IMPLEMENTASI BISNIS APLIKASI MECHA: LAYANAN PERBAIKAN KENDARAAN BERBASIS *ANDROID* MENGGUNAKAN *MVP* SEBAGAI *DESIGN PATTERN*



Disusun Oleh:

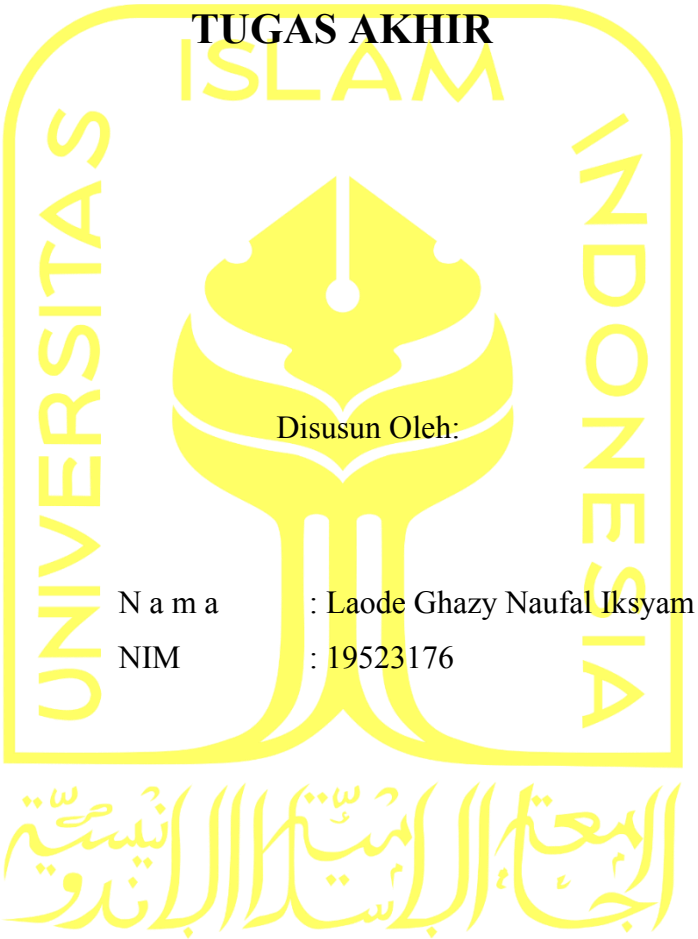
N a m a : Laode Ghazy Naufal Iksyam
NIM : 19523176

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2023

HALAMAN PENGESAHAN DOSEN PEMBIMBING

IMPLEMENTASI BISNIS APLIKASI MECHA: LAYANAN PERBAIKAN KENDARAAN BERBASIS *ANDROID* MENGGUNAKAN *MVP* SEBAGAI *DESIGN PATTERN*

TUGAS AKHIR



Disusun Oleh:

N a m a : Laode Ghazy Naufal Iksyam

NIM : 19523176

Yogyakarta, 14 Oktober 2023

Pembimbing,

(Irving Vitra Paputungan, S.T., M.Sc., Ph.D.)

HALAMAN PENGESAHAN DOSEN PENGUJI

IMPLEMENTASI BISNIS APLIKASI MECHA: LAYANAN PERBAIKAN KENDARAAN BERBASIS *ANDROID* MENGGUNAKAN *MVP* SEBAGAI *DESIGN PATTERN*

TUGAS AKHIR

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, 1 Oktober 2023

Tim Penguji

Irving Vitra Paputungan, S. T., M.Sc., Ph.D.

Anggota 1

Moh. Idris, S.Kom., M.Kom.

Anggota 2

Sri Mulyati, S.Kom., M.Kom.

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

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

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Laode Ghazy Naufal Iksyam

NIM : 19523176

Tugas akhir dengan judul:

IMPLEMENTASI BISNIS APLIKASI MECHA: LAYANAN PERBAIKAN KENDARAAN BERBASIS *ANDROID* MENGGUNAKAN *MVP* SEBAGAI *DESIGN PATTERN*

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, 19 Oktober 2023



(Laode Ghazy Naufal Iksyam)

HALAMAN PERSEMBAHAN

Alhamdulillah puji dan syukur ke hadirat Allah SWT atas segala rahmat, hidayah, inayah serta karunia-Nya sehingga saya dapat menyelesaikan laporan tugas akhir ini dengan baik. Tugas akhir ini saya persembahkan kepada:

Terima kasih kepada Allah SWT yang telah memberikan saya kesempatan dan kesehatan untuk menyelesaikan Skripsi ini. Semoga Skripsi ini dapat bermanfaat bagi saya pribadi dan juga bagi semua pihak yang membaca dan menggunakannya.

Keluarga saya Bapak Laode Muhammad Syamrin dan Ibu Ike Sastrawati yang telah membesarkan saya dan mendidik saya dengan penuh kasih sayang hingga saat ini.

Ibu Aridhanyati Arifin, S.T., M.Cs. Terima kasih telah membantu saya selama masa perkuliahan dan memberikan saya arahan selama masa perkuliahan saya berlangsung.

Bapak Irving Vitra Paputungan S.T., M.SC., PH.D., Terima kasih telah membimbing, mengajar dan memberikan saya motivasi dalam pengerjaan laporan tugas akhir ini.

Kevin Raihan Saleh dan Rio Yuda Sakti, anggota tim saya, telah memberikan kontribusi yang sangat berharga serta bekerja sama dengan saya dalam mengembangkan dan menyelesaikan rancangