahwa integrasi antara modul transaksi dan dapur mampu meningkatkan kualitas pelayanan restoran. Melalui integrasi tersebut, data pesanan yang dicatat oleh kasir dapat langsung diteruskan ke bagian dapur tanpa memerlukan proses pencatatan ulang. Pendekatan ini memungkinkan proses operasional berjalan lebih efisien karena mengurangi keterlambatan penyampaian informasi dan meminimalkan potensi kesalahan manusia.

Selain pengembangan sistem POS dan *Kitchen Display System*, beberapa penelitian juga membahas implementasi REST API sebagai media komunikasi antara *frontend* dan *backend*. Penggunaan REST API dinilai mampu meningkatkan fleksibilitas sistem karena memungkinkan pertukaran data dilakukan secara terstruktur menggunakan format JSON. Dengan pendekatan tersebut, aplikasi *frontend* dapat dikembangkan secara independen tanpa bergantung pada tampilan yang disediakan oleh *backend*. Penelitian mengenai *dashboard real-time* menunjukkan bahwa mekanisme pembaruan data secara berkala dapat meningkatkan

kemampuan pengguna dalam memantau kondisi sistem. Salah satu teknik yang sering digunakan adalah *polling*, yaitu proses pengambilan data dari server secara berkala dalam interval tertentu. Meskipun metode ini relatif sederhana untuk diimplementasikan, beberapa penelitian menunjukkan bahwa penggunaan *polling* secara terus-menerus dapat meningkatkan beban server apabila tidak disertai mekanisme optimasi yang memadai.

Berdasarkan hasil kajian terhadap penelitian terdahulu, sebagian besar penelitian berfokus pada pengembangan sistem *Point of Sale*, *Kitchen Display System*, atau *dashboard* pemantauan data secara *real-time*. Namun demikian, masih sedikit penelitian yang membahas optimasi performa *frontend* pada *Kitchen Dashboard* melalui kombinasi mekanisme *polling*, *hash comparison*, *debounce*, dan *Page Visibility* API. Sebagian besar penelitian hanya berfokus pada penyajian data secara *real-time* tanpa memperhatikan efisiensi proses render maupun penggunaan sumber daya browser.

Oleh karena itu, pengembangan *Kitchen Dashboard* pada laporan ini tidak hanya berfokus pada penyajian informasi pesanan secara *real-time*, tetapi juga menerapkan berbagai teknik optimasi untuk meningkatkan efisiensi sistem. Teknik *hash comparison* digunakan untuk membatasi proses render ulang hanya ketika terjadi perubahan data. Sistem juga memanfaatkan *debounce* untuk mengurangi jumlah permintaan pencarian ke server serta *Page Visibility API* untuk menghentikan *polling* ketika halaman tidak aktif. Kombinasi teknik tersebut diharapkan mampu menghasilkan *Kitchen Dashboard* yang responsif, ringan, dan sesuai dengan kebutuhan operasional restoran yang dinamis.

BAB III
PELAKSANAAN MAGANG

3.1 Aktivitas Magang

Magang dilaksanakan selama enam bulan di PT. Vastu Cipta Persada dengan fokus pada pengembangan proyek CAVAA berbasis web menggunakan framework Laravel, Blade sebagai *template engine*, dan Tailwind CSS untuk perancangan antarmuka pengguna. Selama periode tersebut, mahasiswa dilibatkan secara langsung dalam proses perancangan, implementasi, dan pengujian sistem, khususnya pada pengembangan *Kitchen Dashboard* sebagai komponen utama yang menjadi fokus laporan ini. Aktivitas magang secara keseluruhan dapat dilihat pada Tabel 3.1.

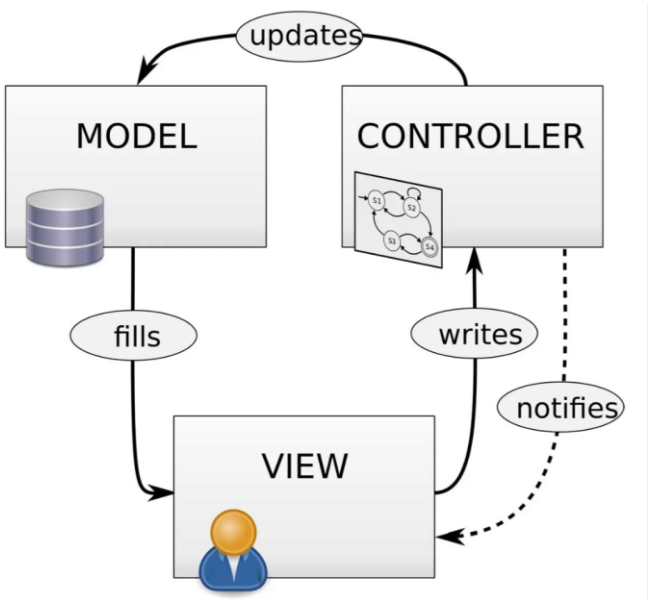
Tabel 3. 1 Aktivitas Magang.

No	Aktivitas	Durasi Waktu
1.	On-boarding magang dan analisis sistem awal	15 September 2025
2.	Perancangan dan pengembangan sistem kitchen	16 September 2025 – 31 Oktober 2025
3.	Pengembangan fitur halaman setting	3 November 2025 – 5 Desember 2025
4.	Pengembangan fitur outlet	6 Desember 2025 – 8 Desember 2025
5.	Testing sistem dan perbaikan bug	8 Desember 2025 – 19 Desember 2025
6.	Masuk Ke Proyek lain	22 Desember 2025 – 13 Maret 2026

3.1.1 *On-boarding* Magang

Pada tahap awal pelaksanaan magang, dilakukan kegiatan *on-boarding* yang bertujuan untuk memahami profil perusahaan, alur kerja tim, serta ruang lingkup proyek yang akan dikembangkan. Kegiatan ini diawali dengan sesi *briefing* bersama pembimbing proyek yang membahas tujuan pengembangan sistem, target pengguna aplikasi, serta teknologi yang digunakan dalam proses pengembangan, yaitu framework Laravel sebagai *backend*, Blade sebagai *template engine*, dan Tailwind CSS untuk perancangan antarmuka pengguna. Sistem menggunakan arsitektur *Model View Controller* (MVC) untuk memisahkan logika aplikasi, tampilan, dan pengelolaan data sehingga memudahkan pengembangan dan pemeliharaan sistem (Mardiana, Wahyudin, & Junaeti, 2024).

Setelah memperoleh gambaran umum mengenai sistem, dilakukan eksplorasi terhadap struktur folder proyek untuk memahami arsitektur yang diterapkan, khususnya pola *Model-View-Controller* (MVC). Pola MVC merupakan pendekatan arsitektur perangkat lunak yang membagi aplikasi menjadi tiga komponen utama, yaitu Model, *View*, dan *Controller*. Model berfungsi untuk mengelola data serta berinteraksi dengan basis data, *View* bertanggung jawab dalam menampilkan data kepada pengguna melalui antarmuka, sedangkan Controller berperan sebagai penghubung yang mengatur alur logika aplikasi dan memproses permintaan dari pengguna sebelum diteruskan ke Model maupun *View* ("Sistem Pemrograman Model View Controller (MVC) - Informatika Universitas Ciputra," n.d.). Sistem menerapkan pola arsitektur *Model-View-Controller* (MVC) seperti yang ditunjukkan pada Gambar 3.1.



Gambar 3. 1 Diagram MVC.

Melalui proses eksplorasi tersebut, dipelajari pembagian tanggung jawab antar direktori seperti pengelolaan logika aplikasi pada *controller*, pengolahan data pada model, serta penyajian tampilan melalui *view*. Dilakukan juga peninjauan terhadap file konfigurasi seperti *composer.json* dan *package.json* untuk memahami dependensi yang digunakan dalam pengembangan aplikasi pada sisi *backend* dan *frontend*. Analisis awal terhadap struktur basis data juga dilakukan untuk memahami skema tabel, relasi antar entitas, serta mekanisme penyimpanan data dalam sistem.

3.1.2 Penugasan dan Ruang Lingkup Magang

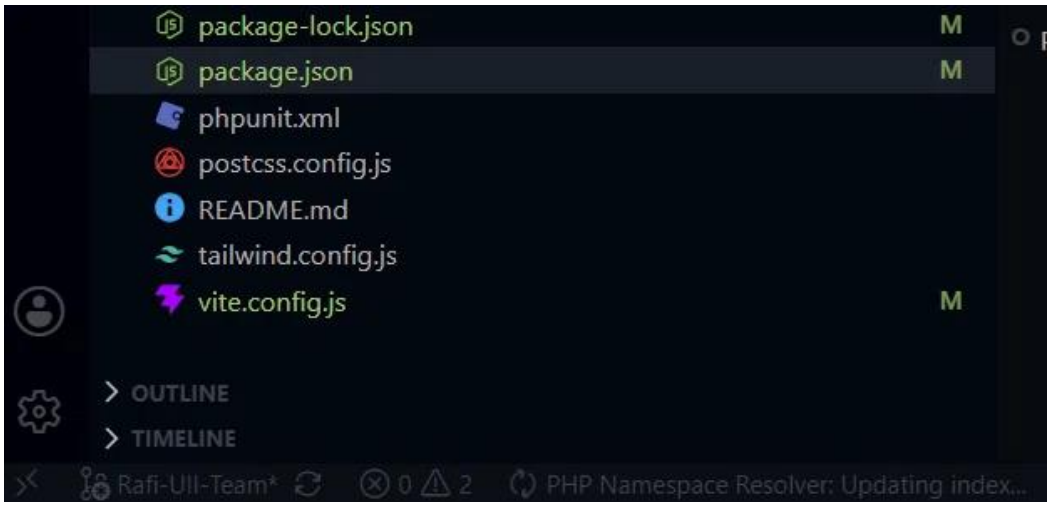
Setelah *on-boarding* selesai, mahasiswa diberikan penugasan untuk mengembangkan *Kitchen Dashboard* sebagai modul utama yang digunakan *staff* dapur dalam memantau dan

mengelola pesanan secara *real-time*. Koordinasi dilakukan melalui diskusi rutin setiap awal pekan bersama tim pengembang. *Tools* yang digunakan selama pengembangan dapat dilihat pada Tabel 3.2.

Tabel 3. 2 *Tools* yang Digunakan Selama Pengembangan.

No	Tools	Fungsi
1.	Laravel	Mengelola data pesanan dan menyediakan enpoint REST API.
2.	JavaScript	Mengelola logika frontend polling dan interaksi pengguna.
3.	Blade	Membangung struktur HTML halaman Kitchen Dashboard.
4.	Tailwind CSS	Mengatur styling dan responsivitas antarmuka
5.	Figma	Perancangan antarmuka pengguna sebelum implementasi.

Sebagai langkah awal pengerjaan, penulis melakukan inisialisasi dan pengaturan repositori kode menggunakan *platform* GitHub. Sesuai dengan arahan tim pengembang, seluruh pekerjaan yang dilakukan selama periode magang dikerjakan pada *branch* khusus dengan nama Rafi-UII-Team. Hal ini bertujuan untuk memisahkan pekerjaan mahasiswa magang dari kode utama (*main branch*) tim, sehingga proses integrasi kode dapat dilakukan secara terkendali saat kode siap untuk digabungkan. Visualisasi *branch* khusus yang digunakan sebagai wadah pengerjaan modul *Kitchen Dashboard* dapat dilihat pada Gambar 3.2.



Gambar 3. 2 Tampilan *branch* Rafi-UII-Team pada repositori proyek sebagai media pengumpulan hasil pekerjaan magang.

Setelah mengatur *branch* mahasiswa diberikan penugasan untuk mengembangkan *Kitchen Dashboard* sebagai modul utama yang digunakan *staff* dapur dalam memantau dan mengelola pesanan secara *real-time*. Koordinasi dilakukan melalui diskusi rutin setiap awal pekan bersama tim pengembang. Adapun ruang lingkup pengembangan *Kitchen Dashboard* dalam pelaksanaan magang ini adalah sebagai berikut:

a. Pengembangan antarmuka *Kitchen Dashboard* berbasis JavaScript yang teringtegrasi dengan backend laravel melalui *REST API*
b. Implementasi fitur utama *Kitchen Dashboard* yang meliputi :
    1) Panel antrian pesanan (*orders queue*) yang digunakan untuk menampilkan daftar pesanan yang baru masuk dari kasir beserta detail informasi pesanan.
    2) Panel pesanan aktif (*active orders*) yang digunakan untuk menampilakn pesanan yang sedang diproses oleh *staff* dapur dalam bentuk kartu pesanan.
    3) Panel pesanan selesai (*served orders*) yang digunakan untuk menampilkan riwayat pesanan yang telah selesai disajikan kepada pelanggan.
c. Implementasi mekanisme *polling API* dengan interval 3 detik untuk memperoleh data pesanan terbaru secara otomatis tanpa perlu memuat ulang halaman.
d. Implementasi teknik *hash comparison* untuk mendeteksi perubahan data dan membatasi proses render ulang hanya pada data yang mengalami perubahan.
e. Implementasi pengelolaan status pesanan melalui *REST API* yang meliputi :
    1) Pesanan masuk ke antrian (*paid)*
    2) Pesanan diproses (*process*)
    3) Menandai pesanan sebagai selesai disajikan (*serve*)
f. Implementasi fitur pendukung yang meliputi :
    1) Pengelolaan polling menggunakan *Page Visibility API*.
    2) Tampilan responsif berbasis CSS *Grid* untuk perangkat desktop dan mobile
    3) Status Pencarian (*debounce*)
g. Pengujian fungsionalitas sistem untuk memastikan seluruh fitur *Kitchen Dashboard* dapat berjalan sesuai dengan kebutuhan pengguna.

3.1.3 Metodologi Pengembangan Agile

Pengembangan Kitchen Dashboard menerapkan pendekatan Agile sebagai metodologi utama. Pendekatan ini dipilih karena sifat proyek yang dinamis, di mana kebutuhan pengguna dapat berubah sewaktu-waktu selama proses pengembangan. Dengan menggunakan Agile, tim dapat lebih cepat beradaptasi dan terus menghasilkan perbaikan secara bertahap.

1. Iterasi Pengembangan

Pengerjaan sistem dilakukan secara bertahap dalam beberapa siklus iterasi. Setiap iterasi berlangsung selama dua minggu dan diawali dengan sesi perencanaan bersama seluruh tim pengembang serta pemilik proyek. Pada sesi ini, ditentukan fitur-fitur mana yang menjadi prioritas untuk dikerjakan terlebih dahulu. Adapun tahapan pengembangan Kitchen Dashboard adalah sebagai berikut:

- Iterasi 1 : Analisis Kebutuhan, perancangan arsitektur dan desain antarmuka Figma
- Iterasi 2 : Implementasi tampilan dasar (layout, grid, styling) dan integrasi blade
- Iterasi 3 : Implementasi sisi backend (controller, model, dan REST API)
- Iterasi 4 : Pengembangan fitur polling dan optimasi sistem (hash comparison dan debounce)
- Iterasi 5 : Pengujian fungsionalitas dan perbaikan bug

2. Koordinasi Tim

Koordinasi antar anggota tim pengembang dilakukan setiap hari Senin melalui *meeting*. Dalam rapat mingguan ini, dibahas beberapa hal pokok, yaitu:

1. Evaluasi hasil pekerjaan dari minggu sebelumnya.
2. Rencana pekerjaan yang akan dikerjakan selama minggu berjalan.
3. Kendala atau hambatan teknis yang dihadapi serta solusi yang ditawarkan.

Rapat ini juga menjadi ajang untuk menyelaraskan pemahaman antar anggota tim mengenai tujuan pengembangan serta memastikan bahwa seluruh pekerjaan berjalan sesuai dengan prioritas yang telah ditentukan. Jika ditemukan perubahan kebutuhan dari pemilik proyek, hal tersebut langsung dibahas dan dijadwalkan pada iterasi berikutnya.

3. Evaluasi Berkala

Di akhir setiap iterasi, dilakukan dua kegiatan evaluasi:

- Demonstrasi Hasil: Tim mempertunjukkan fitur-fitur yang telah selesai dikembangkan kepada pemilik proyek dan pemangku kepentingan lainnya. Masukan yang diterima kemudian digunakan sebagai bahan perbaikan untuk iterasi berikutnya.

- Perbaikan Proses Kerja: Tim melakukan refleksi internal mengenai proses yang telah berjalan. Dibahas hal-hal apa saja yang sudah berjalan baik dan apa yang perlu ditingkatkan. Beberapa hasil perbaikan yang diterapkan antara lain:
  - Peningkatan dokumentasi API untuk memudahkan integrasi.
  - Penambahan waktu khusus untuk pengujian di setiap iterasi.
  - Penyederhanaan struktur kode frontend agar lebih mudah dipelihara.

4. Keunggulan Penerapan Agile pada Proyek

Penerapan Agile memberikan beberapa keuntungan selama pengembangan Kitchen Dashboard:

1. **Fleksibel terhadap Perubahan**: Selama pengembangan, terjadi beberapa perubahan permintaan dari pemilik proyek, seperti penambahan fitur debounce pada pencarian dan penyesuaian tampilan mobile. Pendekatan Agile memungkinkan perubahan tersebut diakomodasi tanpa mengganggu jadwal keseluruhan secara berarti.
2. **Umpan Balik yang Cepat**: Dengan adanya evaluasi rutin setiap dua minggu, tim memperoleh masukan secara teratur sehingga kesalahan atau ketidaksesuaian dapat segera diperbaiki.
3. **Hasil Terlihat Lebih Cepat**: Fitur-fitur dasar dashboard dapat diselesaikan dan diperlihatkan sejak iterasi kedua, sehingga pemangku kepentingan dapat melihat kemajuan proyek secara nyata lebih awal.
4. **Kualitas Kode Terjaga**: Melalui perbaikan kode (refactoring) yang dilakukan secara berkala, struktur program terus diperbaiki sehingga sistem menjadi lebih stabil dan mudah dikembangkan di masa depan.

Dengan pendekatan Agile, pengembangan Kitchen Dashboard menjadi lebih terarah, responsif terhadap kebutuhan, dan menghasilkan perangkat lunak yang benar-benar sesuai dengan operasional dapur.

5. Kekurangan dan Tantangan Penerapan Agile

Meskipun pendekatan Agile memberikan banyak keuntungan, terdapat beberapa kendala yang dirasakan selama proses pengembangan Kitchen Dashboard:

1. Kurangnya Dokumentasi yang Memadai

Agile menekankan pada working software daripada dokumentasi yang komprehensif, tim cenderung mengabaikan pembuatan dokumentasi teknis secara detail. Hal ini menyulitkan ketika ada anggota tim baru yang bergabung di tengah proyek atau ketika

ingin melacak keputusan teknis yang telah dibuat sebelumnya. Beberapa kali terjadi kebingungan mengenai alasan di balik suatu implementasi karena tidak tercatat dengan baik.

2. **Ketergantungan Tinggi pada Komunikasi dan Kehadiran Tim**

   Agile sangat bergantung pada komunikasi intensif antar anggota tim. Ketika salah satu anggota tim tidak dapat hadir atau komunikasi terhambat, proses pengambilan keputusan menjadi melambat. Dalam proyek ini, terdapat beberapa kali keterlambatan karena ketidaktersediaan pemilik proyek untuk sesi planning atau review, sehingga prioritas fitur menjadi kurang jelas.

3. **Perubahan Kebutuhan yang Terlalu Sering**

   Meskipun fleksibilitas adalah keunggulan Agile, perubahan kebutuhan yang terlalu sering dapat menghambat progres. Pada proyek ini, terjadi beberapa kali perubahan spesifikasi pada tampilan Kitchen Dashboard yang memaksa tim untuk mengulang pekerjaan yang telah selesai. Misalnya, desain kartu pesanan diubah sebanyak tiga kali selama iterasi, yang menyebabkan pemborosan waktu pengembangan.

4. **Kesulitan dalam Estimasi Waktu**

   Dengan pendekatan iteratif, estimasi waktu untuk menyelesaikan fitur sering kali kurang akurat. Fitur yang awalnya diperkirakan selesai dalam satu iterasi (dua minggu) ternyata membutuhkan waktu lebih lama karena kompleksitas teknis yang baru diketahui di tengah pengerjaan. Hal ini menyebabkan beberapa fitur harus ditunda ke iterasi berikutnya.

5. **Tekanan untuk Menyelesaikan di Setiap Iterasi**

   Tuntutan untuk menghasilkan fitur yang siap digunakan di akhir setiap iterasi menimbulkan tekanan tersendiri. Tim kadang harus mengorbankan kualitas kode atau mengurangi cakupan pengujian demi memenuhi deadline iterasi. Akibatnya, beberapa bug baru ditemukan pada iterasi berikutnya, yang sebenarnya bisa dicegah dengan pengujian yang lebih mendalam.

6. **Kurangnya Keterlibatan Pengguna Akhir**

   Dalam Agile, keterlibatan pengguna akhir sangat penting untuk memberikan umpan balik. Namun pada proyek ini, keterbatasan akses ke staff dapur sebagai pengguna akhir menyebabkan umpan balik yang diperoleh tidak sepenuhnya mencerminkan kebutuhan operasional sebenarnya. Hal ini berisiko menghasilkan sistem yang secara teknis berfungsi tetapi kurang sesuai dengan alur kerja dapur yang sesungguhnya.

3.2 Analisis dan Perancangan Sistem Kitchen Dashboard

3.2.1 Analisis Kebutuhan Sistem

Tahap analisis kebutuhan sistem dilakukan untuk memahami kebutuhan pengguna serta menentukan fitur-fitur yang harus disediakan oleh *Kitchen Dashboard*. Analisis dilakukan melalui diskusi dengan pembimbing proyek dan tim pengembang untuk mengidentifikasi permasalahan yang terjadi dalam proses koordinasi antara kasir dan dapur.

Hasil analisis ini memperoleh beberapa kebutuhan utama sistem. Sistem harus mampu menampilkan data pesanan secara *real-time* agar *staff* dapur dapat langsung mengetahui pesanan yang masuk tanpa menunggu informasi dari kasir secara manual. Sistem juga harus menyediakan mekanisme pengelolaan status pesanan sehingga *staff* dapur dapat memperbarui status pengerjaan pesanan secara langsung. Kebutuhan performa juga menjadi pertimbangan penting karena sistem menggunakan mekanisme *polling* yang berjalan terus-menerus, sehingga diperlukan teknik optimasi untuk menghindari proses render yang tidak diperlukan serta mengurangi beban server. Hasil analisis kebutuhan pengguna dapat dilihat pada Tabel 3.3.

Tabel 3. 3 Hasil Analisis Kebutuhan Pengguna.

No	Kebutuhan Sistem	Deskripsi	Tujuan
1.	Menampilkan daftar pesanan	Sistem menampilkan pesanan masuk secara real-time	Mempercepat proses kerja dapur
2.	Informasi detail pesanan	Menampilkan detail pesanan	Meminimalisir kesalahan pengerjaan
3.	Status pesanan	Menyediakan status antrian, diproses, dan selesai	Memantau proses pesanan
4.	Perubahan status langsung	Staff dapat mengubah status pesanan secara langsung	Meningkatkan efisiensi operasional
5.	Optimasi performa	Hash comparison dan Page Visibility API	Mengurangi beban server dan browser
6.	Tampilan responsif	UI dapat digunakan pada desktop, tablet maupun mobile	Fleksibilitas penggunaan sistem

3.2.2 Analisis Kebutuhan Fungsional

Analisis alur proses bisnis dilakukan untuk memahami bagaimana pesanan berpindah dari satu tahapan ke tahapan berikutnya dalam operasional restoran. Alur dimulai ketika kasir

memasukkan pesanan pelanggan ke dalam sistem POS. Setelah pesanan berhasil dicatat, sistem secara otomatis mengirimkan data pesanan ke antrian Kitchen Dashboard melalui REST API.

Staff dapur dapat mengelola pesanan melalui beberapa aktivitas utama. Dari daftar antrian pesanan (status PAID), staff dapur dapat melihat detail pesanan, mengambil pesanan untuk diproses (status menjadi PROCESSED), atau membatalkan pesanan (status menjadi CANCEL). Pada panel pesanan aktif, staff dapur dapat memantau pekerjaan berlangsung, menandai setiap item yang selesai, menandai pesanan selesai (status menjadi SERVED), atau membatalkan pesanan aktif sehingga kembali ke antrian (status menjadi PAID).

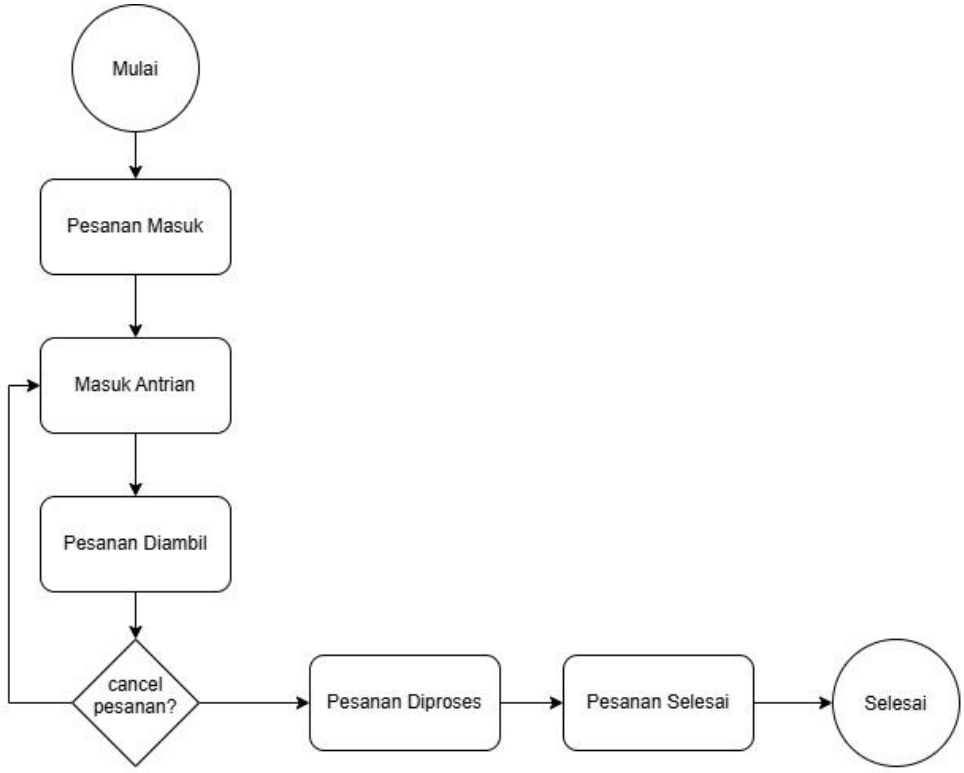
Selain itu, staff dapur dapat melihat riwayat pesanan selesai, mencari pesanan berdasarkan kata kunci, serta memfilter pesanan berdasarkan status. Seluruh aktivitas tersebut direpresentasikan dalam use case diagram pada Gambar 3.3. Pemetaan alur status pesanan dapat dilihat pada Tabel 3.4 serta flowchart alur pemesanan dapat dilihat pada Gambar 3.4.

Tabel 3. 4 Pemetaan Alur Status Pesanan.

No	Tahapan Proses	Role	Deskripsi	Status Sistem
1.	Input Pesanan	Cashier	Kasir memasukkan pesanan pelanggan ke sistem	Pesanan masuk
2.	Antrian Pesanan	Kitchen	Sistem menampilkan pesanan ke antrian dapur	Menunggu
3.	Ambil Pesanan	Kitchen	Kitchen mengambil pesanan untuk mulai diproses	Diproses
4.	Pembatalan Pesanan	Kitchen	Pesanan dibatalkan dan dikembalikan ke antrian	Menunggu
5.	Penyelesaian Pesanan	Kitchen/Cashier	Pesanan ditandai selesai setelah proses memasak selesai	Selesai



Gambar 3. 3 *Use Case Diagram Kitchen Dashboard*



Gambar 3. 4 Flowchart Alur Pemesanan *Kitchen Dashboard.*

3.2.3 Analisis Kebutuhan Non-Fungsional

Sistem harus memperbarui data pesanan secara berkala tanpa interaksi manual dari pengguna. Tampilan harus dapat menyesuaikan diri pada berbagai ukuran layar, baik desktop, tablet, maupun mobile. Hanya pengguna dengan role *kitchen* yang dapat mengakses dan mengoperasikan *dashboard*, diatur melalui *middleware* autentikasi. Mekanisme pembaruan data tidak boleh membebani server maupun browser secara berlebihan, sehingga diperlukan teknik optimasi tambahan. Rincian kebutuhan non-fungsional dapat dilihat pada Tabel 3.5.

Tabel 3. 5 Kebutuhan Non-Fungsional.

No	Kebutuhan Non-Fungsional	Deskripsi
1.	Real-time	Data diperbarui otomatis
2.	Responsif	Tampilan menyesuaikan ukuran layar perangkat
3.	Keamanan akses	Middleware membatasi akses hanya untuk role kitchen
4.	Performa sistem	Render hanya terjadi ketika benar-benar berubah

3.2.4 Perancangan Arsitektur Sistem

1. Perancangan Arsitektur Sistem

Perancangan arsitektur sistem dilakukan untuk menggambarkan hubungan antar komponen yang terlibat dalam proses pengelolaan pesanan pada *Kitchen Dashboard*. Arsitektur sistem dirancang menggunakan pendekatan tiga lapisan (*three-tier architecture*) yang terdiri dari lapisan *frontend*, *backend*, dan basis data. Pendekatan ini dipilih karena mampu memisahkan tanggung jawab setiap komponen sistem sehingga proses pengembangan, pemeliharaan, dan pengujian aplikasi dapat dilakukan dengan lebih mudah.

a. Frontend

Lapisan frontend berfungsi sebagai antarmuka yang digunakan oleh staff dapur untuk memantau dan mengelola pesanan pelanggan. Frontend dibangun menggunakan Blade sebagai *template engine* pada framework Laravel, Tailwind CSS untuk mengatur tampilan antarmuka, serta JavaScript untuk mengelola logika aplikasi pada sisi klien. Seluruh informasi pesanan ditampilkan dalam bentuk *dashboard* yang terdiri dari panel antrian pesanan, pesanan aktif, dan pesanan selesai. JavaScript digunakan untuk menangani proses pengambilan data dari server secara berkala melalui mekanisme polling, melakukan pembaruan tampilan ketika terdapat perubahan data, serta memproses interaksi pengguna seperti mengambil pesanan, membatalkan pesanan, dan menyelesaikan pesanan. Dengan memanfaatkan JavaScript, proses pembaruan data dapat dilakukan tanpa perlu memuat ulang halaman sehingga pengalaman pengguna menjadi lebih baik (Pranantya, 2025).

b. Backend

Lapisan *backend* dibangun menggunakan framework Laravel yang berfungsi sebagai pusat pengelolaan logika bisnis sistem. *Backend* bertanggung jawab menerima permintaan dari frontend, memproses data yang diperlukan, berinteraksi dengan basis data, dan mengembalikan hasil dalam format *JSON* melalui REST API.Selain menyediakan *endpoint* API untuk pengambilan data pesanan, *backend* juga menangani perubahan status pesanan yang dilakukan oleh *staff* dapur. Setiap aksi yang dilakukan pengguna akan diproses terlebih dahulu pada *backend* untuk memastikan data yang tersimpan tetap konsisten. Laravel juga menyediakan *middleware* autentikasi dan otorisasi yang digunakan untuk membatasi akses hanya kepada pengguna yang memiliki hak akses sebagai *staff* dapur (Nurjaman, Utomo, & Hermanto, 2024).

c. Basisdata

Lapisan basisdata berfungsi sebagai tempat penyimpanan seluruh informasi yang digunakan dalam operasional sistem. Pengelolaan basis data dilakukan menggunakan *Eloquent ORM* yang merupakan fitur bawaan Laravel. Melalui *Eloquent ORM*, proses

pengambilan dan pengolahan data dapat dilakukan menggunakan model sehingga kode program menjadi lebih terstruktur dan mudah dipahami. Basis data menyimpan berbagai informasi yang berkaitan dengan operasional restoran, seperti data pesanan, detail item pesanan, data pelanggan, informasi meja, serta status pesanan. Relasi antar tabel dirancang untuk memastikan setiap data dapat saling terhubung dengan baik sehingga proses pengambilan informasi dapat dilakukan secara efisien (Anggraeni & Susetyo, 2025).

2. Perancangan Komunikasi REST API

Komunikasi antara frontend dan backend dirancang menggunakan pendekatan *Representational State Transfer Application Programming Interface* (REST API). Pendekatan ini dipilih karena mampu memisahkan proses pengelolaan data pada *backend* dengan proses penyajian antarmuka pada *frontend*. Dengan menggunakan REST API, pertukaran data dapat dilakukan secara fleksibel menggunakan format JSON yang ringan dan mudah diproses oleh aplikasi (Maulana, Sulistyanto, & Asih Septia Rini, 2021).

Pada *Kitchen Dashboard*, seluruh data pesanan diperoleh melalui *endpoint API* yang disediakan oleh *backend* Laravel. *Frontend* mengirimkan permintaan HTTP ke *endpoint* tertentu untuk memperoleh data pesanan maupun melakukan perubahan status pesanan. Setelah permintaan diterima, backend memproses data yang diperlukan dan mengembalikan respons dalam format *JSON*. *Endpoint API* yang dirancang terdiri dari *endpoint* untuk mengambil data antrian pesanan, pesanan aktif, pesanan selesai, serta *endpoint* yang digunakan untuk mengubah status pesanan. Daftar *endpoint* yang digunakan pada *Kitchen Dashboard* dapat dilihat pada Tabel 3.6.

Tabel 3. 6 *Endpoint* REST API *Kitchen Dashboard.*

No	Endpoint	Method	Fungsi
1.	kitchen/orders/queue	GET	Mengambil datar antrian pesanan
2.	kitchen/orders/active	GET	Mengambil daftar pesanan akitf
3.	kitchen/orders/served	GET	Mengambil daftar pesanan selesai
4.	kitchen/orders/pickup/:id	PUT	Mengubah status pesanan menjadi diproses
5.	kitchen/orders/serve/:id	PUT	Mengubah status pesanan menjadi selesai
6.	kitchen/orders/cancel/:id	PUT	Mengembalikan pesanan aktif ke antrian

Perancangan komunikasi API ini memungkinkan proses pertukaran data berlangsung secara *real-time* karena *frontend* dapat melakukan pengambilan data secara berkala tanpa perlu melakukan penyegaran halaman secara manual.

3. Perancangan Struktur Data

Perancangan struktur data dilakukan untuk menentukan informasi apa saja yang perlu disimpan dan dipertukarkan antara *backend* dan *frontend*. Struktur data yang baik diperlukan agar proses pengolahan data dapat dilakukan secara efisien dan konsisten pada seluruh bagian sistem.

A. Struktur Data Pesanan

Objek pesanan dirancang untuk memuat seluruh informasi yang dibutuhkan *staff* dapur selama proses pengerjaan pesanan. Setiap pesanan memiliki atribut utama seperti identitas pesanan, informasi pelanggan, waktu pemesanan, status pesanan, dan daftar item yang dipesan. Data pesanan terdiri dari beberapa atribut penting, antara lain:

- *id* sebagai identitas unik pesanan.
- *booking_order_code* sebagai kode pesanan.
- *customer_name* sebagai nama pelanggan.
- *order_time* sebagai waktu pemesanan.
- *total_items* sebagai jumlah item pesanan.
- *table* sebagai informasi meja pelanggan.
- *cashier_process_id* sebagai indikator bahwa pesanan sedang diproses oleh kasir.
- *order_details* yang berisi daftar item makanan atau minuman yang dipesan.

Struktur data tersebut dirancang agar seluruh informasi penting dapat ditampilkan secara lengkap pada kartu pesanan di *Kitchen Dashboard*.

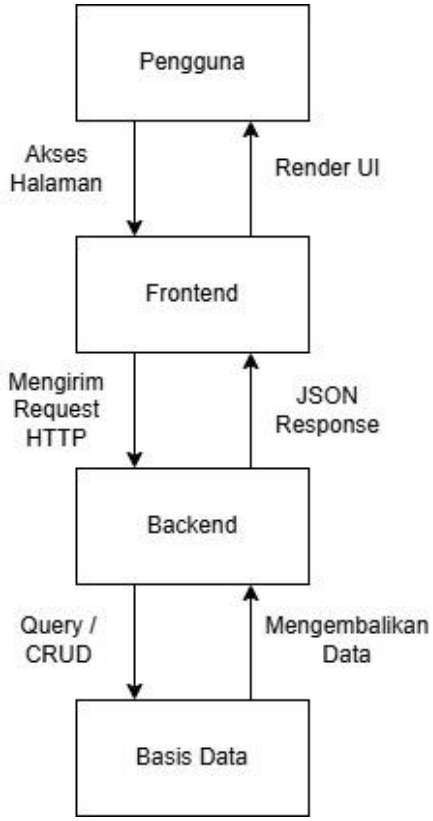
B. Struktur Respon API

Untuk mempermudah proses pengolahan data pada *frontend*, seluruh *endpoint* API dirancang menggunakan format respon yang konsisten. Setiap respon API memiliki struktur yang sama sehingga *frontend* tidak perlu menggunakan logika yang berbeda untuk setiap *endpoint*. Struktur respon yang digunakan terdiri dari *field success* yang menunjukkan status keberhasilan permintaan dan *field* data yang berisi informasi yang diminta oleh pengguna. Apabila terjadi kesalahan, sistem juga dapat mengirimkan informasi pesan kesalahan melalui *field message* sehingga proses penanganan *error* dapat dilakukan dengan lebih mudah. Konsistensi struktur respon ini membantu meningkatkan keterbacaan kode serta mempermudah proses integrasi antara *frontend* dan *backend* (Evry Nazyli Ciptanto, 2024).

4. Diagram Arsitektur Sistem

Berdasarkan perancangan yang telah dilakukan, hubungan antara *frontend*, *backend*, dan basis data divisualisasikan dalam bentuk diagram arsitektur sistem. Diagram tersebut

menunjukkan alur komunikasi data mulai dari pengguna yang mengakses *Kitchen Dashboard* melalui browser, kemudian *frontend* mengirimkan permintaan ke *backend* melalui *REST API*. Selanjutnya *backend* memproses permintaan tersebut dengan mengambil atau memperbarui data pada basis data sebelum mengembalikan hasilnya ke *frontend* dalam format *JSON*. Melalui arsitektur tersebut, setiap komponen memiliki tanggung jawab yang jelas sehingga sistem menjadi lebih terstruktur, mudah dikembangkan, dan mampu mendukung proses pembaruan data pesanan secara *real-time* (Rozzi Pratama, Budi, Kom, Robbani, & Md, 2023). Diagram arsitektur sistem *Kitchen Dashboard* dapat dilihat pada Gambar 3.5.



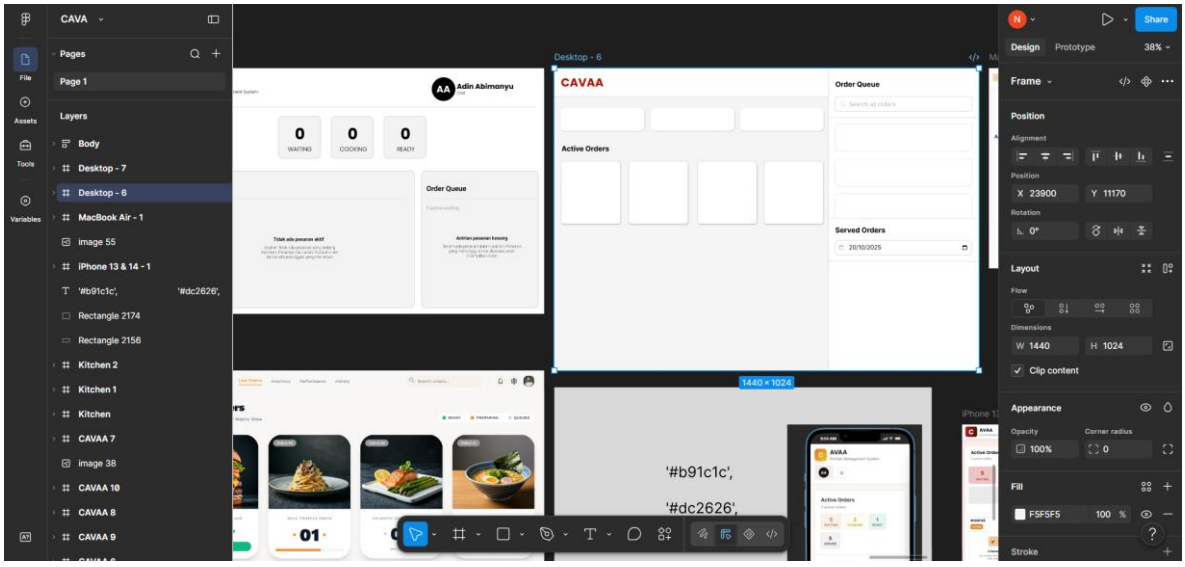
Gambar 3. 5 Diagram Arsitektur Sistem *Kitchen Dashboard*.

3.2.5 Perancangan Antarmuka Pengguna

Perancangan antarmuka pengguna *Kitchen Dashboard* dilakukan menggunakan Figma sebagai alat desain visual. Pada tahap ini, dibuat rancangan tampilan *dashboard* yang menampilkan tiga panel utama secara bersamaan dalam satu layar, yaitu panel antrian pesanan, panel pesanan aktif, dan panel pesanan selesai. Setiap panel dirancang dengan tata letak yang jelas agar *staff* dapur dapat membedakan status pesanan secara visual dengan cepat.

Setiap kartu pesanan dirancang memuat informasi yang paling sering dibutuhkan *staff* dapur, meliputi kode pesanan, nama pelanggan, nomor meja, waktu pemesanan, jumlah item,

serta daftar menu. Tombol aksi ditempatkan pada posisi yang mudah dijangkau. Pada rancangan tampilan mobile, disediakan komponen *sidebar* geser dengan tab navigasi yang memisahkan tampilan antrian pesanan dan riwayat pesanan selesai karena keterbatasan ruang layar pada perangkat mobile yang tidak memungkinkan ketiga panel ditampilkan sekaligus. Perancangan antarmuka dilakukan menggunakan Figma dengan konsep tiga panel utama dalam satu layar. Hasil desain Figma dapat dilihat pada Gambar 3.6, yang kemudian dijadikan acuan selama proses implementasi.



Gambar 3. 6 Hasil Desain Figma *Kitchen Dashboard*.

3.3 Realisasi dan Pengembangan Sistem *Kitchen Dashboard*

3.3.1 Implementasi *Frontend* Menggunakan Blade dan Tailwind CSS

Implementasi *frontend* merupakan tahap pengembangan antarmuka pengguna yang berfungsi sebagai media interaksi antara staf dapur dengan sistem *Kitchen Dashboard*. Pada tahap ini, rancangan antarmuka yang telah dibuat sebelumnya direalisasikan menggunakan Blade sebagai *template engine* Laravel, Tailwind CSS sebagai framework *styling*, serta JavaScript untuk menangani interaksi dan pembaruan data secara dinamis.

Penggunaan Blade dipilih karena mampu terintegrasi secara langsung dengan framework Laravel sehingga proses pengiriman data dari backend ke frontend dapat dilakukan dengan lebih sederhana. Blade juga menyediakan fitur *reusable component* yang memungkinkan bagian antarmuka yang sering digunakan, seperti sidebar dan navbar, dapat digunakan kembali tanpa perlu menuliskan kode yang sama berulang kali.

Sementara itu, Tailwind CSS digunakan untuk membangun tampilan yang responsif dan konsisten pada berbagai ukuran perangkat. Pendekatan utility-first yang diterapkan Tailwind CSS mempermudah proses pengembangan antarmuka karena sebagian besar kebutuhan styling dapat diterapkan langsung pada elemen HTML. Dengan kombinasi Blade dan Tailwind CSS, Kitchen Dashboard dapat menampilkan informasi pesanan secara terstruktur, responsif, dan mudah digunakan oleh staf dapur dalam kondisi operasional yang dinamis.

A. Implementasi Layout Dashboard

Layout dashboard dirancang sebagai kerangka utama yang menampung seluruh komponen antarmuka Kitchen Dashboard. Struktur layout dibagi menjadi tiga bagian utama, yaitu sidebar, navbar, dan content area. Pembagian ini bertujuan untuk memisahkan fungsi navigasi, informasi pengguna, dan area kerja utama sehingga antarmuka menjadi lebih terorganisasi dan mudah dipahami.

1) Sidebar

Sidebar berfungsi sebagai panel navigasi utama yang memungkinkan pengguna berpindah antar halaman yang tersedia pada sistem. Selain menyediakan menu navigasi, sidebar juga menampilkan indikator jumlah pesanan yang masih berada dalam antrian sehingga *staff* dapur dapat mengetahui kondisi operasional secara cepat tanpa harus membuka halaman lain.

2) Navbar

Navbar ditempatkan pada bagian atas halaman dan berfungsi untuk menampilkan informasi pengguna yang sedang login. Navbar juga menyediakan tombol *refresh* manual yang dapat digunakan untuk memuat ulang data apabila diperlukan. Keberadaan navbar membantu pengguna memperoleh informasi status sistem secara cepat dan mudah diakses.

3) Content Area

Content area merupakan area utama yang digunakan untuk menampilkan seluruh informasi pesanan. Pada *Kitchen Dashboard*, area ini menampilkan tiga panel utama, yaitu *Queue Orders*, *Active Orders*, dan *Served Orders*. Ketiga panel tersebut disusun menggunakan struktur Blade yang bersifat *reusable* sehingga memudahkan proses pengembangan dan pemeliharaan sistem. Potongan kode *layout* dasar Blade dapat dilihat pada Gambar 3.7.

```
<div class="flex h-screen">
    @include('layouts.sidebar')
    <div class="flex-1 flex flex-col">
        @include('layouts.navbar')
        <main class="flex-1 overflow-auto p-4">
```

```
            @yield('content')
        </main>
    </div>
</div>
```

Gambar 3. 7 Potongan Kode *Layout* Blade.

Potongan kode pada Gambar 3.6 menunjukkan struktur dasar *layout* yang digunakan pada *Kitchen Dashboard*. Komponen sidebar dan navbar dipanggil menggunakan *directive @include* sehingga dapat digunakan kembali pada halaman lain tanpa perlu menduplikasi kode. Sementara itu, *directive @yield*('*content*') digunakan sebagai *placeholder* untuk menampilkan konten utama yang berbeda pada setiap halaman. Pendekatan ini membuat struktur kode menjadi lebih modular, mudah dipelihara, dan mempermudah pengembangan fitur baru di masa mendatang.

B. Implementasi Grid Pesanan

Salah satu kebutuhan utama *Kitchen Dashboard* adalah kemampuan menampilkan banyak pesanan secara bersamaan tanpa mengurangi keterbacaan informasi. Oleh karena itu, digunakan CSS *Grid* untuk mengatur tata letak kartu pesanan agar dapat menyesuaikan ukuran layar secara otomatis. *Grid* diterapkan khusus pada panel *Active Orders* karena panel ini merupakan area yang paling sering dipantau oleh *staff* dapur selama proses operasional. Dengan menggunakan CSS *Grid*, sistem dapat menampilkan jumlah kolom yang berbeda sesuai ukuran perangkat yang digunakan. Potongan kode implementasi *grid* dapat dilihat pada Gambar 3.8.

```
@media (max-width: 380px) {
    #activeOrders { grid-template-columns: 1fr; }
}
@media (min-width: 768px) {
    #activeOrders { grid-template-columns: repeat(3, 1fr); }
}
@media (min-width: 1024px) {
    #activeOrders { grid-template-columns: repeat(4, 1fr); }
}
```

Gambar 3. 8 Potongan Kode *Grid.*

Kode tersebut menunjukkan penggunaan *breakpoint* responsif untuk mengatur jumlah kolom yang ditampilkan. Pada perangkat mobile dengan lebar layar kecil, sistem hanya menampilkan satu kolom agar isi kartu tetap mudah dibaca. Ketika ukuran layar bertambah menjadi tablet, jumlah kolom meningkat menjadi tiga kolom. Pada perangkat desktop, sistem menampilkan empat kolom sehingga ruang layar dapat dimanfaatkan secara maksimal.

Pendekatan ini memastikan antarmuka tetap nyaman digunakan pada berbagai ukuran perangkat.

C. Implementasi Styling dengan Tailwind CSS

Setelah struktur layout dan grid selesai dikembangkan, tahap berikutnya adalah penerapan *styling* menggunakan Tailwind CSS. Framework ini dipilih karena menyediakan berbagai *utility class* yang dapat digunakan untuk mengatur tata letak, warna, tipografi, ukuran elemen, animasi, serta responsivitas tampilan. Penggunaan *utility class* memungkinkan proses styling dilakukan secara konsisten di seluruh bagian *dashboard*. Tailwind CSS juga membantu mengurangi jumlah kode CSS kustom yang harus dibuat sehingga proses pengembangan menjadi lebih efisien. Penggunaan *utility* Tailwind CSS pada *Kitchen Dashboard* dapat dilihat pada Tabel 3.7.

Tabel 3. 7 Tabel *Utility* Tailwind CSS.

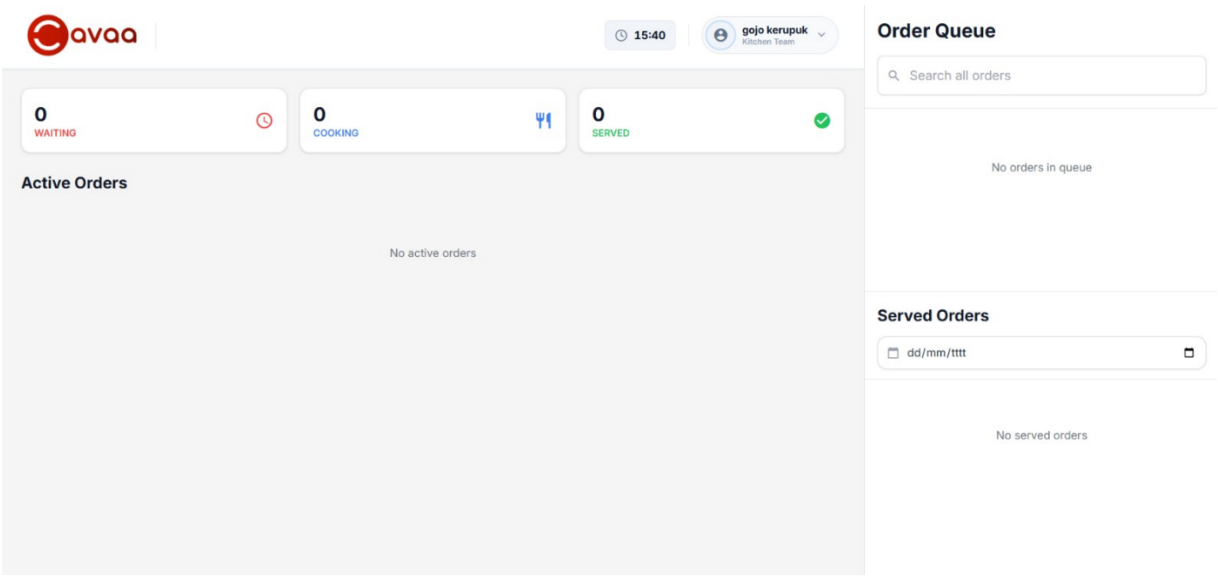
No	Elemen UI	Utility Tailwind CSS	Tampilan Mobile	Tampilan Desktop
1.	Layout Grid	grid-cols-1 lg:grid-cols-4	1 Kolom	4 Kolom
2.	Sidebar	hidden md:block	Disembunyikan	Ditampilkan penuh
3.	Card Pesanan	w-full md:w-auto	Lebar penuh	Menyesuaikan kontainer
4.	Animasi	transition-all duration-300	Tetap aktif	Tetap aktif
5.	Tab Mobile	block md:hidden	Ditampilkan	Disembunyikan

Berdasarkan tabel tersebut, sebagian besar komponen antarmuka memanfaatkan *utility class* responsif untuk menyesuaikan tampilan antara perangkat mobile dan desktop. Sebagai contoh, sidebar disembunyikan pada perangkat mobile untuk menghemat ruang layar, sedangkan pada perangkat desktop sidebar ditampilkan secara penuh. Animasi transisi juga diterapkan untuk memberikan pengalaman pengguna yang lebih interaktif saat terjadi perubahan data.

D. Hasil Implementasi Frontend

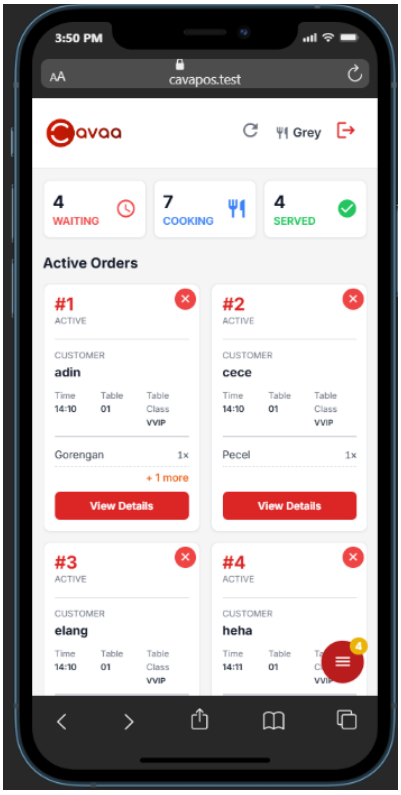
Setelah seluruh komponen *frontend* berhasil diimplementasikan, dilakukan pengujian tampilan pada berbagai ukuran perangkat untuk memastikan seluruh elemen antarmuka berfungsi dengan baik. Hasil implementasi menunjukkan bahwa *dashboard* mampu menampilkan informasi pesanan secara jelas baik pada perangkat desktop maupun perangkat

mobile. Pada tampilan desktop, ketiga panel utama dapat ditampilkan secara bersamaan sehingga *staff* dapur dapat memantau seluruh status pesanan dalam satu layar. Sementara itu, pada tampilan mobile, tata letak disesuaikan agar tetap nyaman digunakan pada layar yang lebih kecil tanpa mengurangi fungsi utama sistem. Gambar 3.9 menunjukkan tampilan *Kitchen Dashboard* pada perangkat desktop. Tampilan ini menampilkan tiga panel utama secara bersamaan, yaitu panel *Queue Orders* (antrian pesanan), *Active Orders* (pesanan aktif), dan *Served Orders* (pesanan selesai). Pada bagian atas dashboard terdapat indikator jumlah pesanan untuk setiap status yang memudahkan *staff* dapur dalam memantau kondisi operasional secara cepat.



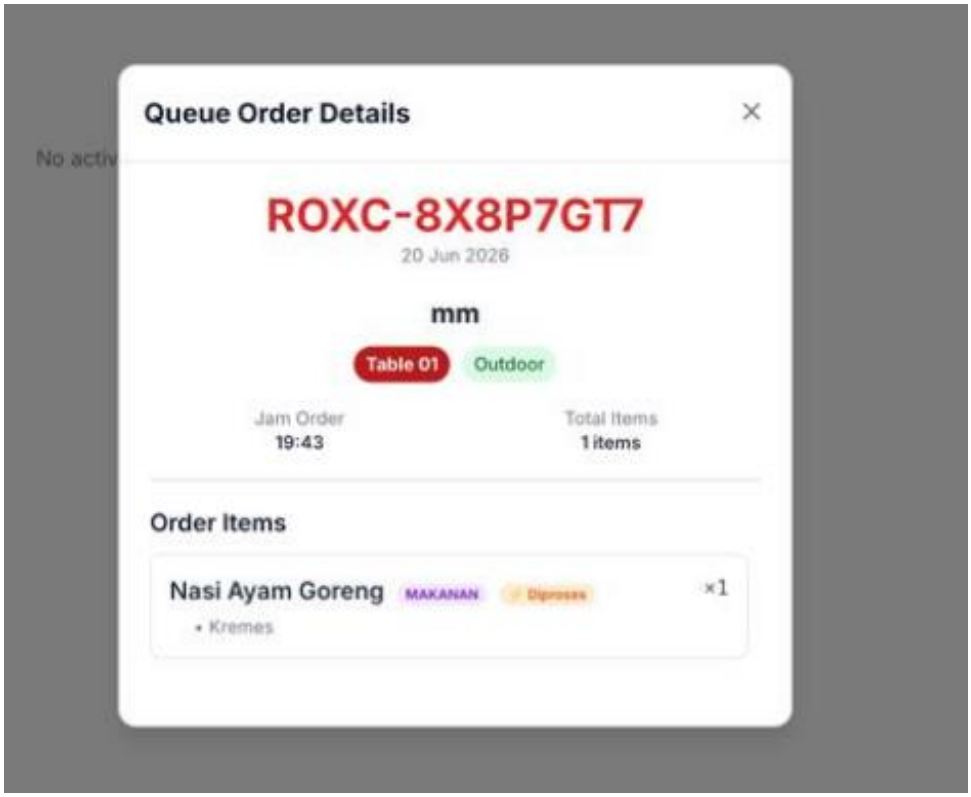
Gambar 3. 9 Tampilan Desktop *Kitchen Dashboard.*

Selain tampilan desktop, Kitchen Dashboard juga dirancang responsif untuk mendukung penggunaan pada perangkat mobile. Pada tampilan mobile, sidebar berubah menjadi off-canvas sidebar yang dapat diakses melalui tombol hamburger, sedangkan ketiga panel utama disusun menggunakan tab navigasi untuk menghemat ruang layar. Hasil tampilan mobile dapat dilihat pada Gambar 3.10.



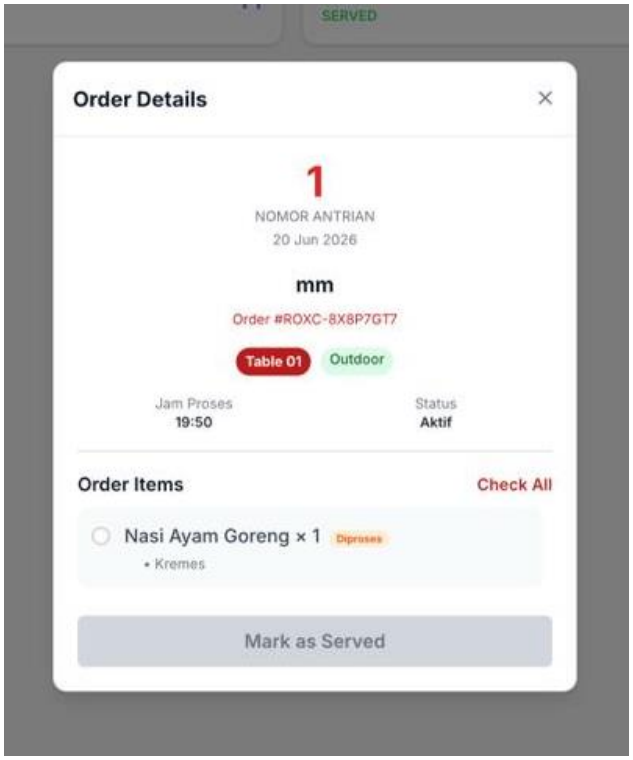
Gambar 3. 10 Tampilan Mobile *Kitchen Dashboard*.

Untuk menampilkan informasi pesanan secara lebih detail, sistem menyediakan modal *Order Details* yang dapat diakses dengan mengklik tombol *View Details* pada setiap kartu pesanan. Gambar 3.11 menampilkan modal detail pesanan pada panel antrian (*Queue Orders*). Modal ini menampilkan informasi lengkap pesanan seperti kode pesanan, nama pelanggan, nomor meja, waktu pemesanan, total item, serta daftar menu yang dipesan beserta catatan khusus dari pelanggan. Pada tampilan ini, tombol *Start Cooking* hanya aktif apabila pesanan belum diambil oleh *staff* dapur dan tidak sedang dalam proses kasir.



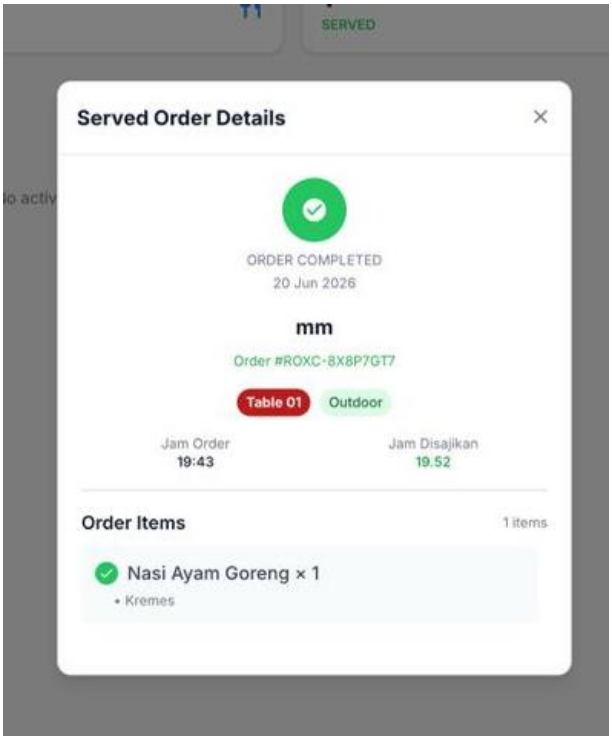
Gambar 3. 11 Tampilan Modal Detail Pesanan pada Antrian.

Setelah pesanan diambil dari antrian, pesanan akan berpindah ke panel *Active Orders*. Gambar 3.12 menunjukkan modal detail pesanan aktif yang menampilkan informasi serupa dengan modal antrian, namun dilengkapi dengan fitur *checkbox* pada setiap item pesanan. Fitur ini digunakan oleh *staff* dapur untuk menandai item yang telah selesai disiapkan. Tombol *Mark as Served* hanya dapat diakses setelah seluruh *checkbox* pada item pesanan dicentang, sehingga memastikan tidak ada item yang terlewat sebelum pesanan dinyatakan selesai.



Gambar 3. 12 Tampilan Modal Detail Pesanan Aktif.

Setelah pesanan selesai diproses dan ditandai sebagai *served*, pesanan akan berpindah ke panel *Served Orders*. Gambar 3.13 menampilkan modal detail pesanan selesai yang berisi informasi lengkap pesanan beserta waktu penyelesaian (*served time*). Modal ini digunakan untuk melihat riwayat pesanan yang telah selesai disajikan kepada pelanggan dan dapat menjadi bahan evaluasi kinerja operasional dapur.



Gambar 3. 13 Tampilan Modal Detail Pesanan Selesai.

Dengan adanya tampilan dashboard yang terstruktur dan modal detail yang informatif, staff dapur dapat mengelola pesanan dengan lebih efisien. Informasi pesanan dapat diakses dengan cepat, proses pengerjaan dapat dipantau secara *real-time* dan risiko kesalahan dalam penyajian pesanan dapat diminimalkan.

3.3.2 Implementasi Backend Menggunakan Laravel

Implementasi *backend* dilakukan untuk menyediakan layanan yang mendukung seluruh proses bisnis pada *Kitchen Dashboard*. *Backend* dikembangkan menggunakan framework Laravel karena menyediakan struktur aplikasi yang terorganisasi, fitur keamanan yang lengkap, serta kemudahan dalam pengelolaan database melalui *Eloquent ORM*. Pada *Kitchen Dashboard*, *backend* berperan sebagai penghubung antara antarmuka pengguna dengan basis data, sehingga seluruh proses pengambilan data pesanan dan perubahan status pesanan dapat dilakukan secara terpusat.

A. Analisis Kebutuhan Backend

Sebelum proses implementasi dilakukan, terlebih dahulu dilakukan analisis kebutuhan *backend* untuk menentukan fungsi-fungsi yang harus disediakan agar sistem dapat berjalan sesuai kebutuhan operasional dapur.

1) Pengelolaan Status Pesanan

Salah satu kebutuhan utama sistem adalah kemampuan untuk mengelola perubahan status pesanan secara konsisten. Setiap pesanan yang masuk harus mengikuti alur status yang telah ditentukan, yaitu dari status *paid* menjadi *processed* ketika diambil oleh *staff* dapur, kemudian berubah menjadi *served* setelah pesanan selesai disiapkan. Sistem juga harus mendukung proses pembatalan pesanan aktif sehingga status pesanan dapat dikembalikan ke kondisi sebelumnya apabila diperlukan. *Backend* harus memastikan setiap perubahan status tersimpan dengan benar pada basis data dan dapat langsung tercermin pada tampilan dashboard. Konsistensi data menjadi faktor penting karena perubahan status yang tidak sinkron dapat menyebabkan kesalahan informasi pada operasional dapur.

2) Validasi Akses

Selain mengelola data pesanan, *backend* juga harus memastikan keamanan sistem melalui mekanisme validasi akses. Tidak seluruh pengguna memiliki hak untuk mengakses *Kitchen Dashboard* maupun melakukan perubahan status pesanan. Oleh karena itu, *backend* perlu menerapkan autentikasi dan otorisasi untuk memastikan hanya pengguna yang memiliki hak akses sebagai *staff* dapur yang dapat menggunakan fitur-fitur yang tersedia.

B. Implementasi Controller Kitchen Dashboard

Berdasarkan kebutuhan yang telah dianalisis, dikembangkan sebuah *controller* khusus yang bertanggung jawab menangani seluruh proses yang berkaitan dengan *Kitchen Dashboard*. *Controller* ini berfungsi sebagai pusat logika bisnis yang menerima permintaan dari *frontend*, memproses data yang diperlukan, kemudian mengembalikan hasil dalam format *JSON*. Struktur dasar *controller Kitchen Dashboard* dapat dilihat pada Gambar 3.14.

```
class KitchenOrderController extends Controller
{
    public function queue()
    public function active()
    public function pickup($orderId)
    public function cancel($orderId)
    public function serve($orderId)
}
```

Gambar 3. 14 Potongan Kode *Controller Kitchen Dashboard*.

Controller Kitchen Dashboard terdiri dari beberapa fungsi utama yang memiliki tanggung jawab berbeda sesuai kebutuhan sistem, yaitu:

- Fungsi `*queue*()` digunakan untuk mengambil seluruh pesanan yang masih berada dalam status *paid*. Data yang diperoleh kemudian dikirimkan ke *frontend* untuk ditampilkan pada panel antrian pesanan.
- Fungsi `*active*()` digunakan untuk mengambil seluruh pesanan yang sedang diproses oleh dapur. Data tersebut ditampilkan pada panel *Active Orders* sehingga staf dapur dapat memantau pesanan yang sedang dikerjakan.
- Fungsi `*pickup*()` digunakan ketika *staff* dapur mengambil pesanan dari antrian. Fungsi ini mengubah status pesanan dari *paid* menjadi *processed* dan mencatat waktu pengambilan pesanan apabila diperlukan.
- Fungsi `*cancel*()` digunakan untuk membatalkan pesanan yang sedang diproses. Ketika fungsi ini dijalankan, status pesanan akan dikembalikan ke *paid* sehingga pesanan dapat kembali muncul pada daftar antrian.
- Fungsi `*serve*()` digunakan untuk menandai pesanan sebagai selesai. Fungsi ini mengubah status pesanan menjadi *served* dan mencatat waktu penyelesaian pesanan sebagai bagian dari riwayat transaksi.

Pemisahan fungsi-fungsi tersebut bertujuan untuk menjaga struktur kode tetap modular dan memudahkan proses pemeliharaan sistem apabila terdapat perubahan kebutuhan di masa mendatang.

C. *Middleware* dan Hak Akses

Untuk menjaga keamanan sistem, Laravel menyediakan *middleware* yang digunakan sebagai lapisan pemeriksaan sebelum pengguna dapat mengakses endpoint tertentu.

- ***Authentication***

  Mekanisme authentication digunakan untuk memastikan bahwa pengguna telah melakukan login sebelum mengakses *Kitchen Dashboard*. Apabila pengguna belum terautentikasi, sistem akan mengarahkan pengguna ke halaman login sehingga *endpoint* tidak dapat diakses secara langsung.

- ***Authorization***

  Selain autentikasi, sistem juga menerapkan *authorization* untuk membatasi hak akses berdasarkan peran pengguna. Pada *Kitchen Dashboard*, hanya pengguna dengan role *kitchen* yang diperbolehkan melihat data pesanan dan melakukan perubahan status pesanan. Penerapan authorization ini penting untuk mencegah pengguna lain seperti *customer* atau role lainnya melakukan tindakan yang dapat memengaruhi operasional dapur. Dengan demikian, keamanan dan integritas data dapat terjaga dengan baik.

D. Hasil Implementasi *Backend*

Setelah seluruh endpoint berhasil dikembangkan, dilakukan pengujian dengan menjalankan fitur-fitur yang terhubung dengan *backend* dan memverifikasi perubahan data pada basis data menggunakan aplikasi TablePlus. Pengujian dilakukan pada proses pengambilan data pesanan, penyimpanan detail item pesanan, serta perubahan status pesanan yang dikirim dari *frontend*.

Hasil pengujian menunjukkan bahwa backend mampu memproses setiap permintaan dengan baik dan melakukan operasi *Create*, *Read*, *Update*, dan *Delete* (CRUD) pada basis data. Sebagai salah satu bentuk validasi integrasi sistem, dilakukan pengecekan langsung pada tabel *order_items* untuk memastikan data pesanan yang dikirim melalui REST API oleh frontend berhasil disimpan secara konsisten. Tabel tersebut mencatat setiap item makanan atau minuman yang dipesan beserta detail kuantitas, harga, dan kode pesanan yang terasosiasi. Hasil verifikasi penyimpanan data pada basis data dapat dilihat pada Gambar 3.15.

id BIGINT	booking_order_id BIGINT	partner_product_id BIGINT	product_code VARCHAR	product_name VARCHAR	quantity INT	base_price DECIMAL	cogs DECIMAL	promo_amount DECIMAL	options_price DECIMAL	promo_id BIGINT	promo_type VARCHAR	customer_note TEXT	kitchen_process_id BIGINT
200656	200594	78	PPD-6-6923DC3211703	Nasi Ayam Goreng	21	20000.00	6000.00	4000.00	0.00	13	percentage	NULL	NULL
200657	200594	79	PPD-6-6923DC3212377	Nasi Goreng	17	15000.00	6500.00	0.00	0.00	NULL	NULL	NULL	NULL
200658	200594	108	PPD-6-695F600F18043	Bakso	13	10000.00	6300.00	0.00	0.00	NULL	NULL	NULL	NULL
200659	200594	82	PPD-6-693F79D7362A3	Es Teh	26	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200660	200594	112	PPD-6-69A93274B739A	Es Jeruk	17	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200761	200695	108	PPD-6-695F600F18043	Bakso	2	10000.00	5000.00	0.00	0.00	NULL	NULL	NULL	NULL
200762	200696	78	PPD-6-6923DC3211703	Nasi Ayam Goreng	13	20000.00	6000.00	4000.00	1000.00	13	percentage	NULL	NULL
200763	200696	79	PPD-6-6923DC3212377	Nasi Goreng	19	15000.00	6500.00	0.00	0.00	NULL	NULL	NULL	NULL
200764	200696	108	PPD-6-695F600F18043	Bakso	10	10000.00	6300.00	0.00	0.00	NULL	NULL	NULL	NULL
200765	200696	82	PPD-6-693F79D7362A3	Es Teh	28	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200766	200696	112	PPD-6-69A93274B739A	Es Jeruk	15	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200767	200697	78	PPD-6-6923DC3211703	Nasi Ayam Goreng	10	20000.00	6000.00	4000.00	0.00	13	percentage	NULL	NULL
200768	200697	79	PPD-6-6923DC3212377	Nasi Goreng	17	15000.00	6500.00	0.00	0.00	NULL	NULL	NULL	NULL
200769	200697	108	PPD-6-695F600F18043	Bakso	12	10000.00	6300.00	0.00	0.00	NULL	NULL	NULL	NULL
200770	200697	82	PPD-6-693F79D7362A3	Es Teh	14	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200771	200697	112	PPD-6-69A93274B739A	Es Jeruk	22	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200772	200698	79	PPD-6-6923DC3212377	Nasi Goreng	17	15000.00	6500.00	0.00	0.00	NULL	NULL	NULL	NULL
200773	200698	78	PPD-6-6923DC3211703	Nasi Ayam Goreng	9	20000.00	6000.00	4000.00	0.00	13	percentag	NULL	NULL
200774	200698	108	PPD-6-695F600F18043	Bakso	5	10000.00	6300.00	0.00	0.00	NULL	NULL	NULL	NULL
200775	200698	82	PPD-6-693F79D7362A3	Es Teh	19	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200776	200698	112	PPD-6-69A93274B739A	Es Jeruk	8	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200777	200699	79	PPD-6-6923DC3212377	Nasi Goreng	1	15000.00	6500.00	0.00	0.00	NULL	NULL	NULL	NULL
200778	200699	112	PPD-6-69A93274B739A	Es Jeruk	1	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200779	200699	82	PPD-6-693F79D7362A3	Es Teh	1	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
200780	200699	108	PPD-6-695F600F18043	Bakso	1	10000.00	6300.00	0.00	0.00	NULL	NULL	NULL	NULL
201781	201700	79	PPD-6-6923DC3212377	Nasi Goreng	4	15000.00	6500.00	0.00	0.00	NULL	NULL	NULL	NULL
201782	201700	112	PPD-6-69A93274B739A	Es Jeruk	2	3000.00	1000.00	0.00	0.00	NULL	NULL	NULL	NULL
201783	201701	78	PPD-6-6923DC3211703	Nasi Ayam Goreng	1	20000.00	6000.00	4000.00	1000.00	13	percentage	NULL	0
201784	201702	78	PPD-6-6923DC3211703	Nasi Ayam Goreng	1	20000.00	6000.00	0.00	1000.00	NULL	NULL	NULL	0
201785	201703	78	PPD-6-6923DC3211703	Nasi Ayam Goreng	1	20000.00	6000.00	0.00	1000.00	NULL	NULL	NULL	NULL

Gambar 3. 15 Hasil verifikasi penyimpanan data pesanan pada basis data menggunakan TablePlus (Data sampel sistem).

Berdasarkan Gambar 3.9, seluruh data yang dikirimkan oleh frontend melalui mekanisme *polling* dan interaksi pengguna (seperti *pickup*, *cancel*, dan *serve*) berhasil tercatat dengan benar pada basis data. Hal ini membuktikan bahwa integrasi antara antarmuka *Kitchen Dashboard* dan *backend* Laravel melalui REST API telah berjalan sesuai dengan rancangan yang diharapkan. Data yang tersimpan dan diperbarui pada basis data dapat diverifikasi secara langsung melalui TablePlus sehingga memastikan tidak terjadi kehilangan data atau ketidaksinkronan status antara tampilan pengguna dan sistem pusat.

3.3.3 Pengembangan REST API

REST API dikembangkan sebagai mekanisme komunikasi antara *frontend* dan *backend*. Dengan menggunakan REST API, proses pertukaran data dapat dilakukan secara terstruktur menggunakan format JSON yang ringan dan mudah diproses oleh aplikasi. Pendekatan ini memungkinkan *frontend* dan *backend* dikembangkan secara terpisah tanpa mengurangi kemampuan integrasi antar komponen sistem.

A. Implementasi KitchenApiService

Pada sisi *frontend*, seluruh komunikasi dengan backend dipusatkan dalam kelas *KitchenApiService*. Pendekatan ini digunakan agar proses pengiriman permintaan HTTP dapat dilakukan secara konsisten melalui satu titik akses. Metode utama yang digunakan adalah "*_fetch*()" sebagaimana ditunjukkan pada Gambar 3.16.

```
async _fetch(endpoint, method = 'GET') {
    const headers = {
        'Content-Type': 'application/json',
        'Accept': 'application/json',
        'X-Requested-With': 'XMLHttpRequest'
    };
    if (method !== 'GET' && this.csrfToken) {
        headers['X-CSRF-TOKEN'] = this.csrfToken;
    }
    const response = await fetch(endpoint, { method, headers,
        credentials: 'same-origin' });
    if (!response.ok) throw new Error(`HTTP error! status:
${response.status}`);
    return await response.json();
}
```

Gambar 3. 16 Potongan Kode *Method _fetch*().

Metode `*_fetch*()` berfungsi sebagai pembungkus seluruh permintaan HTTP yang dikirimkan ke server. Sebelum permintaan dikirim, metode ini secara otomatis menambahkan *header* yang diperlukan seperti *Content-Type*, *Accept*, dan *X-Requested-With*. Untuk permintaan yang melakukan perubahan data, metode ini juga menambahkan CSRF Token sebagai mekanisme perlindungan terhadap serangan *Cross Site Request Forgery* (CSRF). Setelah permintaan dikirim, metode akan memeriksa status respons yang diterima. Jika terjadi kesalahan, sistem akan menghasilkan *exception* sehingga kesalahan dapat ditangani oleh bagian lain dari aplikasi. Dengan pendekatan ini, seluruh proses komunikasi dengan *backend* menjadi lebih terstandarisasi dan mudah dikelola.

B. Implementasi *Caching API*

Untuk meningkatkan efisiensi sistem, diterapkan mekanisme *caching* pada beberapa endpoint yang sering diakses namun tidak mengalami perubahan data secara terus-menerus.

Implementasi *caching* dilakukan melalui metode `*_fetchWithCache*()` seperti yang ditunjukkan pada Gambar 3.17.

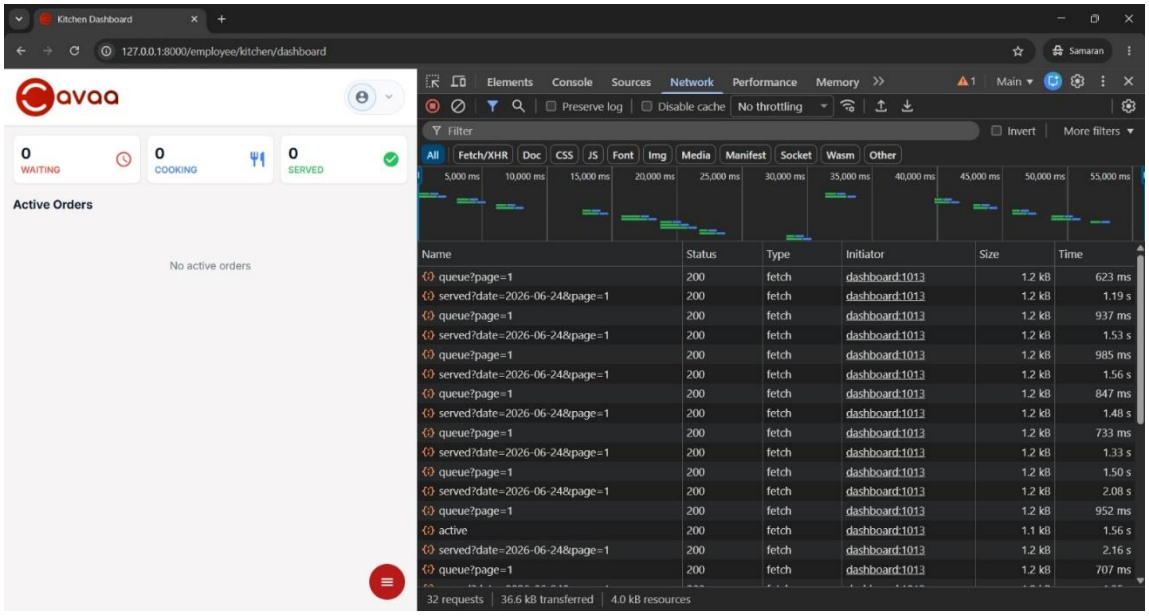
```
async _fetchWithCache(cacheKey, endpoint) {
    const cached = this.cache.get(cacheKey);
    if (cached && Date.now() - cached.timestamp < this.cacheTimeout) {
        return cached.data;
    }
    const data = await this._fetch(endpoint);
    this.cache.set(cacheKey, { data, timestamp: Date.now() });
    return data;
}
```

Gambar 3. 17 Potongan Kode *_fetchWithCache*().

Metode ini terlebih dahulu memeriksa apakah data yang diminta telah tersedia pada *cache*. Apabila data masih berada dalam batas waktu *cache* yang ditentukan, sistem akan langsung mengembalikan data tersebut tanpa melakukan permintaan baru ke server. Sebaliknya, apabila data tidak tersedia atau sudah melewati batas waktu *cache*, sistem akan mengirimkan permintaan baru ke server dan menyimpan hasilnya ke dalam *cache* untuk digunakan pada permintaan berikutnya. Mekanisme ini mampu mengurangi jumlah permintaan HTTP yang tidak diperlukan, mempercepat waktu respon aplikasi, serta mengurangi beban kerja server.

C. Hasil Pengujian API

Setelah implementasi REST API selesai dilakukan, tahap berikutnya adalah pengujian respons yang dihasilkan oleh setiap *endpoint*. Pengujian dilakukan untuk memastikan struktur JSON yang dikembalikan sesuai dengan kebutuhan *frontend*. Hasil pengujian menunjukkan bahwa seluruh *endpoint* berhasil mengembalikan data yang dibutuhkan, baik untuk daftar antrian pesanan, pesanan aktif, maupun riwayat pesanan yang telah selesai. Hasil respon JSON dari setiap *endpoint* dapat dilihat pada Gambar 3.18.



Gambar 3. 18 Hasil Respon JSON dari Setiap *Endpoint*.

3.3.4 Pengembangan Struktur Frontend Modular

Untuk meningkatkan keteraturan kode dan mempermudah proses pemeliharaan aplikasi, *frontend Kitchen Dashboard* dikembangkan menggunakan pendekatan modular. Pendekatan ini membagi tanggung jawab aplikasi ke dalam beberapa kelas yang memiliki fungsi spesifik sehingga setiap bagian dapat dikembangkan dan diuji secara terpisah.

A. Implementasi Kelas *KitchenApiService*

KitchenApiService merupakan kelas yang bertanggung jawab mengelola seluruh komunikasi antara *frontend* dan *backend*. Seluruh proses pengambilan data pesanan, pembaruan status pesanan, serta pengelolaan *cache* dilakukan melalui kelas ini. Dengan memusatkan seluruh komunikasi API pada satu kelas, perubahan *endpoint* atau mekanisme komunikasi dapat dilakukan tanpa memengaruhi bagian *frontend* lainnya. Pendekatan ini meningkatkan *maintainability* dan mengurangi duplikasi kode.

B. Implementasi Kelas *KitchenUIRenderer*

KitchenUIRenderer dikembangkan untuk menangani proses pembentukan elemen antarmuka secara dinamis. Kelas ini berisi kumpulan metode yang bertugas menghasilkan struktur HTML untuk berbagai jenis kartu pesanan yang ditampilkan pada *dashboard*. Implementasi metode *createQueueItem* dan *createActiveOrderCard* dapat dilihat pada Gambar 3.19 dan Gambar 3.20.

```
static createQueueItem(order) {
```

```
    const isTakenByCashier = order.cashier_process_id;
    return `<div class="compact-queue-card
        ${isTakenByCashier ? 'opacity-60' : ''}" data-order-
id="${order.id}">
        <div class="text-red-600 font-
black">${order.booking_order_code}</div>
        <div class="text-xs">${order.customer_name}</div>
    </div>`;
}
```

Gambar 3. 19 Potongan Kode *createQueueItem*().

```
static createActiveOrderCard(order) {
    return `<div class="kitchen-active-order" id="kitchen-active-order-
${order.id}">
        <div class="text-red-600 font-black text-
xl">#${order.active_queue_number}</div>
        <h3>${order.customer_name}</h3>
        <button class="kitchen-details-btn" data-order-
id="${order.id}">
            View Details
        </button>
    </div>`;
}
```

Gambar 3. 20 Potongan Kode *createQueueItem*()

Metode `*createQueueItem*()` digunakan untuk membangun tampilan kartu pesanan pada panel antrian. Selain menampilkan informasi dasar pesanan, metode ini juga menangani kondisi ketika pesanan sedang diproses oleh kasir dengan memberikan efek *opacity* yang lebih rendah sebagai indikator visual. Sementara itu, metode `*createActiveOrderCard*()` digunakan untuk membangun tampilan kartu pada panel pesanan aktif. Kartu ini menampilkan nomor urut antrian aktif, nama pelanggan, serta tombol yang digunakan untuk membuka detail pesanan.

C. Implementasi Kelas *KitchenDashboard*

KitchenDashboard merupakan kelas utama yang berfungsi sebagai pengendali seluruh aktivitas pada halaman *dashboard*. Kelas ini mengelola *state* lokal aplikasi, mengatur proses *polling* data secara berkala, menangani interaksi pengguna, mengelola modal, serta memperbarui tampilan ketika terjadi perubahan data. Kelas ini juga bertindak sebagai penghubung antara *KitchenApiService* dan *KitchenUIRenderer*. Data yang diperoleh dari server diproses terlebih dahulu oleh *KitchenDashboard* sebelum diteruskan ke *KitchenUIRenderer* untuk ditampilkan kepada pengguna.

D. Hasil Implementasi Struktur Modular

Penerapan struktur modular menghasilkan kode yang lebih terorganisasi dan mudah dipelihara. Setiap kelas memiliki tanggung jawab yang jelas sehingga perubahan pada satu bagian tidak memengaruhi bagian lainnya secara langsung. Sebagai bentuk implementasi

modular, seluruh logika pembuatan elemen antarmuka (UI) dipisahkan ke dalam kelas *KitchenUIRenderer*. Dengan pendekatan ini, *method createQueueItem()* dan *createActiveOrderCard()* dapat digunakan kembali secara konsisten. Hasil dari struktur modular ini telah berhasil menghasilkan tampilan kartu pesanan dan modal detail pesanan yang dinamis, seperti yang telah ditunjukkan pada Gambar 3.8, Gambar 3.9, dan Gambar 3.10.

3.3.5 Implementasi Pengelolaan Status Pesanan

Pengelolaan status pesanan merupakan salah satu fungsi utama yang dikembangkan pada Kitchen Dashboard. Fitur ini memungkinkan *staff* dapur untuk memantau perkembangan setiap pesanan mulai dari saat pesanan diterima hingga pesanan selesai disajikan kepada pelanggan. Implementasi pengelolaan status pesanan dirancang untuk memberikan informasi yang akurat mengenai kondisi terkini setiap pesanan sehingga proses operasional dapur dapat berjalan lebih terorganisasi dan efisien. Melalui fitur ini, setiap perubahan status pesanan akan langsung diperbarui pada antarmuka pengguna tanpa memerlukan penyegaran halaman secara manual. Perubahan status yang dilakukan oleh pengguna juga akan disimpan ke dalam basis data sehingga seluruh informasi pesanan tetap konsisten antara *frontend* dan *backend*.

A. Analisis Kebutuhan Status Pesanan

Sebelum implementasi dilakukan, terlebih dahulu dilakukan analisis terhadap alur status pesanan yang digunakan dalam proses operasional restoran. Berdasarkan hasil analisis, setiap pesanan harus melalui beberapa tahapan status yang menggambarkan progres pengerjaan pesanan oleh *staff* dapur.

1) *Paid*

Status *paid* merupakan status awal yang diberikan kepada pesanan ketika pesanan baru diterima dari kasir dan belum diambil oleh *staff* dapur. Seluruh pesanan dengan status *paid* ditampilkan pada panel *Queue Orders* yang berfungsi sebagai daftar antrian pekerjaan yang harus diproses. Keberadaan status *paid* membantu staf dapur mengetahui jumlah pesanan yang belum dikerjakan dan menentukan prioritas pengerjaan berdasarkan urutan pesanan yang masuk.

2) Processed

Status *processed* menunjukkan bahwa pesanan telah diambil oleh *staff* dapur dan sedang dalam proses pengerjaan. Ketika status ini diterapkan, pesanan akan berpindah dari panel *Queue Orders* ke panel *Active Orders*. Status *processed* digunakan untuk memberikan informasi bahwa pesanan sedang ditangani sehingga tidak dapat

diambil kembali oleh *staff* dapur lain. Dengan demikian, risiko terjadinya duplikasi pekerjaan dapat diminimalkan.

3) Served

Status *served* merupakan status akhir yang menunjukkan bahwa seluruh item dalam pesanan telah selesai disiapkan dan siap disajikan kepada pelanggan. Pesanan yang telah mencapai status ini akan dipindahkan ke panel *Served Orders* sebagai bagian dari riwayat pesanan yang telah selesai diproses. Penerapan status *served* membantu *staff* dapur dan manajemen restoran dalam melakukan pemantauan terhadap jumlah pesanan yang telah berhasil diselesaikan selama periode operasional tertentu. Berdasarkan ketiga status tersebut, alur perubahan status pesanan yang diterapkan pada *Kitchen Dashboard* dapat digambarkan sebagai berikut:

Paid → Processed → Served

Selain alur utama tersebut, sistem juga menyediakan mekanisme pembatalan pesanan aktif sehingga pesanan dengan status *processed* dapat dikembalikan kembali ke status *paid* apabila terjadi kesalahan pengambilan pesanan atau perubahan kondisi operasional.

Fitur *Pickup Order* dikembangkan untuk menangani proses pengambilan pesanan dari daftar antrian oleh *staff* dapur. Ketika pengguna menekan tombol *pickup* pada kartu pesanan, sistem akan mengirimkan permintaan ke server untuk mengubah status pesanan dari *paid* menjadi *processed*. Implementasi fungsi *pickUpOrder* dapat dilihat pada Gambar 3.21.

```
async pickUpOrder(orderId) {
    const result = await this.api.pickUpOrder(orderId);
    if (result.success) {
        this.queueOrders = this.queueOrders.filter(o => o.id !=
orderId);
        this.activeOrders.push({ ...order, order_status: 'PROCESSED'
});
        this.addSingleActiveOrderCard(newActiveOrder);
    }
}
```

Gambar 3. 21 Potongan Kode *pickUpOrder.*

Apabila server mengembalikan respons berhasil (`*result.success*` bernilai true), sistem akan melakukan pembaruan data pada sisi *frontend*. Data pesanan yang sebelumnya berada pada *array* `*queueOrders*` akan dihapus menggunakan metode `*filter*()`, sehingga pesanan tidak lagi ditampilkan pada panel antrian.

Selanjutnya, objek pesanan yang sama ditambahkan ke dalam *array* `*activeOrders*` dengan status baru yaitu `*PROCESSED*`. Setelah data berhasil diperbarui, sistem memanggil

fungsi `addSingleActiveOrderCard()` untuk menampilkan kartu pesanan pada panel *Active Orders*. Pendekatan ini memungkinkan perubahan tampilan terjadi secara langsung tanpa perlu menunggu proses polling berikutnya. Dengan demikian, pengguna memperoleh umpan balik secara *real-time* setelah melakukan tindakan pengambilan pesanan.

B. Implementasi Fitur *Cancel Order*

Fitur *Cancel Order* dikembangkan untuk mengakomodasi kondisi ketika pesanan yang sedang diproses perlu dikembalikan ke daftar antrian. Kondisi ini dapat terjadi apabila pesanan diambil secara tidak sengaja atau terdapat perubahan keputusan dalam proses operasional dapur. Implementasi fungsi *cancelOrder* dapat dilihat pada Gambar 3.22.

```
async cancelOrder(orderId) {
    const result = await this.api.cancelOrder(orderId);
    if (result.success) {
        this.activeOrders = this.activeOrders.filter(o => o.id !=
orderId);
        this.queueOrders.unshift({ ...order, order_status: 'PAID' });
        this.renumberActiveOrdersAfter(order.active_queue_number);
    }
}
```

Gambar 3. 22 Potongan Kode *cancelOrder.*

Fungsi `*cancelOrder*()` mengirimkan permintaan ke *backend* untuk mengubah status pesanan dari *processed* menjadi *paid*. Setelah server mengembalikan respons berhasil, sistem akan memperbarui data pada sisi *frontend*. Pesanan yang dibatalkan akan dihapus dari array `*activeOrders*` menggunakan metode `*filter*()`. Selanjutnya, objek pesanan yang sama ditambahkan kembali ke awal array `*queueOrders*` menggunakan metode `*unshift*()`. Penggunaan `*unshift*()` bertujuan agar pesanan yang dikembalikan segera terlihat pada bagian atas daftar antrian.

Selain memindahkan pesanan kembali ke antrian, fungsi ini juga menjalankan metode `*renumberActiveOrdersAfter*()`. Metode tersebut digunakan untuk memperbarui nomor urut pesanan aktif lainnya yang terdampak akibat penghapusan pesanan dari daftar *Active Orders*. Pembaruan nomor urut ini penting untuk menjaga konsistensi informasi yang ditampilkan kepada pengguna sehingga urutan pengerjaan pesanan tetap sesuai dengan kondisi aktual pada sistem.

C. Implementasi Fitur Mark As Served

Fitur *Mark As Served* digunakan untuk menandai bahwa seluruh item dalam suatu pesanan telah selesai disiapkan dan siap disajikan kepada pelanggan. Fitur ini merupakan tahap akhir

dalam siklus pengelolaan status pesanan pada *Kitchen Dashboard*. Implementasi fungsi *markAsServed* dapat dilihat pada Gambar 3.23.

```
async markAsServed(orderId) {
    const result = await this.api.markAsServed(orderId);
    if (result.success) {
        this.activeOrders = this.activeOrders.filter(o => o.id !=
orderId);
        this.servedOrders.unshift({ ...order, order_status: 'SERVED'
});
        this.addSingleServedOrderCard(servedOrder);
    }
}
```

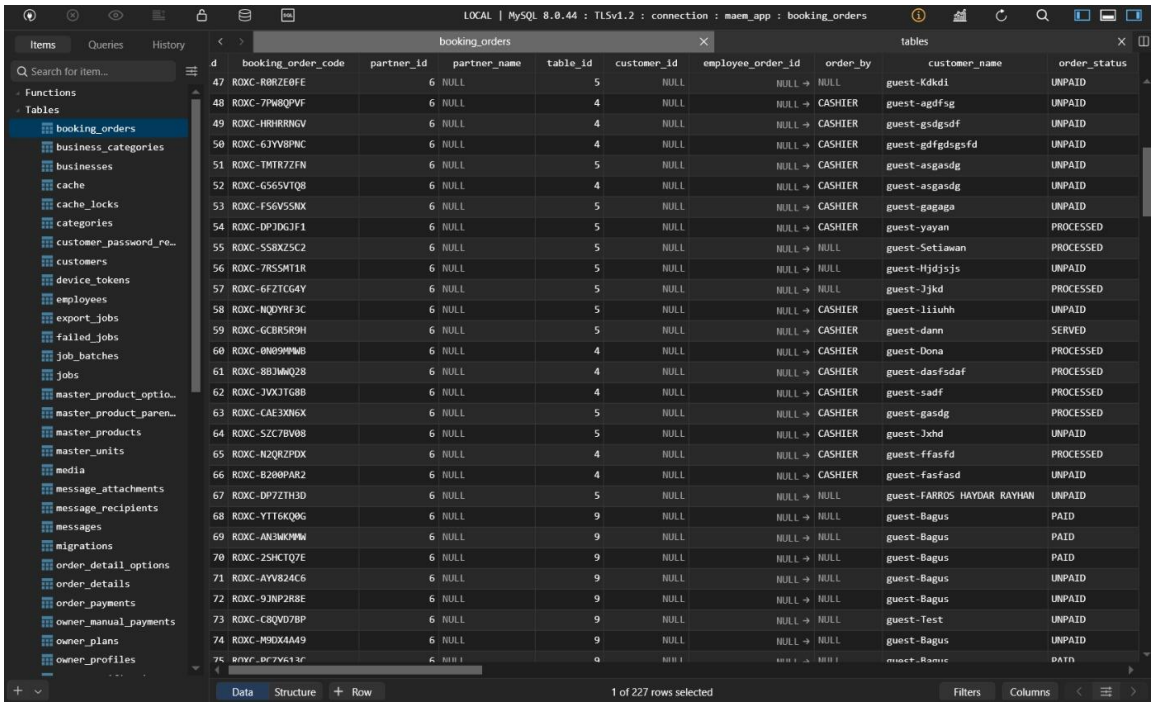
Gambar 3. 23 Potongan Kode *markAsServed.*

Fungsi `*markAsServed*()` diawali dengan mengirimkan permintaan ke *backend* untuk mengubah status pesanan menjadi *served*. Setelah proses pembaruan pada basis data berhasil dilakukan, *backend* akan mengirimkan respons keberhasilan kepada frontend. Pada sisi *frontend*, pesanan yang telah selesai diproses akan dihapus dari array `*activeOrders*`. Selanjutnya, objek pesanan ditambahkan ke dalam array `*servedOrders*` menggunakan metode `*unshift*()` sehingga pesanan terbaru akan tampil pada bagian atas daftar riwayat pesanan selesai. Setelah data berhasil diperbarui, sistem memanggil fungsi `*addSingleServedOrderCard*()` untuk menampilkan kartu pesanan pada panel *Served Orders*. Dengan cara ini, pengguna dapat langsung melihat bahwa pesanan telah berhasil dipindahkan ke kategori pesanan selesai. Untuk menjaga validitas proses operasional, fungsi ini hanya dapat dijalankan setelah seluruh item pesanan pada modal detail telah ditandai selesai melalui *checkbox* yang tersedia. Mekanisme tersebut bertujuan untuk memastikan bahwa tidak ada item pesanan yang terlewat sebelum pesanan dinyatakan selesai dan disajikan kepada pelanggan.

D. Hasil Implementasi Status Pesanan

Setelah seluruh fitur pengelolaan status pesanan berhasil dikembangkan, dilakukan pengujian untuk memastikan setiap transisi status berjalan sesuai dengan alur bisnis yang telah dirancang. Pengujian dilakukan dengan mensimulasikan perubahan status mulai dari pesanan baru masuk hingga pesanan selesai disajikan kepada pelanggan. Hasil pengujian menunjukkan bahwa setiap perubahan status dapat dilakukan dengan baik dan langsung tercermin pada tampilan *Kitchen Dashboard* tanpa memerlukan penyegaran halaman secara manual. Pesanan yang diambil berhasil berpindah dari panel *Queue Orders* ke *Active Orders*, pesanan yang dibatalkan dapat kembali ke daftar antrian, dan pesanan yang selesai diproses berhasil dipindahkan ke panel *Served Orders*.

Untuk memastikan konsistensi data antara tampilan pengguna dan sistem pusat, dilakukan verifikasi langsung pada basis data menggunakan TablePlus. Hasil verifikasi menunjukkan bahwa setiap perubahan status yang dilakukan melalui antarmuka pengguna tercatat dengan benar pada kolom *order_status* di tabel *booking_orders*. Nilai status yang terekam di basis data meliputi *UNPAID* (pesanan baru menunggu pembayaran), *PAID* (pesanan telah dibayar dan siap diproses), *PROCESSED* (pesanan sedang dimasak oleh dapur), *SERVED* (pesanan telah selesai disajikan), serta *CANCEL* (pesanan dibatalkan dan dikembalikan ke antrian). Konsistensi antara status pada tampilan dashboard dan basis data menunjukkan bahwa integrasi antara *frontend* dan *backend* telah berjalan dengan baik. Visualisasi data status pesanan yang tersimpan pada basisdata dapat dilihat pada Gambar 3.24.



Gambar 3. 24 Visualisasi Data Status Pesanan yang Tersimpan Menggunakan TablePlus.

3.4 Optimalisasi Mekanisme Pembaruan Data

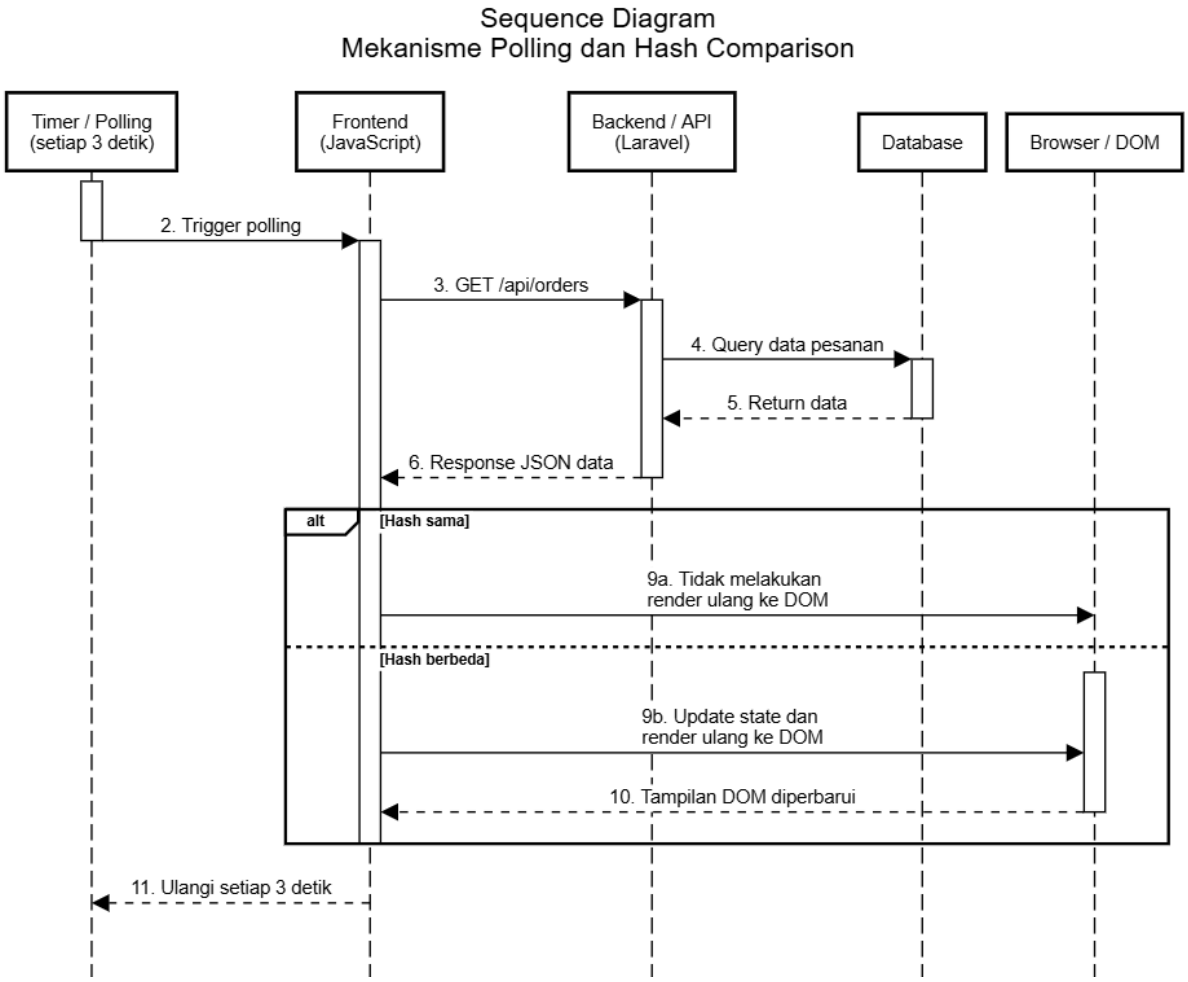
Salah satu kebutuhan utama pada *Kitchen Dashboard* adalah kemampuan untuk menampilkan informasi pesanan yang selalu diperbarui sesuai kondisi operasional restoran. Pada lingkungan kerja dapur, pesanan baru dapat masuk kapan saja dan status pesanan dapat berubah dalam waktu singkat. Oleh karena itu, sistem harus mampu memperbarui data secara cepat tanpa mengharuskan pengguna melakukan penyegaran halaman secara manual. Meskipun pembaruan data secara berkala dapat meningkatkan akurasi informasi yang ditampilkan, mekanisme tersebut juga berpotensi meningkatkan penggunaan sumber daya

server dan browser apabila tidak dikelola dengan baik. Untuk mengatasi permasalahan tersebut, dilakukan beberapa optimasi yang meliputi implementasi *polling API*, *hash comparison*, *Page Visibility API*, dan *debounce* pada fitur pencarian. Kombinasi keempat mekanisme ini bertujuan untuk menjaga keseimbangan antara kecepatan pembaruan data dan efisiensi penggunaan sumber daya sistem.

3.4.1 Implementasi Mekanisme Polling API

Kitchen Dashboard membutuhkan pembaruan data pesanan secara berkala agar perubahan status pesanan dapat ditampilkan tanpa pengguna melakukan *refresh* halaman secara manual. Untuk memenuhi kebutuhan tersebut, sistem menerapkan mekanisme *polling* dengan interval tiga detik. Pada setiap interval, *frontend* mengirimkan *request* HTTP melalui REST API untuk memperoleh data pesanan terbaru dari *backend* Laravel. Backend kemudian mengambil data yang diperlukan dari basis data dan mengembalikan *respons* dalam format JSON kepada *frontend*.

Polling akan tetap menghasilkan *request* secara berkala meskipun data belum mengalami perubahan, mekanisme tersebut dikombinasikan dengan *hash comparison* pada sisi *frontend*. Setelah *respons* diterima, *frontend* menghitung dan membandingkan hash data terbaru dengan hash data sebelumnya. Apabila hash tidak berubah, sistem tidak melakukan *render* ulang terhadap data pesanan. Sebaliknya, apabila hash berubah, *frontend* memproses pembaruan tampilan sesuai dengan data terbaru. Alur komunikasi antara *frontend*, *backend*, basis data, serta proses pemeriksaan hash ditunjukkan pada Gambar 3.25.



Gambar 3. 25 *Sequence Diagram* Mekanisme *Polling* dan *Hash Comparison*.

A. Kebutuhan *Real-Time Update*

Kitchen Dashboard digunakan oleh *staff* dapur untuk memantau kondisi pesanan secara langsung selama proses operasional berlangsung. Pada praktiknya, pesanan baru dapat masuk sewaktu-waktu dari sistem kasir sehingga informasi yang ditampilkan pada *dashboard* harus selalu diperbarui. Apabila sistem hanya memperbarui data ketika halaman dimuat ulang secara manual, terdapat kemungkinan staf dapur terlambat mengetahui adanya pesanan baru. Kondisi tersebut dapat menyebabkan keterlambatan proses pelayanan dan menurunkan efisiensi operasional restoran. Berdasarkan kebutuhan tersebut, diperlukan mekanisme pembaruan data otomatis yang dapat berjalan secara berkala tanpa memerlukan interaksi pengguna. Mekanisme yang dipilih adalah *polling API* karena relatif sederhana untuk diimplementasikan dan mampu memberikan informasi yang cukup cepat untuk kebutuhan operasional *Kitchen Dashboard*.

B. Implementasi *Start Realtime Polling*

Polling diimplementasikan menggunakan fungsi `*setInterval*()` yang secara berkala mengirimkan permintaan ke server untuk memeriksa kemungkinan adanya perubahan data pesanan. Interval pembaruan ditetapkan setiap tiga detik untuk menjaga keseimbangan antara kecepatan pembaruan informasi dan beban yang diterima server. Implementasi fungsi `*startRealtimePolling*()` dapat dilihat pada Gambar 3.26.

```
startRealtimePolling() {
    if (this.pollingInterval) clearInterval(this.pollingInterval);
    this.pollingInterval = setInterval(async () => {
        if (this.isRefreshing) return;
        await this.checkForUpdates();
    }, this.pollingIntervalTime);
}
```

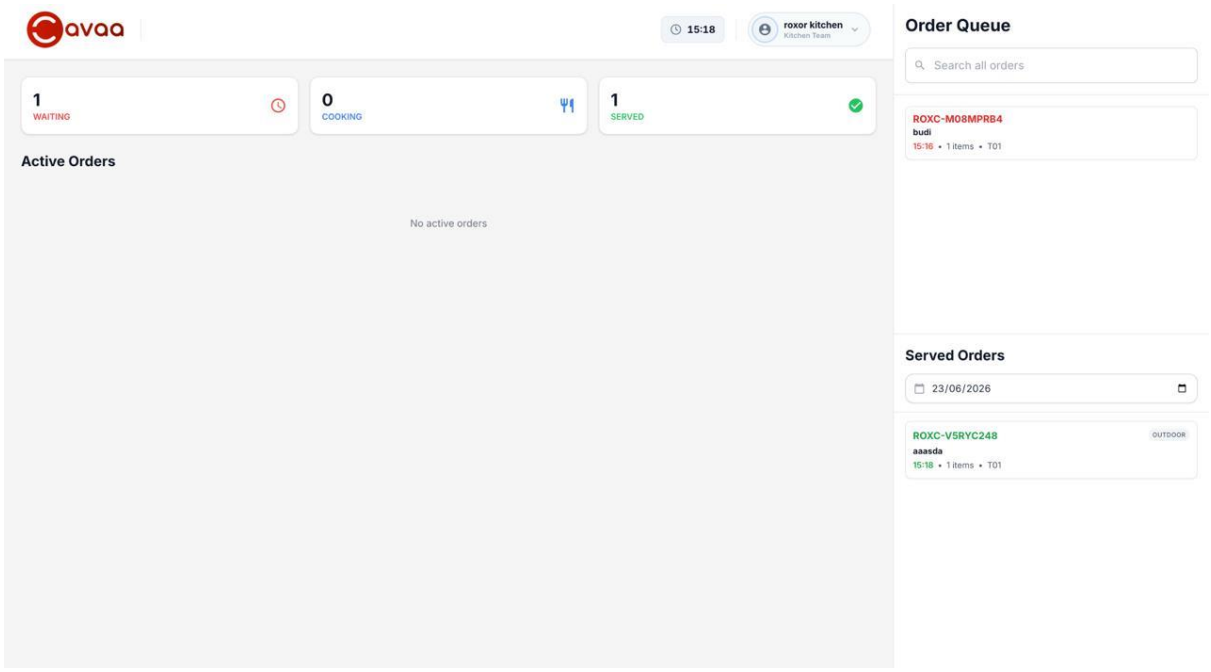
Gambar 3. 26 Potongan Kode *startRealtimePolling*.

Fungsi `*startRealtimePolling*()` bertanggung jawab untuk mengaktifkan mekanisme *polling* pada *Kitchen Dashboard*. Sebelum membuat interval baru, sistem terlebih dahulu memeriksa apakah terdapat *polling* yang masih berjalan melalui variabel `*pollingInterval*`. Apabila ditemukan *polling* aktif, interval tersebut akan dihentikan menggunakan `*clearInterval*()` untuk mencegah terjadinya duplikasi proses *polling*.

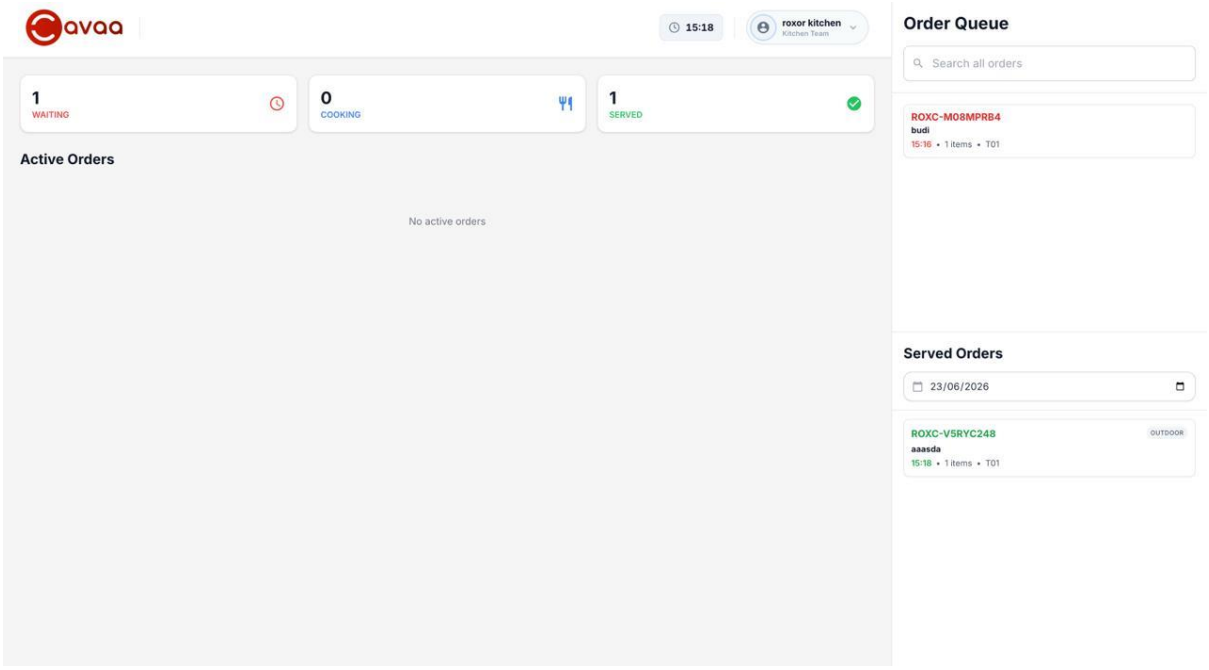
Setelah itu, sistem membuat interval baru menggunakan `*setInterval*()` yang akan menjalankan fungsi `*checkForUpdates*()` secara berkala sesuai nilai yang ditentukan pada variabel `*pollingIntervalTime*`. Terdapat juga pemeriksaan terhadap variabel `*isRefreshing*`. Pemeriksaan ini bertujuan untuk mencegah proses *polling* berjalan ketika sistem sedang melakukan pemuatan data secara penuh. Dengan demikian, kemungkinan terjadinya konflik antar proses pembaruan data dapat diminimalkan.

C. Hasil Implementasi *Polling*

Setelah mekanisme *polling* diterapkan, *Kitchen Dashboard* mampu memperbarui informasi pesanan secara otomatis tanpa memerlukan penyegaran halaman. Ketika terdapat pesanan baru atau perubahan status pesanan, informasi tersebut dapat ditampilkan secara langsung pada dashboard dalam waktu singkat. Hasil implementasi menunjukkan bahwa *staff* dapur dapat memperoleh informasi terbaru secara lebih cepat dibandingkan metode pembaruan manual. Tampilan pembaruan data otomatis dapat dilihat pada Gambar 3.26 dan Gambar 3.27.



Gambar 3. 27 Data Sebelum Masuk.



Gambar 3. 28 Data Langsung Masuk

3.4.2 Implementasi *Hash Comparison*

Hash comparison adalah teknik optimasi yang bekerja sepenuhnya di sisi klien (*browser*). Teknik ini tidak mengurangi jumlah request yang dikirim ke server, namun mengurangi beban pemrosesan di *browser* dengan cara membatasi *render* ulang hanya ketika terjadi perubahan data. Pada *Kitchen Dashboard*, *hash comparison* diimplementasikan pada fungsi

checkForUpdates(). Setiap kali data diterima dari server, sistem menghasilkan nilai hash menggunakan fungsi generateHash(). Nilai hash baru ini dibandingkan dengan nilai hash sebelumnya yang tersimpan di variabel lastQueueHash, lastActiveHash, dan lastServedHash. Jika kedua hash identik, sistem menyimpulkan bahwa tidak ada perubahan data sehingga proses *render* ulang tidak dilakukan.

Perlu dibedakan bahwa *hash comparison* tidak berfungsi untuk mengurangi jumlah permintaan (*request*) ke server. Mekanisme *polling* tetap mengirimkan request sesuai interval yang telah ditentukan, yaitu setiap tiga detik. *Hash comparison* bekerja setelah data diterima oleh browser dengan membandingkan *hash* data terbaru dengan *hash* sebelumnya. Apabila tidak terdapat perubahan, proses render ulang tidak dilakukan. Dengan demikian, manfaat utama *hash comparison* terletak pada efisiensi pemrosesan di sisi klien, khususnya dalam mengurangi penggunaan CPU *browser* dan proses manipulasi DOM yang tidak diperlukan, bukan pada pengurangan beban server.

A. Analisis Kebutuhan Optimasi

Meskipun *polling* mampu memperbarui data secara otomatis, proses tersebut berpotensi menimbulkan render ulang yang tidak diperlukan apabila data yang diterima dari server sebenarnya tidak mengalami perubahan. Tanpa mekanisme optimasi, sistem akan tetap merender ulang seluruh komponen dashboard pada setiap siklus *polling*. Proses ini dapat meningkatkan penggunaan CPU browser, memperlambat performa aplikasi, serta mengurangi kenyamanan pengguna ketika jumlah data pesanan semakin banyak. Oleh karena itu, diperlukan mekanisme yang mampu mendeteksi apakah data yang diterima benar-benar mengalami perubahan sebelum sistem melakukan proses render ulang. Untuk memenuhi kebutuhan tersebut, digunakan teknik *hash comparison*.

B. Implementasi *Check For Updates*

Teknik *hash comparison* diterapkan pada fungsi `*checkForUpdates*()`. Fungsi ini bertugas mengambil data terbaru dari server, menghasilkan nilai hash dari data yang diterima, kemudian membandingkannya dengan hash data sebelumnya. Implementasi fungsi `*checkForUpdates*()` dapat dilihat pada Gambar 3.29.

```
async checkForUpdates() {
    const results = await Promise.allSettled([
        this.api.getOrderQueue(1),
        this.api.getActiveOrders(),
        this.api.getServedOrders(this.currentDate, 1)
    ]);
    const newHash =
this.generateHash(results[0].value.data.queue_orders);
    if (this.lastQueueHash !== newHash) {
```

```
            this.lastQueueHash = newHash;
            this.queueOrders = results[0].value.data.queue_orders;
            this.renderOrderQueue();
        }
    }
```

Gambar 3. 29 Potongan Kode *checkForUpdates.*

Pada tahap awal, fungsi memanggil beberapa endpoint secara bersamaan menggunakan `*Promise.allSettled*()`. Pendekatan ini dipilih karena memungkinkan seluruh permintaan tetap diproses meskipun salah satu *endpoint* mengalami kegagalan. Setelah data berhasil diperoleh, sistem menghasilkan nilai hash baru menggunakan fungsi `*generateHash*()`. Nilai hash tersebut kemudian dibandingkan dengan nilai hash yang tersimpan sebelumnya pada variabel `*lastQueueHash*`. Apabila kedua nilai hash berbeda, sistem menyimpulkan bahwa data telah mengalami perubahan sehingga proses render ulang perlu dilakukan. Sebaliknya, apabila nilai hash sama, sistem tidak melakukan render ulang karena data dianggap masih identik dengan data sebelumnya. Pendekatan ini mampu mengurangi jumlah render yang tidak diperlukan dan meningkatkan efisiensi penggunaan sumber daya browser.

C. Hasil Pengujian *Hash Comparison*

Setelah mekanisme *hash comparison* diimplementasikan pada proses pembaruan data Kitchen Dashboard, dilakukan pengujian untuk mengetahui pengaruh mekanisme tersebut terhadap jumlah proses render pada sisi *frontend*. Pengujian dilakukan dengan membandingkan jumlah pemanggilan fungsi render sebelum dan setelah *hash comparison* diterapkan. Pengukuran dilakukan menggunakan console.count() pada fungsi renderOrderQueue(), renderActiveOrdersDirect(), dan renderServedOrders() sehingga setiap pemanggilan fungsi dapat dihitung melalui *Console* pada *browser*.

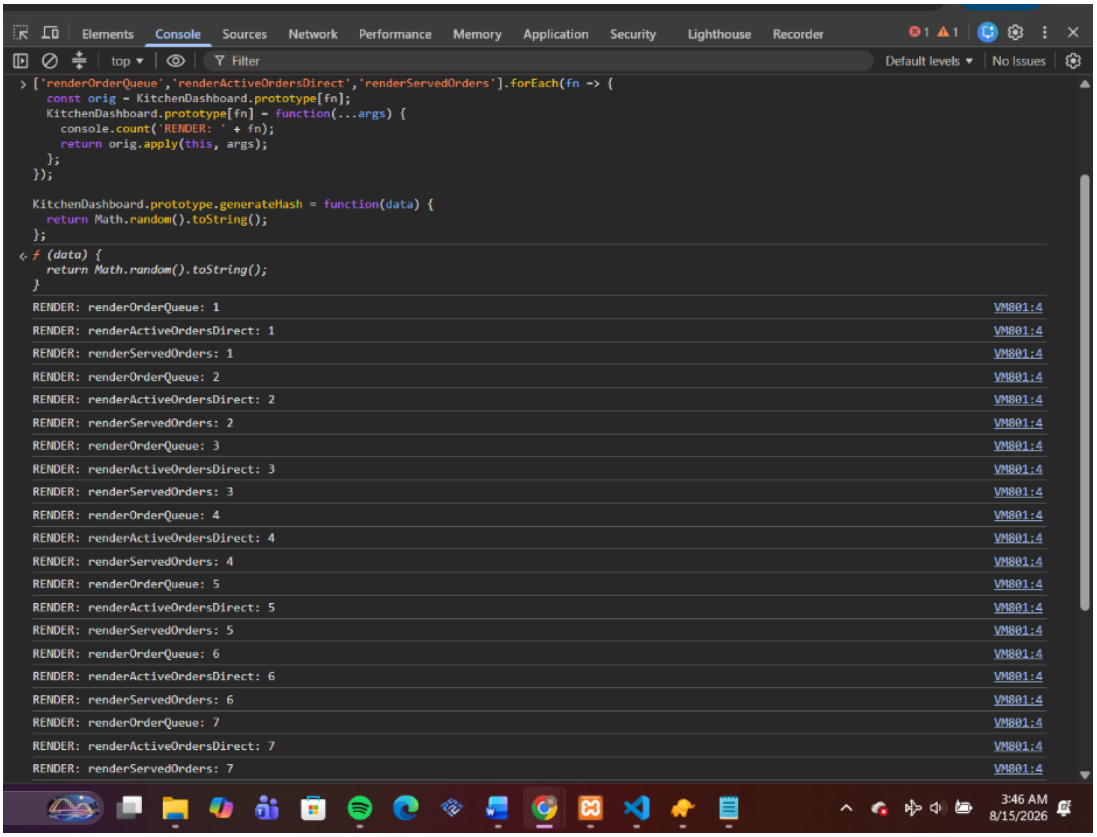
Pengujian dilakukan dalam dua kondisi, yaitu kondisi sebelum penerapan *hash comparison* dan kondisi setelah penerapan *hash comparison*. Kedua kondisi tersebut digunakan untuk mengetahui perubahan jumlah proses *render* ketika sistem menerima pembaruan data secara berkala. Penggunaan *hash comparison* pada Kitchen Dashboard bertujuan untuk membandingkan kondisi data terbaru dengan data sebelumnya sehingga proses *render* hanya dilakukan apabila terdapat perubahan data.

1) Pengujian Sebelum Penerapan Hash Comparison

Pengujian pertama dilakukan sebelum mekanisme *hash comparison* diterapkan. Pada tahap ini, proses *render* tetap dijalankan setiap kali terjadi pembaruan data, tanpa terlebih dahulu memeriksa apakah data yang diterima mengalami perubahan. Untuk

memperoleh data kuantitatif, frekuensi pemanggilan fungsi *render* dipantau menggunakan `console.count()`.

Selama periode pengujian, fungsi `renderOrderQueue()` dipanggil sebanyak **7 kali**, `renderActiveOrdersDirect()` sebanyak **7 kali**, dan `renderServedOrders()` sebanyak **7 kali**. Dengan demikian, total pemanggilan fungsi *render* dari ketiga bagian tersebut mencapai **21 kali**.

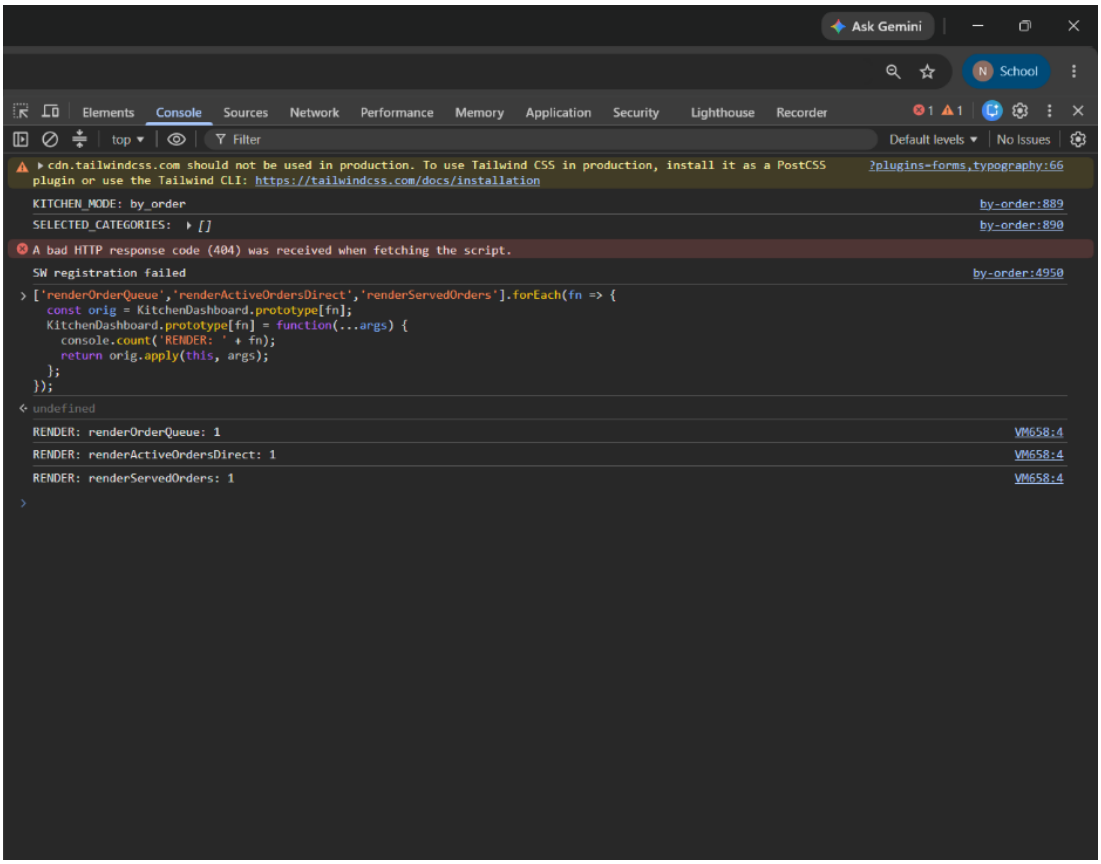


Gambar 3. 30 Hasil Pengujian Jumlah Render Sebelum Penerapan Hash Comparison

Gambar 3.30 memperlihatkan bahwa ketiga fungsi *render* dipanggil berulang selama proses pembaruan data. Masing-masing fungsi menghasilkan tujuh kali pemanggilan, sehingga seluruh bagian antarmuka kembali menjalankan proses *render* pada setiap siklus pembaruan. Hal ini menunjukkan bahwa sebelum penerapan *hash comparison*, sistem belum membedakan antara data yang mengalami perubahan dan data yang tetap sama. Akibatnya, proses *render* tetap dilakukan meskipun pembaruan data tidak selalu menghasilkan perubahan pada tampilan.

2) Pengujian Setelah Penerapan Hash Comparison

Pengujian selanjutnya dilakukan setelah mekanisme *hash comparison* diterapkan. Pada tahap ini, sistem terlebih dahulu membandingkan nilai hash data terbaru dengan nilai hash dari data sebelumnya. Jika tidak ditemukan perubahan pada data, proses *render* tidak dijalankan kembali. Sebaliknya, ketika terdapat perubahan, sistem akan memperbarui tampilan melalui fungsi *render* yang sesuai. Pemantauan menggunakan console.count() menunjukkan bahwa renderOrderQueue() dipanggil sebanyak **1 kali**, renderActiveOrdersDirect() sebanyak **1 kali**, dan renderServedOrders() sebanyak **1 kali**. Dengan demikian, total pemanggilan fungsi render selama periode pengujian hanya mencapai **3 kali**.



Gambar 3. 31 Hasil Pengujian Jumlah Render Setelah Penerapan *Hash Comparison*.

Pada Gambar 3.31 terlihat bahwa setiap fungsi render hanya dipanggil satu kali selama periode pengujian. Hasil ini menunjukkan bahwa mekanisme *hash comparison* mampu mencegah *render* ulang ketika data yang diterima masih sama dengan data sebelumnya. Antarmuka tidak perlu diperbarui pada setiap siklus *polling*, sehingga proses *render* hanya dijalankan ketika sistem mendeteksi adanya perubahan data. Jika

disandingkan dengan pengujian sebelumnya, hasilnya menjadi cukup kuat secara kuantitatif: sebelum penerapan terdapat **21 pemanggilan *render***, sedangkan setelah penerapan hanya 3 pemanggilan *render*. Ini berarti terjadi pengurangan **18 pemanggilan** atau sekitar **85,71%.**

3) Perbandingan Jumlah *Render*

Untuk memperjelas hasil pengujian, perbandingan jumlah pemanggilan fungsi *render* sebelum dan setelah penerapan *hash comparison* disajikan pada Tabel 3.8.

Tabel 3. 8 Perbandingan Jumlah *Render*.

Fungsi Render	Sebelum Hash Comparison	Setelah Hash Comparison	Penurunan
renderOrderQueue()	7	1	6
renderActiveOrdersDirect()	7	1	6
renderServedOrders()	7	1	6
Total	21	3	18

Berdasarkan Tabel 3.8, jumlah pemanggilan fungsi *render* **mengalami penurunan dari 21 kali menjadi 3 kali** setelah mekanisme *hash comparison* diterapkan. Terdapat pengurangan sebanyak 18 pemanggilan fungsi *render* selama periode pengujian. Persentase penurunan jumlah *render* dihitung menggunakan persamaan berikut:

$$\text{Persentase Penurunan} = \frac{\text{Jumlah Render Sebelum} - \text{Jumlah Render Setelah}}{\text{Jumlah Render Sebelum}} \times 100\%$$

$$= \frac{21 - 3}{21} \times 100\%$$

$$= 85,7\%$$

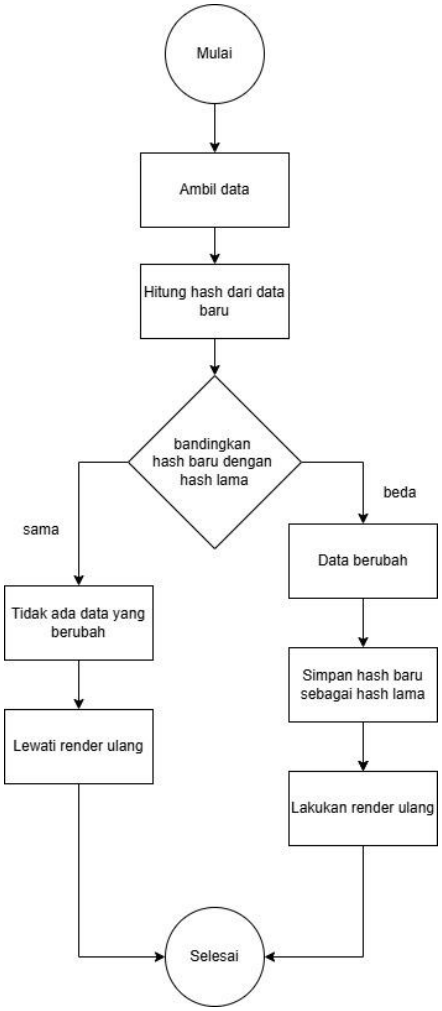
Hasil perhitungan menunjukkan bahwa penerapan *hash comparison* menghasilkan **penurunan jumlah proses *render* sebesar 85,7%**, dari **21 kali** menjadi **3 kali** pada periode pengamatan yang sama. Penurunan tersebut terjadi karena sistem tidak menjalankan proses *render* ulang ketika data terbaru memiliki kondisi yang sama dengan data sebelumnya. Hasil ini menunjukkan bahwa *hash comparison* mampu membuat proses pembaruan antarmuka menjadi lebih selektif. Proses *render* hanya dijalankan ketika terdapat perubahan data, sedangkan data yang tidak mengalami

perubahan tidak menyebabkan pembaruan tampilan kembali. Mekanisme tersebut dapat mengurangi pemrosesan *render* yang tidak diperlukan pada sisi *frontend Kitchen Dashboard*.

Penerapan *hash comparison* juga mendukung mekanisme optimasi lain yang digunakan pada *Kitchen Dashboard*, seperti *polling*, *Page Visibility* API, dan *debounce*. Masing-masing mekanisme memiliki fungsi yang berbeda dalam mengendalikan proses pembaruan data dan interaksi pada *frontend*, sedangkan *hash comparison* secara khusus digunakan untuk menentukan apakah perubahan data memerlukan proses *render* ulang. Berdasarkan hasil pengujian tersebut, dapat disimpulkan bahwa penerapan *hash comparison* memberikan dampak yang terukur terhadap frekuensi proses *render*. Pada skenario pengujian, jumlah pemanggilan fungsi render berkurang dari **21 kali** menjadi **3 kali**, atau mengalami **penurunan sebesar 85,7%**. Hasil ini menunjukkan bahwa mekanisme *hash comparison* dapat mengurangi *render* ulang yang tidak diperlukan ketika data tidak mengalami perubahan.

D. Hasil Optimasi *Hash Comparison*

Penerapan hash comparison berhasil mengurangi proses render ulang yang terjadi selama mekanisme *polling* berjalan. *Dashboard* hanya memperbarui tampilan ketika benar-benar terdapat perubahan data sehingga penggunaan sumber daya menjadi lebih efisien. Selain meningkatkan performa aplikasi, mekanisme ini juga membantu menjaga responsivitas antarmuka ketika jumlah pesanan yang ditampilkan semakin banyak. Alur kerja optimasi *hash comparison* dapat dilihat pada Gambar 3.32.



Gambar 3. 32 Alur Kerja Optimasi *Hash Comparison*.

3.4.3 Implementasi *Page Visibility API*

A. Analisis Penggunaan *Resource*

Pada implementasi *polling* standar, proses pembaruan data akan tetap berjalan meskipun pengguna tidak sedang melihat halaman *dashboard*. Kondisi ini dapat terjadi ketika pengguna berpindah ke tab browser lain atau meminimalkan jendela aplikasi. Apabila *polling* tetap berjalan dalam kondisi tersebut, browser dan server akan terus memproses permintaan yang sebenarnya tidak memberikan manfaat langsung kepada pengguna. Akibatnya, penggunaan *bandwidth*, CPU, dan memori menjadi kurang efisien. Untuk mengatasi permasalahan tersebut, diperlukan mekanisme yang dapat mendeteksi apakah halaman sedang aktif atau tidak aktif sehingga *polling* hanya dijalankan ketika benar-benar dibutuhkan.

Berbeda dengan *hash comparison*, *Page Visibility* API bekerja pada mekanisme pembaruan data sebelum proses request *polling* berikutnya dilakukan. Ketika halaman tidak aktif, sistem

menghentikan proses *polling* sehingga *request* ke server tidak lagi dikirim selama halaman berada pada kondisi tersebut. Ketika halaman kembali aktif, proses *polling* dijalankan kembali. Oleh karena itu, *Page Visibility* API tidak berfokus pada pengurangan proses *render* di *browser*, tetapi pada pengurangan aktivitas *polling* yang tidak diperlukan. Dampaknya dapat terlihat pada berkurangnya jumlah *request* HTTP, konsumsi *bandwidth*, serta beban pemrosesan pada server dan browser ketika *dashboard* tidak sedang digunakan.

B. Implementasi *Page Visibility API*

Page Visibility API digunakan untuk memantau status visibilitas halaman pada browser. Ketika halaman tidak lagi terlihat oleh pengguna, sistem akan menghentikan *polling* secara otomatis. Sebaliknya, ketika halaman kembali aktif, *polling* akan dijalankan kembali. Implementasi *Page Visibility API* dapat dilihat pada Gambar 3.33.

```
document.addEventListener('visibilitychange', () => {
    if (document.hidden) {
        window.kitchenDashboard.stopRealtimePolling();
    } else {
        window.kitchenDashboard.startRealtimePolling();
    }
});
```

Gambar 3. 33 Potongan Kode *Page Visibility API.*

Kode tersebut memanfaatkan event `*visibilitychange*` yang akan dipicu setiap kali status visibilitas halaman berubah. Ketika nilai `*document.hidden*` bernilai *true*, sistem menganggap halaman sedang berada di latar belakang sehingga fungsi `*stopRealtimePolling*()` dipanggil untuk menghentikan proses *polling*. Sebaliknya, ketika halaman kembali aktif dan `*document.hidden*` bernilai *false*, fungsi `startRealtimePolling()` dijalankan kembali untuk melanjutkan proses pembaruan data. Dengan pendekatan ini, *polling* hanya berjalan ketika *dashboard* benar-benar digunakan oleh pengguna sehingga penggunaan sumber daya menjadi lebih efisien.

C. Pengujian Kuantitatif Jumlah Request per Menit

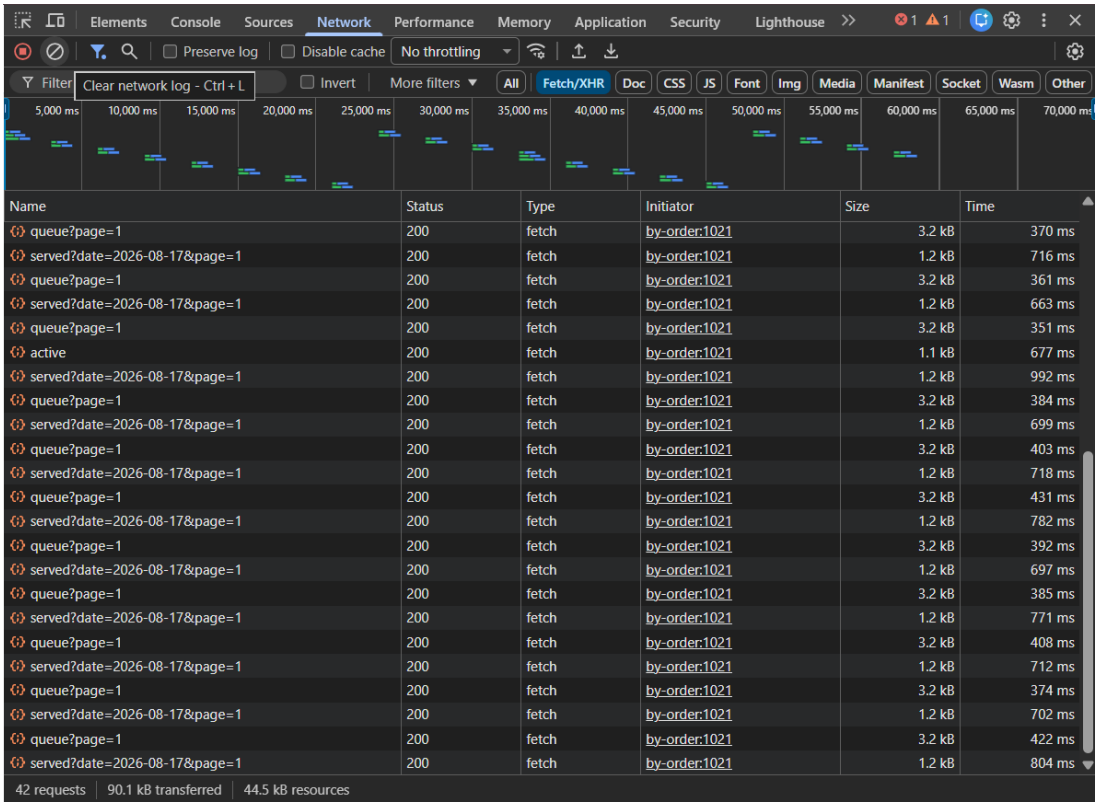
Untuk memvalidasi efektivitas Page Visibility API dalam mengurangi aktivitas *polling* yang tidak diperlukan, dilakukan pengujian dengan menghitung jumlah *request* HTTP yang dikirimkan ke *endpoint* orders/queue, orders/active, dan orders/served ketika halaman berada pada kondisi tidak aktif (*hidden*). Pengujian dilakukan dalam dua kondisi, yaitu kondisi sebelum dan sesudah penerapan *Page Visibility* API. Pada kondisi sebelum penerapan *Page Visibility* API, fungsi stopRealtimePolling() dinonaktifkan sehingga mekanisme *polling* tetap berjalan meskipun halaman berada dalam kondisi tidak aktif. Sementara itu, pada kondisi

sesudah penerapan, digunakan implementasi asli dengan *Page Visibility* API yang menghentikan proses polling ketika halaman tidak sedang terlihat.

Pengamatan jumlah request dilakukan menggunakan panel *Network* pada *Chrome DevTools* dengan memilih kategori *Fetch/XHR*. Jumlah *request* yang tercatat selama periode pengujian kemudian digunakan sebagai data kuantitatif untuk membandingkan aktivitas jaringan sebelum dan sesudah penerapan Page Visibility API.

1) Pengujian Sebelum Penerapan *Page Visibility* API

Pada kondisi sebelum penerapan *Page Visibility* API, mekanisme *polling* tetap berjalan ketika tab berada dalam kondisi tidak aktif. Berdasarkan hasil pengamatan pada panel *Network*, terdapat 42 *request* yang dikirimkan selama periode pengujian. Request tersebut berasal dari beberapa *endpoint* yang digunakan dalam proses pembaruan data Kitchen Dashboard, yaitu orders/queue, orders/served, dan orders/active.



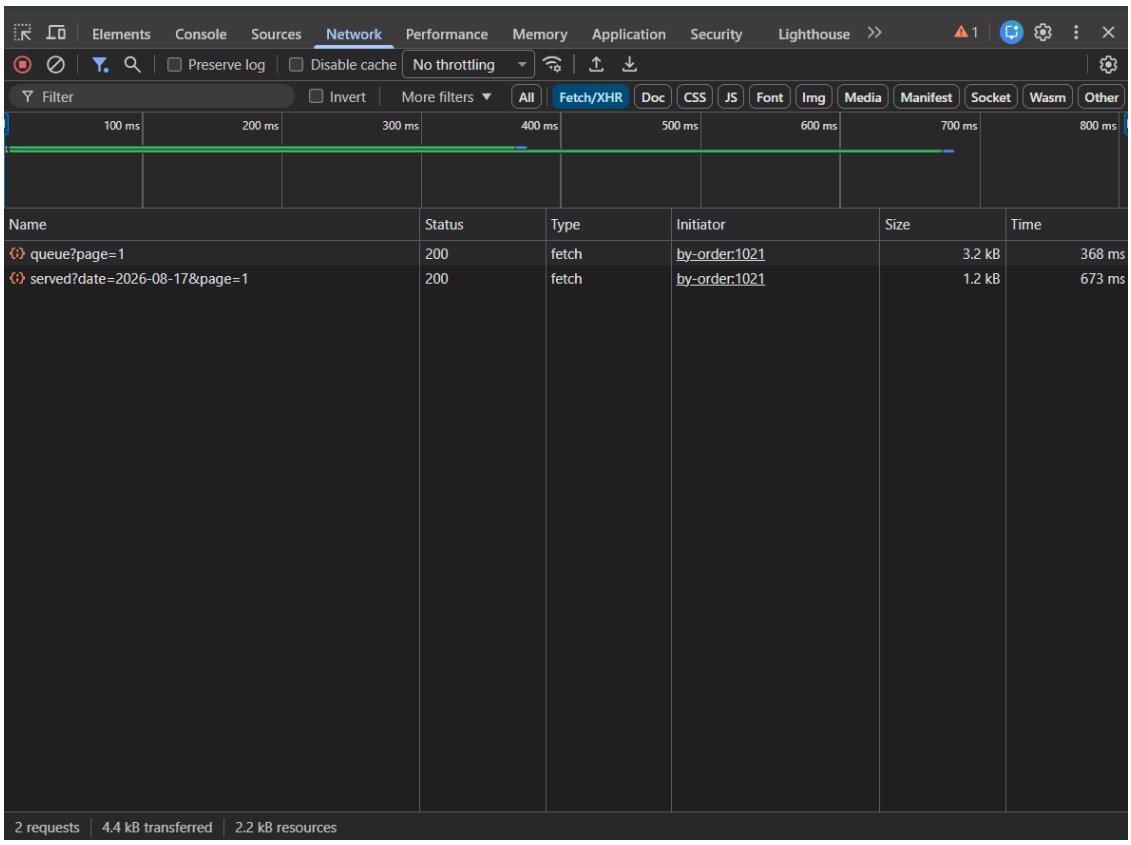
Gambar 3. 34 Hasil Pengujian Jumlah Request Sebelum Penerapan Page Visibility API.

Berdasarkan Gambar 3.34, terlihat bahwa request tetap dikirimkan secara berkala meskipun halaman tidak sedang aktif digunakan. Pada panel *Network* tercatat sebanyak 42 *request* selama periode pengamatan. Kondisi tersebut menunjukkan bahwa

tanpa mekanisme penghentian *polling* berdasarkan visibilitas halaman, sistem tetap melakukan komunikasi dengan *server* ketika pengguna tidak sedang melihat *dashboard*.

2) Pengujian Setelah Penerapan Page Visibility API

Pengujian selanjutnya dilakukan menggunakan implementasi asli *Page Visibility* API. Pada kondisi ini, ketika *browser* mendeteksi bahwa halaman berada dalam kondisi tidak aktif (*hidden*), fungsi stopRealtimePolling() dijalankan sehingga proses polling dihentikan. Berdasarkan hasil pengamatan menggunakan panel *Network*, setelah *Page Visibility* API aktif hanya terdapat 2 request yang tercatat selama periode pengujian.



Gambar 3. 35 Hasil Pengujian Jumlah *Request* Setelah Penerapan *Page Visibility* API.

Berdasarkan Gambar 3.35, terlihat bahwa setelah halaman berada dalam kondisi tidak aktif, tidak terjadi pengiriman *request* secara berkala seperti pada kondisi sebelum penerapan *Page Visibility* API. Pada panel *Network* hanya tercatat 2 *request* selama periode pengamatan. Dua *request* tersebut merupakan *request* yang masih tercatat pada saat proses perubahan kondisi halaman atau proses awal sebelum mekanisme

penghentian *polling* sepenuhnya berlaku. Setelah proses tersebut, tidak terlihat adanya pola pengiriman *request* secara berkala seperti pada kondisi sebelum penerapan *Page Visibility* API.

3) Perbandingan Jumlah Request

Perbandingan jumlah request sebelum dan sesudah penerapan *Page Visibility* API ditunjukkan pada Tabel 3.9.

Tabel 3. 9 Perbandingan Jumlah *Request*.

Kondisi	Kondisi Tab	Jumlah Request
Sebelum Page Visibility API	Tidak aktif (*hidden*)	42
Setelah Page Visibility API	Tidak aktif (*hidden*)	2
Penurunan		40

Berdasarkan Tabel 3.9, jumlah *request* mengalami penurunan dari 42 *request* menjadi 2 *request* setelah *Page Visibility* API diterapkan. Hasilnya terdapat pengurangan sebanyak 40 *request* selama periode pengujian. Persentase penurunan jumlah *request* dihitung menggunakan persamaan berikut:

$$\text{Persentase Penurunan} = \frac{\text{Request Sebelum} - \text{Request Sesudah}}{\text{Request Sebelum}} \times 100\%$$

$$= \frac{42 - 2}{42} \times 100\%$$

$$= 95,24\%$$

Hasil pengujian penerapan *Page Visibility* API menghasilkan **penurunan jumlah *request* sebesar 95,24%** pada kondisi halaman tidak aktif, yaitu dari 42 *request* menjadi 2 *request*. Hasil tersebut menunjukkan bahwa penerapan *Page Visibility* API mampu menghentikan aktivitas *polling* ketika halaman tidak sedang terlihat oleh pengguna. Penurunan jumlah request menunjukkan bahwa komunikasi antara *frontend* dan *backend* yang tidak diperlukan ketika dashboard tidak sedang digunakan dapat dikurangi secara signifikan. Penerapan *Page Visibility* API tidak hanya menunjukkan keberhasilan secara fungsional dalam mendeteksi kondisi visibilitas halaman, tetapi

juga menghasilkan temuan kuantitatif berupa **penurunan jumlah request sebesar 95,24%** pada skenario pengujian yang dilakukan.

D. Hasil Implementasi

Hasil implementasi menunjukkan bahwa *Page Visibility* API yang diterapkan pada *Kitchen Dashboard* berfungsi sesuai dengan yang diharapkan. *Polling* berhasil dihentikan secara otomatis ketika pengguna berpindah ke tab lain dan kembali aktif ketika pengguna membuka kembali halaman *Dashboard*. Mekanisme ini mampu mengurangi jumlah permintaan HTTP yang tidak diperlukan sekaligus menurunkan beban kerja browser dan server. Dengan adanya optimasi ini, penggunaan *bandwidth* dan sumber daya sistem menjadi lebih efisien, terutama pada kondisi ketika *Dashboard* tidak sedang digunakan secara aktif oleh staff dapur. Hal ini membuktikan bahwa pendekatan *Page Visibility* API mampu mendukung operasional Kitchen Dashboard secara lebih optimal, baik dari sisi performa maupun efisiensi penggunaan sumber daya.

3.4.4 Implementasi Debounce

A. Analisis Kebutuhan Pencarian

Kitchen Dashboard menyediakan fitur pencarian yang digunakan untuk membantu *staff* dapur menemukan pesanan tertentu dengan lebih cepat. Namun, apabila proses pencarian dijalankan setiap kali pengguna menekan tombol *keyboard*, sistem akan melakukan proses filter secara berulang dalam waktu yang sangat singkat. Kondisi tersebut dapat menimbulkan penggunaan sumber daya yang berlebihan, terutama ketika jumlah data pesanan yang diproses cukup besar atau pengguna mengetik dengan kecepatan tinggi. Untuk meningkatkan efisiensi proses pencarian, diperlukan mekanisme yang dapat menunda eksekusi fungsi pencarian hingga pengguna selesai mengetik.

B. Implementasi Debounce

Optimasi pencarian dilakukan menggunakan teknik *debounce* dengan waktu tunda sebesar 300 milidetik. Teknik ini memastikan bahwa proses pencarian hanya dijalankan setelah pengguna berhenti mengetik dalam rentang waktu tertentu. Implementasi *debounce* dapat dilihat pada Gambar 3.36.

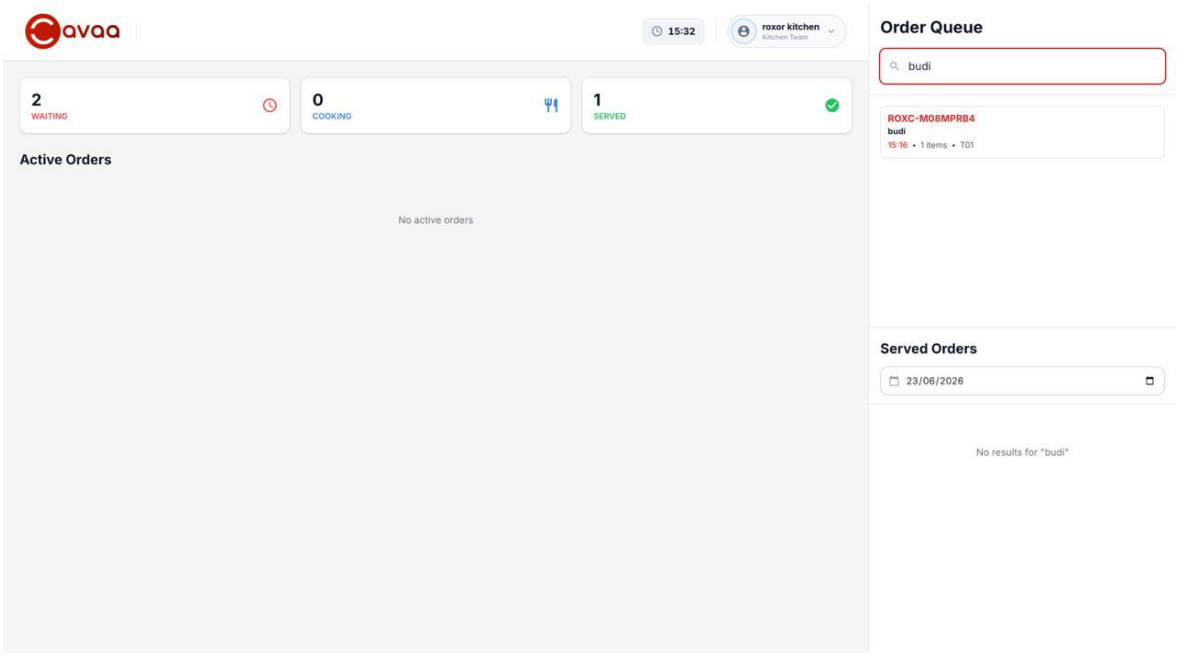
```
newInput.addEventListener('input', (e) => {
    clearTimeout(timer);
    timer = setTimeout(() => {
        this.searchTerm = (e.target.value || '').toLowerCase().trim();
        this.handleSearch();
    }, 300);
});
```

Gambar 3. 36 Potongan Kode *Debounce*.

Setiap kali pengguna mengetik pada kolom pencarian, sistem terlebih dahulu menghapus timer sebelumnya menggunakan fungsi `*clearTimeout()*`. Setelah itu, sistem membuat *timer* baru menggunakan `*setTimeout()*`. Apabila pengguna kembali mengetik sebelum waktu 300 milidetik berakhir, timer sebelumnya akan dibatalkan dan digantikan dengan timer yang baru. Fungsi `*handleSearch()*` hanya akan dijalankan ketika pengguna benar-benar berhenti mengetik. Pendekatan ini mampu mengurangi jumlah proses pencarian yang tidak diperlukan dan meningkatkan efisiensi aplikasi secara keseluruhan.

C. Hasil Implementasi Debounce

Setelah mekanisme debounce diterapkan, proses pencarian menjadi lebih responsif dan tidak lagi melakukan filter data pada setiap karakter yang diketikkan pengguna. Pengujian menunjukkan bahwa jumlah eksekusi fungsi pencarian berkurang secara signifikan tanpa memengaruhi pengalaman pengguna. Fitur pencarian tetap memberikan hasil yang cepat sekaligus mengurangi beban pemrosesan pada browser. Hasil implementasi debounce pada fitur pencarian dapat dilihat pada Gambar 3.37.



Gambar 3. 37 Fitur Pencarian Data Antrian Pesanan.

3.5 Evaluasi Sistem

A. Hasil Pengujian dengan Black-Box

Pengujian *black-box* dilakukan untuk memastikan setiap fitur pada Kitchen Dashboard berjalan sesuai skenario penggunaan yang diharapkan. Hasil pengujian ditunjukkan pada Tabel 3.10.

Tabel 3.10 Hasil Pengujian dengan Black-Box.

No.	Skenario Pengujian	Hasil yang Diharapkan	Kesimpulan
1	Menampilkan Order Queue	Membuka halaman Kitchen Dashboard	Valid
2	Pickup - proses normal	Klik pesanan pada Order Queue, lalu klik Start Cooking pada pesanan yang belum diproses kasir	Valid
3	Pickup - pesanan sedang diproses kasir	Klik pesanan dengan cashier_process_id yang sudah terisi	Valid
4	Menampilkan Active Orders	Membuka atau memperbarui data Kitchen Dashboard saat terdapat pesanan yang sedang diproses	Valid
5	Serve - seluruh item selesai	Membuka detail pesanan aktif, mencentang seluruh checklist item, lalu klik Serve	Valid
6	Serve - item belum lengkap	Membuka detail pesanan aktif dan hanya mencentang sebagian checklist item	Valid
7	Cancel - konfirmasi disetujui	Klik tombol Cancel pada pesanan aktif, lalu klik tombol konfirmasi	Valid
8	Cancel - konfirmasi dibatalkan	Klik tombol Cancel, lalu menutup modal tanpa melakukan konfirmasi	Valid
9	Search - berdasarkan nama pelanggan	Memasukkan nama pelanggan yang terdapat pada data pesanan	Valid
10	Search - berdasarkan kode booking	Memasukkan kode booking yang terdapat pada data pesanan	Valid
11	Search - kata kunci tidak ditemukan	Memasukkan kata kunci yang tidak terdapat pada data pesanan	Valid
12	Load More Order Queue	Klik "Show X more orders" pada bagian Order Queue	Valid
13	Filter tanggal Served Orders	Memilih tanggal pada *date picker* di Served Orders	Valid
14	Refresh data secara manual	Klik tombol Refresh pada Kitchen Dashboard	Valid
15	Pembaruan data otomatis	Menunggu beberapa detik setelah terjadi perubahan data dari sisi lain	Valid

B. Hasil Kuesioner Kepuasan Pengguna

Selain pengujian fungsional, dilakukan pula pengujian kepuasan pengguna terhadap Kitchen Dashboard menggunakan kuesioner berskala Likert 1 hingga 5, di mana nilai 1 berarti sangat tidak setuju dan nilai 5 berarti sangat setuju. Struktur pernyataan kuesioner diadaptasi dari kerangka System Usability Scale (SUS), namun seluruh sepuluh pernyataan disusun bernada positif agar lebih mudah dipahami oleh responden. Karena tidak menggunakan pernyataan berselang-seling positif-negatif sebagaimana format SUS baku, rumus skor SUS standar tidak diterapkan; hasil dilaporkan berupa skor Likert per pernyataan. Kuesioner diberikan kepada *supervisor* setelah mereka mencoba menggunakan sistem secara langsung. Hasil kuesioner ditunjukkan pada Tabel 3.10

Tabel 3.11 Hasil Kuesioner.

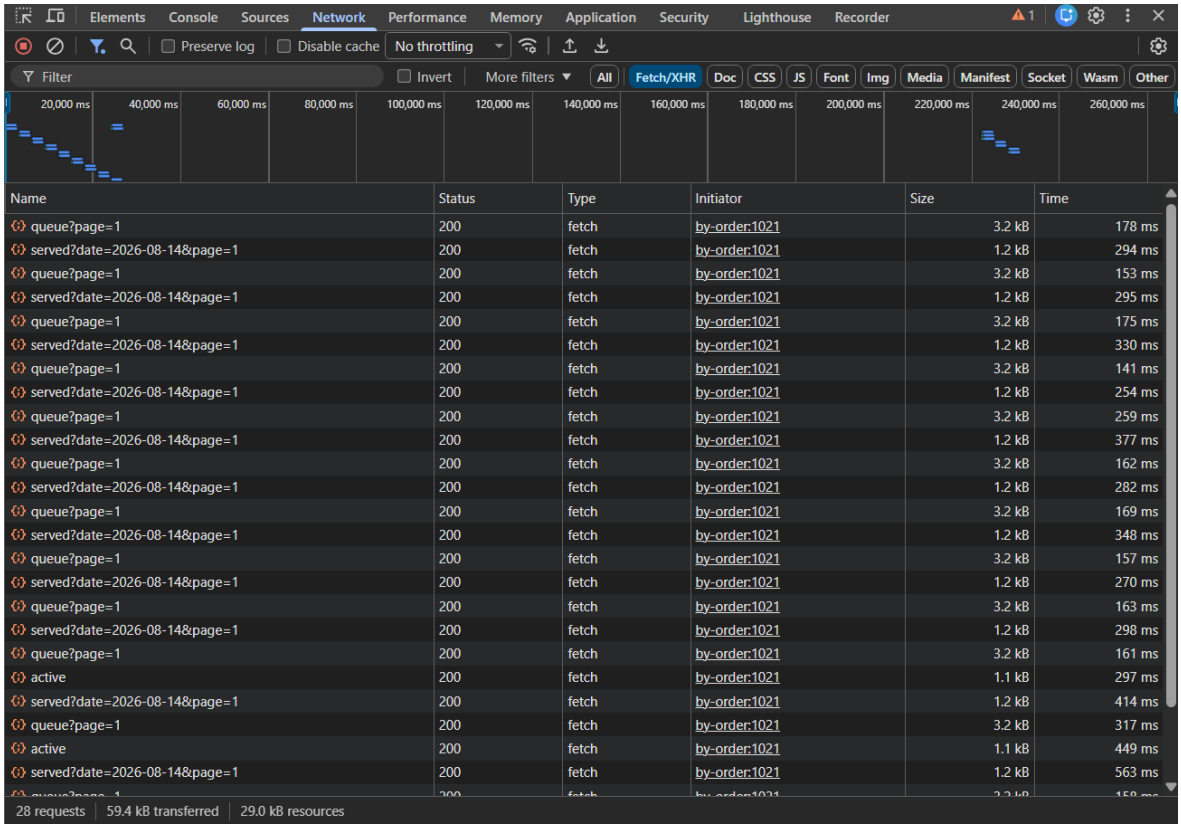
No	The System Usability Scale Standard Version	1	2	3	4	5
1.	Saya merasa sistem kitchen mudah digunakan					√
2.	Saya merasa proses penggunaan sistem kitchen berjalan dengan lancar				√	
3.	Saya merasa proses mengambil pesanan dari antrean dan menandainya sebagai sudah disajikan sangat mudah dilakukan hanya dengan beberapa klik				√	
4.	Saya menemukan bahwa fitur filter tanggal, paginasi antrean, dan pembaruan otomatis terintegrasi dengan sangat baik dalam satu tampilan kitchen				√	
5.	Saya merasa bahwa jumlah antrean, pesanan aktif, dan total pesanan selesai yang ditampilkan sistem ini akurat dan terkini				√	
6.	Saya merasa tampilan kartu pesanan aktif dengan nomor antrean (#1, #2, dst.) sangat membantu saya memprioritaskan pesanan mana yang harus diselesaikan terlebih dahulu.					√
7.	Saya merasa fitur cancel yang mengembalikan pesanan ke antrean memberikan saya fleksibilitas untuk mengoreksi kesalahan tanpa harus memulai proses dari awal.				√	
8.	Saya merasa tata letak tiga kolom (antrean, aktif, selesai) pada layar dasbor memungkinkan saya memantau seluruh				√	

	status pesanan hanya dalam satu tampilan tanpa berpindah halaman.					
9.	Saya merasa sidebar mobile dengan tab antrean dan selesai memudahkan saya memantau dan mengelola pesanan meskipun hanya menggunakan perangkat smartphone.				√	
10.	Saya merasa tombol konfirmasi sebelum menandai pesanan sebagai "Served" membantu mencegah kesalahan.				√	

C. Analisis Request dan Waktu Respons Endpoint

Gambar 3.38 menunjukkan adanya pola *request* yang dilakukan secara berkala sebagai konsekuensi dari penerapan mekanisme *polling* pada Kitchen Dashboard. Mekanisme tersebut digunakan untuk memperoleh pembaruan data secara berkala dari beberapa *endpoint*. Namun, tidak seluruh *endpoint* menghasilkan *request* baru pada setiap siklus *polling*. *Endpoint* kitchen/orders/active, yang digunakan untuk memperoleh data pesanan aktif, menerapkan mekanisme *caching* melalui metode _fetchWithCache(). Ketika data masih berada dalam periode *cache*, *frontend* menggunakan data yang telah tersimpan sehingga tidak perlu mengirimkan *request* HTTP baru ke *server*.

Pada implementasinya, nilai *cacheTimeout* untuk data pesanan aktif adalah **30 detik**. Apabila proses *polling* berlangsung sebelum periode *cache* tersebut berakhir, pemanggilan *endpoint* tidak selalu menghasilkan *request* baru yang dapat diamati melalui panel *Network*. Kondisi tersebut menjelaskan mengapa *endpoint* kitchen/orders/active tidak terlihat pada setiap siklus request meskipun fungsi checkForUpdates() tetap memanggil getActiveOrders(). Pengambilan data pada kondisi tersebut dilakukan melalui *cache* pada sisi *frontend*.



Gambar 3. 38 Hasil Pengamatan *Request* dan Waktu *Respons Endpoint* melalui *Chrome DevTools*

Berdasarkan Gambar 3.38, *request* yang ditampilkan pada panel *Network* memperoleh status 200, yang menunjukkan bahwa *request* berhasil diproses. Kolom *Time* menunjukkan waktu yang dibutuhkan untuk menyelesaikan masing-masing *request* dan digunakan sebagai dasar pengambilan sampel waktu *respons*. Untuk memperoleh data kuantitatif, dilakukan pengukuran terhadap empat sampel waktu *respons* pada *endpoint* orders/queue dan orders/served. *Endpoint* orders/active tidak digunakan sebagai sampel utama karena penerapan mekanisme *caching* selama 30 detik menyebabkan endpoint tersebut tidak selalu menghasilkan *request* HTTP baru pada setiap siklus *polling*.

Tabel 3.12 Hasil Pengukuran Waktu Respons Endpoint.

Endpoint	**Sampel 1**	**Sampel 2**	**Sampel 3**	**Sampel 4**	**Rata - Rata**
orders/queue	178 ms	153 ms	175 ms	141 ms	161,75 ms
orders/served	294 ms	295 ms	330 ms	254 ms	293,25 ms

Berdasarkan Tabel 3.10, *endpoint* orders/queue memiliki waktu *respons* sebesar 178 ms, 153 ms, 175 ms, dan 141 ms dengan rata-rata 161,75 ms. Sementara itu, *endpoint* orders/served memiliki waktu *respons* sebesar 294 ms, 295 ms, 330 ms, dan 254 ms dengan rata-rata 293,25 ms. Rata-rata tersebut diperoleh dengan menjumlahkan seluruh sampel waktu *respons* kemudian membaginya dengan jumlah sampel.

Rata-rata waktu *respons* orders/queue adalah:

$$\frac{178 + 153 + 175 + 141}{4} = 161,75 \text{ ms}$$

Sedangkan rata-rata waktu *respons* orders/served adalah:

$$\frac{294 + 295 + 330 + 254}{4} = 293,25 \text{ ms}$$

Hasil tersebut menunjukkan bahwa *endpoint* orders/served memiliki rata-rata waktu *respons* 131,5 ms lebih tinggi dibandingkan *endpoint* orders/queue. Perbedaan waktu *respons* tersebut menunjukkan adanya perbedaan waktu penyelesaian *request* pada kedua *endpoint* berdasarkan sampel yang diperoleh selama pengujian. Namun, hasil pengujian ini hanya menunjukkan waktu *respons* yang terukur melalui panel *Network* dan tidak digunakan untuk menyimpulkan secara spesifik penyebab perbedaan waktu *respons* pada sisi *backend*.

Selain waktu *respons*, pengamatan pada panel *Network* juga menunjukkan penerapan mekanisme *caching* pada *endpoint* kitchen/orders/active. Dengan durasi *cache* selama 30 detik, data pesanan aktif yang masih berada dalam periode tersebut dapat digunakan kembali oleh *frontend* tanpa melakukan *request* HTTP baru. Oleh karena itu, tidak setiap pemanggilan getActiveOrders() menghasilkan request baru yang terlihat pada panel *Network*. Hal tersebut menjelaskan mengapa *endpoint active* tidak muncul pada setiap siklus *polling* seperti *endpoint* lainnya.

Hasil pengamatan *Network* memberikan dua temuan kuantitatif pada proses komunikasi data *Kitchen Dashboard*. Pertama, *endpoint* orders/queue memiliki rata-rata waktu *respons* sebesar 161,75 ms, sedangkan orders/served memiliki rata-rata sebesar 293,25 ms, dengan selisih rata-rata 131,5 ms. Kedua, penerapan *caching* selama 30 detik pada kitchen/orders/active menyebabkan tidak setiap siklus *polling* menghasilkan request HTTP baru. Hasil tersebut menunjukkan bahwa mekanisme *caching* dapat mengurangi pengiriman *request* yang tidak diperlukan ketika data masih tersedia dalam periode *cache*.

Penerapan *caching* tersebut melengkapi mekanisme optimasi lainnya pada *Kitchen Dashboard*, seperti *hash comparison* yang membatasi proses *render* ulang ketika data tidak mengalami perubahan. Dengan kombinasi mekanisme tersebut, pembaruan data tetap dapat dilakukan secara berkala, sementara proses pengambilan data dan *rendering* tidak selalu dilakukan secara penuh pada setiap siklus *polling*.

D. Kesesuaian dengan Kebutuhan Sistem

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, seluruh kebutuhan sistem yang telah diidentifikasi pada tahap analisis berhasil direalisasikan dengan baik. Sistem mampu menampilkan informasi pesanan secara *real-time*, mengelola perubahan status pesanan, menampilkan detail pesanan secara lengkap, serta menyediakan antarmuka yang responsif pada berbagai ukuran perangkat. Selain kebutuhan fungsional, kebutuhan non-fungsional seperti kemudahan penggunaan, performa sistem, dan efisiensi pembaruan data juga berhasil dipenuhi melalui penerapan berbagai mekanisme optimasi yang telah dikembangkan.

E. Keunggulan Sistem

Kitchen Dashboard yang dikembangkan memiliki beberapa keunggulan dibandingkan pendekatan dashboard konvensional yang mengandalkan penyegaran halaman secara manual. Sistem mampu memperbarui data secara otomatis melalui mekanisme *polling* sehingga informasi pesanan selalu berada dalam kondisi terbaru. Penerapan *hash comparison* berhasil mengurangi render ulang yang tidak diperlukan, sementara penggunaan *Page Visibility API* membantu mengurangi beban server ketika dashboard tidak sedang digunakan. Implementasi *debounce* pada fitur pencarian meningkatkan efisiensi proses pencarian tanpa mengurangi kenyamanan pengguna. Kombinasi berbagai mekanisme tersebut menghasilkan aplikasi yang responsif, ringan, dan mampu mendukung operasional dapur secara lebih efektif.

F. Kendala Pengembangan

Selama proses pengembangan sistem, terdapat beberapa kendala teknis yang perlu diselesaikan. Salah satu kendala utama adalah menjaga konsistensi nomor urut pesanan aktif ketika terjadi perubahan data secara bersamaan antara tindakan pengguna dan proses *polling* otomatis. Permasalahan tersebut muncul ketika pesanan dibatalkan atau diselesaikan pada saat sistem sedang melakukan pembaruan data dari server. Kondisi ini berpotensi menyebabkan perbedaan data antara state lokal pada *frontend* dan data yang tersimpan pada *backend*. Untuk mengatasi permasalahan tersebut, diterapkan mekanisme *hash comparison* dan pengelolaan

state yang lebih terstruktur sehingga proses pembaruan data hanya dilakukan ketika benar-benar terdapat perubahan pada data pesanan. Solusi tersebut berhasil menjaga konsistensi tampilan *dashboard* sekaligus mempertahankan performa sistem selama digunakan dalam operasional sehari-hari.

BAB IV
REFLEKSI PELAKSANAAN MAGANG

4.1 Relevansi Akademik

Bab ini berisi refleksi penulis terhadap pengalaman magang yang telah dilaksanakan di PT. Vastu Cipta Persada, khususnya pada pengembangan *Kitchen Dashboard*. Refleksi ini akan membandingkan teori-teori yang telah dipelajari di bangku kuliah (sebagaimana diuraikan dalam Bab II) dengan praktik dan kenyataan yang ditemui di lapangan. Bab ini juga akan membahas berbagai pembelajaran, baik dari sisi teknis maupun non-teknis, yang diperoleh selama proses pengembangan sistem berlangsung. Sub-bab ini menganalisis kesesuaian dan perbedaan antara teori atau konsep akademis yang dipelajari dengan implementasi nyata selama kegiatan magang. Analisis ini bertujuan untuk mengidentifikasi kesenjangan yang terjadi dan memberikan pelajaran berharga bagi pengembangan ilmu pengetahuan dan praktik di industri.

4.1.1 Relevansi Penerapan REST API

Pada Bab II, dijelaskan bahwa REST API merupakan gaya arsitektur yang memungkinkan pertukaran data antara aplikasi yang berbeda melalui protokol HTTP. Dalam teori, REST API diharapkan dapat menjadi jembatan yang fleksibel dan efisien antara *frontend* dan *backend*. Pada praktiknya, implementasi REST API pada proyek *Kitchen Dashboard* sangat sesuai dengan teori tersebut.

Kitchen Dashboard yang dikembangkan memanfaatkan REST API sebagai tulang punggung komunikasi data. *Frontend* yang dibangun dengan JavaScript mengirimkan permintaan HTTP (GET, PUT) ke berbagai *endpoint* yang telah disediakan oleh *backend* Laravel, seperti yang terlihat pada Tabel 3.7. Server kemudian memproses permintaan, berinteraksi dengan basis data, dan mengembalikan respons dalam format JSON yang ringan dan mudah diproses. Hal ini membuktikan bahwa REST API berhasil memisahkan tanggung jawab antara *frontend* (penyajian antarmuka) dan *backend* (logika bisnis dan pengelolaan data), sehingga kedua sisi dapat dikembangkan secara lebih independen. Namun, dalam praktiknya, kami menemukan bahwa pembuatan *endpoint* dan konsistensi struktur data JSON membutuhkan perencanaan yang matang dan komunikasi yang intensif antara tim *frontend* dan *backend* untuk menghindari kesalahan interpretasi data. Hal ini menjadi sebuah pembelajaran

bahwa meskipun REST API menawarkan fleksibilitas, koordinasi tim tetaplah kunci utama keberhasilannya.

4.1.2 Relevansi Optimasi Kinerja Frontend

Teori di Bab II menyebutkan beberapa teknik optimasi seperti *polling*, *hash comparison*, *debounce*, dan *Page Visibility API* untuk meningkatkan efisiensi aplikasi web. Pada praktiknya, keempat teknik ini diimplementasikan secara konkret untuk mengatasi tantangan pada *Kitchen Dashboard* yang membutuhkan pembaruan data secara *real-time*.

a. *Polling*

Teori menyatakan bahwa *polling* adalah teknik pengambilan data secara berkala. Di lapangan, kami menerapkan *polling* dengan interval 3 detik untuk memperbarui data pesanan. Hal ini sesuai dengan teori dan terbukti efektif untuk kebutuhan *real-time* pada operasional dapur. Namun, kami menyadari bahwa interval yang terlalu singkat dapat meningkatkan beban server. Oleh karena itu, kami menyeimbangkannya dengan teknik optimasi lainnya.

b. Hash Comparison

Teori menjelaskan bahwa *hash comparison* digunakan untuk membandingkan kondisi data dan menghindari render ulang yang tidak perlu. Pada implementasi *Kitchen Dashboard*, teknik ini terbukti sangat krusial. Fungsi `*checkForUpdates*()` pada Gambar 3.28 berhasil mengurangi proses render yang tidak diperlukan, sehingga penggunaan CPU browser menjadi lebih efisien, terutama ketika jumlah pesanan mulai banyak. Ini adalah bukti kuat bahwa teori optimasi *frontend* memiliki dampak signifikan dalam praktik.

c. Page Visibility API

Teori menyebutkan API ini untuk mendeteksi status halaman. Implementasinya di *Kitchen Dashboard*, seperti pada Gambar 3.30, berhasil menghentikan *polling* saat tab browser tidak aktif. Hal ini sesuai dengan teori dan secara langsung mengurangi konsumsi *bandwidth* dan beban server yang tidak berguna, menjadikan sistem lebih ramah sumber daya.

d. Debounce

Teori menjelaskan *debounce* sebagai teknik untuk menunda eksekusi fungsi. Pada fitur pencarian di *Kitchen Dashboard*, penerapan *debounce* dengan waktu tunda 300ms, seperti pada Gambar 3.33, berhasil mengurangi jumlah permintaan pemrosesan data

secara signifikan. Hal ini membuktikan bahwa teori optimasi *user experience* melalui pengurangan beban komputasi dapat diimplementasikan dengan baik dan memberikan hasil yang terukur.

Secara keseluruhan, penerapan teknik-teknik optimasi ini di lapangan tidak hanya sesuai dengan teori, tetapi juga menunjukkan bahwa pemahaman mendalam tentang teori sangat penting untuk menyelesaikan masalah nyata di industri, khususnya dalam membangun aplikasi yang responsif dan efisien.

4.1.3 Relevansi Pengujian Sistem

Berdasarkan tinjauan pustaka, metode pengujian seperti *System Integration Testing* (SIT) dan *User Acceptance Testing* (UAT) adalah langkah penting untuk memastikan kualitas perangkat lunak. SIT berfokus pada interoperabilitas antar modul, sedangkan UAT melibatkan pengguna akhir untuk memastikan sistem memenuhi kebutuhan mereka. Dalam proyek *Kitchen Dashboard*, proses pengujian yang dilakukan mencakup pengujian fungsionalitas setiap fitur (seperti yang dijelaskan di Bab III) yang secara konsep mirip dengan SIT, yaitu memastikan setiap *endpoint* dan fitur berjalan sesuai yang diharapkan.

Namun, terdapat kesenjangan (*gap*) yang signifikan dalam pelaksanaan UAT. Pada teori, UAT seharusnya melibatkan pengguna akhir (*end-user*) secara langsung untuk memberikan umpan balik. Pada pelaksanaan magang, proses validasi sistem lebih banyak dilakukan oleh tim pengembang dan pembimbing proyek, belum melibatkan *staff* dapur secara langsung. Hal ini mungkin terjadi karena keterbatasan akses atau proyek yang masih dalam tahap pengembangan. Akibatnya, umpan balik mengenai kemudahan penggunaan dan kesesuaian alur kerja dengan kondisi dapur yang sebenarnya belum sepenuhnya tergali. Ini menjadi sebuah pelajaran berharga bahwa validasi dari pengguna akhir adalah elemen yang tidak tergantikan untuk memastikan sebuah sistem benar-benar siap digunakan di lingkungan produksi.

4.2 Pembelajaran Magang

Selama kurang lebih enam bulan menjalani magang di PT. Vastu Cipta Persada, penulis memperoleh banyak pembelajaran berharga. Pembelajaran ini tidak hanya berasal dari pengalaman teknis dalam mengembangkan sistem, tetapi juga dari interaksi dan dinamika tim di lingkungan kerja profesional.

4.2.1 Pembelajaran Teknis

a. Penguasaan Framework Laravel dan Ekosistemnya

Sebelum magang, pengetahuan penulis tentang Laravel masih bersifat dasar. Selama proyek berlangsung, penulis mendapatkan pemahaman yang lebih dalam, mulai dari pembuatan *controller*, *model*, dan *migration*, hingga implementasi *middleware* untuk keamanan dan pembuatan REST API. Penulis juga belajar tentang struktur proyek Laravel yang menerapkan pola MVC, yang sangat membantu dalam menjaga kode tetap terstruktur dan mudah dipelihara.

b. Pendalaman JavaScript untuk Frontend Interaktif

Pengerjaan *Kitchen Dashboard* memberikan kesempatan bagi penulis untuk mengasah kemampuan JavaScript, khususnya dalam menangani operasi asinkron (*Promise, Async/Await*) untuk komunikasi API, manipulasi DOM secara dinamis, dan implementasi logika bisnis di sisi klien. Pembuatan kelas-kelas modular seperti `*KitchenApiService*` dan `*KitchenDashboard*` juga mengajarkan penulis tentang pentingnya arsitektur kode yang baik untuk aplikasi skala menengah.

c. Pemahaman Konsep Real-Time dan Optimasi

Penulis belajar secara langsung bagaimana membangun sistem dengan pembaruan data mendekati *real-time* menggunakan *polling*. Lebih penting lagi, penulis memahami bahwa kecepatan aplikasi bukanlah satu-satunya ukuran; efisiensi sumber daya juga sangat krusial. Penerapan *hash comparison*, *Page Visibility API*, dan *debounce* adalah bukti nyata bahwa seorang *developer* harus memikirkan dampak setiap baris kode terhadap performa aplikasi secara keseluruhan.

d. Praktik Penggunaan Tools Pendukung

Magang ini juga mengenalkan penulis pada berbagai *tools* profesional seperti Postman untuk pengujian API, TablePlus untuk manajemen basis data, Git untuk kontrol versi, dan platform kolaborasi seperti yang digunakan perusahaan. Pengalaman menggunakan *tools* ini secara langsung dalam proyek nyata sangat berbeda dengan sekadar mempelajarinya di bangku kuliah dan menjadi modal berharga untuk karir di masa depan.

4.2.2 Pembelajaran Non-Teknis

a. Pentingnya Komunikasi dan Kolaborasi

Ini adalah pelajaran paling penting yang penulis dapatkan. Pengembangan perangkat lunak adalah kerja tim. Penulis belajar untuk tidak sungkan bertanya ketika menemui kendala, berdiskusi secara aktif untuk menentukan solusi terbaik, dan memberikan laporan kemajuan secara teratur. Komunikasi yang baik dengan pembimbing proyek dan tim *backend* sangat krusial, terutama dalam menentukan spesifikasi *endpoint* dan format data API agar integrasi berjalan mulus.

b. Manajemen Waktu dan Pemenuhan Deadline

Bekerja di industri menuntut kedisiplinan waktu yang tinggi. Penulis belajar untuk memprioritaskan tugas, mengestimasi waktu pengerjaan, dan memastikan setiap *sprint* atau *milestone* dapat diselesaikan tepat waktu. Pengalaman ini mengajarkan penulis untuk lebih terstruktur dalam bekerja dan tidak menunda-nunda pekerjaan.

c. Profesionalisme dan Etos Kerja

Penulis belajar tentang etika dan profesionalisme di tempat kerja, seperti tanggung jawab terhadap tugas yang diberikan, menjaga kerahasiaan data proyek, dan berkontribusi positif bagi tim. Menyadari bahwa hasil kerja kita akan digunakan oleh orang lain membuat penulis lebih teliti dan berkomitmen untuk memberikan kualitas terbaik.

d. Adaptasi dengan Platform Baru

Meskipun proyek ini menggunakan teknologi yang relatif umum (Laravel, JavaScript), penulis tetap belajar untuk cepat beradaptasi dengan *codebase* yang sudah ada dan mengikuti standar penulisan kode yang telah ditetapkan oleh perusahaan. Kemampuan adaptasi ini adalah *skill* penting bagi seorang *developer* di dunia kerja yang dinamis.

Secara keseluruhan, pengalaman magang ini telah memberikan penulis bekal yang sangat berharga, tidak hanya dalam hal teknis pemrograman, tetapi juga dalam pengembangan diri sebagai seorang profesional di bidang teknologi informasi.

BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan hasil pelaksanaan magang dan pengembangan *Kitchen Dashboard* pada sistem *Point of Sale* untuk usaha kuliner, dapat ditarik kesimpulan sebagai berikut:

1. **Pengembangan *Kitchen Dashboard* Berhasil Memenuhi Kebutuhan Operasional Dapur**

   Sistem yang dikembangkan mampu menampilkan informasi pesanan secara *real-time* melalui mekanisme *polling API* dengan interval 3 detik. Hal ini memungkinkan *staff* dapur untuk memantau pesanan baru yang masuk dari kasir tanpa perlu melakukan penyegaran halaman secara manual. Informasi yang ditampilkan pada tiga panel utama (*Queue Orders*, *Active Orders*, dan *Served Orders*) membantu *staff* dapur dalam mengelola prioritas pengerjaan pesanan secara lebih terstruktur dan efisien.

2. **Integrasi Sistem Berjalan dengan Baik melalui REST API**

   Komunikasi antara *frontend* berbasis JavaScript dan *backend* Laravel berhasil diimplementasikan menggunakan REST API. Seluruh *endpoint* yang dirancang (seperti untuk mengambil daftar pesanan, mengambil pesanan dari antrian, membatalkan pesanan, dan menandai pesanan selesai) berfungsi dengan baik dan mampu mengembalikan data dalam format JSON yang sesuai. Hal ini membuktikan bahwa REST API efektif sebagai jembatan pertukaran data yang fleksibel dan terstruktur.

3. **Penerapan Teknik Optimasi Mampu Meningkatkan Efisiensi Sistem**

   Kombinasi teknik optimasi yang diterapkan terbukti ampuh dalam menjaga performa aplikasi. *Hash comparison* berhasil mengurangi proses render ulang halaman yang tidak diperlukan, sehingga menghemat penggunaan CPU browser. Implementasi *Page Visibility API* mampu menghentikan *polling* saat halaman tidak aktif, mengurangi beban server dan konsumsi *bandwidth*. Sementara itu, penggunaan *debounce* pada fitur pencarian meminimalkan eksekusi fungsi yang berlebihan, membuat proses pencarian menjadi lebih efisien dan responsif.

4. **Tujuan Pengembangan Tercapai**

   Berdasarkan hasil pengujian fungsionalitas, seluruh fitur utama *Kitchen Dashboard* yang direncanakan pada tahap analisis kebutuhan, seperti menampilkan daftar pesanan, mengelola status pesanan, menampilkan detail pesanan, dan menyediakan antarmuka

responsif, telah berhasil diimplementasikan dan berjalan sesuai dengan yang diharapkan. Dengan demikian, tujuan pengembangan untuk membantu *staff* dapur dalam memantau dan mengelola pesanan secara lebih efektif dan efisien dapat tercapai.

5.2 Saran

Meskipun pengembangan *Kitchen Dashboard* telah mencapai tujuan utamanya, terdapat beberapa hal yang dapat menjadi saran untuk pengembangan lebih lanjut dan perbaikan di masa mendatang.

1. **Bagi Perusahaan (PT. Vastu Cipta Persada)**
   - **Melibatkan Pengguna Akhir (*End-User*) pada Tahap Pengujian**

     Seperti yang telah direfleksikan pada Bab IV, pelibatan *staff* dapur secara langsung dalam proses *User Acceptance Testing* (UAT) sangat disarankan. Umpan balik langsung dari pengguna sangat krusial untuk memastikan antarmuka dan alur kerja sistem benar-benar sesuai dengan kebutuhan dan kenyamanan operasional di dapur. Hal ini akan meminimalkan risiko sistem yang tidak *user-friendly* ketika sudah digunakan di lingkungan produksi.
   - **Mempertimbangkan Penggantian Mekanisme *Polling* dengan *WebSocket* untuk *Real-Time* yang Lebih Baik**

     Meskipun *polling* sudah cukup efektif, untuk skala bisnis yang lebih besar atau kebutuhan *real-time* yang lebih instan, disarankan untuk mengeksplorasi teknologi *WebSocket*. *WebSocket* memungkinkan komunikasi dua arah secara persisten antara *client* dan *server*, sehingga data dapat dikirim secara langsung ketika terjadi perubahan tanpa perlu permintaan berkala dari *client*. Hal ini akan lebih efisien dan memberikan pengalaman *real-time* yang lebih mulus.

2. **Bagi Diri Penulis dan Pengembang Selanjutnya**
   - **Mendalami Konsep Arsitektur Sistem dan Desain Pola**

     Pengalaman magang ini menunjukkan bahwa pemahaman yang baik tentang arsitektur sistem (seperti MVC pada Laravel) dan pola desain (seperti struktur modular pada *frontend*) sangat membantu dalam mengelola kompleksitas proyek. Ke depannya, penting untuk terus mempelajari dan menerapkan prinsip-prinsip *clean code* dan *software design pattern* dalam setiap proyek yang dikerjakan.
   - **Mengembangkan Fitur Tambahan untuk Analitik dan Laporan**

Sistem yang ada dapat dikembangkan lebih lanjut dengan menambahkan fitur analitik dan laporan kinerja dapur, misalnya rata-rata waktu pengerjaan pesanan per menu, total pesanan per jam operasional, atau tingkat pembatalan pesanan. Fitur ini akan sangat bermanfaat bagi manajemen restoran untuk mengambil keputusan bisnis yang lebih baik berdasarkan data operasional yang akurat.

DAFTAR PUSTAKA

Ahyar, M., Wibisono, Y., Khusnuliawati, H., & Charolina, A. (2024). IMPLEMENTASI FUNGSI DEBOUNCE PADA APLIKASI DETEKSI CUACA BERBASIS WEB. *JIMSTEK: Jurnal Ilmiah Sains Dan Teknologi*, *6*(02), 87–92. https://doi.org/10.47942/JIMSTEK.V6I02.1837

Anggraeni, M. M., & Susetyo, Y. A. (2025). Implementation of the Laravel Framework in Tutoring Information System (Case Study: Glow Math Course). *Sistemasi*, *14*(3), 1406. https://doi.org/10.32520/stmsi.v14i3.5211

Ario Yudanto, B., Putra Laudri, S., Felina, I., Rizqi Amanusa Abinovan, M., Fadhil, M., Fathir Rakha Putra, R., … Menu Dan Pemesanan, M. (2025). Implementasi REST API Menggunakan Bahasa Go untuk Optimalisasi Manajemen Menu dan Pemesanan. *Jurnal Riset Dan Aplikasi Mahasiswa Informatika (JRAMI)*, *6*(02), 384–393. https://doi.org/10.30998/JRAMI.V6I02.13367

Evry Nazyli Ciptanto, N. (2024). *Journal of Digital Business and Technology Innovation (DBESTI) PENGEMBANGAN REST API UNTUK EKSTRAKSI INFORMASI DARI RESUME MENGGUNAKAN JAVA SPRING BOOT*. *1*(2), 51–60.

Fajrizal, Handayani, S., Taslim, Walhidayat, & Syahril. (2025). PELATIHAN PEMROGRAMAN JAVASCRIPT BAGI SISWA SMK INOVASI RIAU. *J-COSCIS : Journal of Computer Science Community Service*, *5*(1), 37–45. https://doi.org/10.31849/JCOSCIS.V5I1.22458

Mardiana, I., Wahyudin, & Junaeti, E. (2024). Pengembangan Learning Management System dengan Framework Laravel dan Tailwind CSS. *Multinetics*, *10*(1), 40–49. https://doi.org/10.32722/multinetics.v10i1.6678

Maulana, R., Sulistyanto, A., & Asih Septia Rini, N. (2021). Perancangan sistem informasi pengajuan dan pelaporan pembayaran tunjangan kinerja pada lembaga pemasyarakatan salemba berbasis web menggunakan Skema Rest API. *Jurnal Manajamen Informatika Jayakarta*, *1*(4), 283–294. https://doi.org/10.52362/JMIJAYAKARTA.V1I4.507

Nurjaman, I., Utomo, F. S., & Hermanto, N. (2024). Penerapan REST API Laravel sebagai Fondasi Back-end Aplikasi G-MOOC 4D. *Journal of Informatics and Interactive Technology*, *1*(1), 9–18. https://doi.org/10.63547/JIITE.V1I1.4

Pangestu, H. A., Kurniadi, D., & Septiana, Y. (2022). Aplikasi Pengelolaan Data Pegawai Berbasis REST API untuk Transfer Data Real Time dengan Framework Codeigniter.

Jurnal Algoritma, *19*(1), 313–322. https://doi.org/10.33364/ALGORITMA/V.19-1.1090

Pranantya, M. C. (2025). Pengembangan Mobile Learning Media Terintegrasi dengan Artificial Intelligence sebagai Virtual Mentor pada Materi Pemanasan Global untuk Meningkatkan Kemampuan Literasi Sains. *Unnes Physics Education Journal*, *14*(2), 249–258. https://doi.org/10.15294/upej.v14i2.33499

Putra, R. A., Kautsar, I. A., Hindarto, H., & Sumarno, S. (2023). Detection and Prevention of Insecure Direct Object References (IDOR) in Website-Based Applications: Deteksi dan Pencegahan Insecure Direct Object References (IDOR) Pada Aplikasi Berbasis Website. *Procedia of Engineering and Life Science*, *4*. https://doi.org/10.21070/PELS.V4I0.1435

Putu, I., Dharmaadi, A., Made, G., Sasmitha, A., Raya, J., & Unud, K. (2018). Perancangan Sistem Informasi Restoran Terintegrasi Berbasis Java Web Socket Online. *Jurnal Penelitian Pos Dan Informatika*, *8*(1), 51–62. https://doi.org/10.17933/JPPI.V8I1.135

Rahmawati, L., & Sumarsono, S. (2024). Desain Pengembangan Website dengan Arsitektur Model View Controller pada Framework Laravel. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *6*(4), 785–790. https://doi.org/10.47233/JTEKSIS.V6I4.1497

Rozzi Pratama, D., Budi, E., Kom, I. S., Robbani, R., & Md, A. (2023). Pengembangan REST API SiAbang (Sistem Administrasi Pembangunan) Menggunakan Java. *Jurnal Begawe Teknologi Informasi (JBegaTI)*, *4*(1). https://doi.org/10.29303/JBEGATI.V4I1.978

Septiani, N. I., Sediyono, A., & Rochman, A. (2022). PERANCANGAN WEB DEFACEMENT MONITORING DENGAN MENGGUNAKAN METODE KOMPARASI NILAI HASH1. *JIKO (Jurnal Informatika Dan Komputer)*, *5*(2), 150–155. https://doi.org/10.33387/JIKO.V5I2.4852

Sistem Pemrograman Model View Controller (MVC) - Informatika Universitas Ciputra. (n.d.). Retrieved June 24, 2026, from https://informatika.ciputra.ac.id/2016/12/sistem-pemrograman-model-view-controller-mvc/

Telkom University Open Library. (n.d.). Retrieved August 18, 2026, from https://openlibrary.telkomuniversity.ac.id/open/index.php/download/flippingbook/eyJkb3dubG9hZCI6IjEiLCJkd24iOnsia25vd2xlZGdlX2l0ZW1faWQiOiI5NTUwMCIsIm1lbWJlcl9pZCI6IiIsIm5hbWUiOiIxMTMwNzAzMDZfcmVzdW1lLnBkZiJ9LCJyZWFkb25seSI6IjEiLCJyZWFkIjp7Imtub3dsZWRnZV9pdGVtX2lkIjoiOTU1MDAiLCJtZW1iZXJfaWQiOiIiLCJuYW1lIjoiMTEzMDcwMzA2X3Jlc3VtZS5wZGYifSwibmFtZSI6IjExMzA3MDMwNl9yZXN1bWUucGRmIiwibGluayI6IlwvZGF0YVwvYmF0aWtcL3N5bWZvbnlfcHJvamVjdHNcL2Jvb2tcLzExMzA3MDMwNlwvMTEzMDcwMzA2X3Jlc3

VtZS5wZGYifQ==

Delgado, E. (2010, February 24). Menggunakan Page Visibility API. web.dev. https://web.dev/articles/pagevisibility-intro?hl=id

LAMPIRAN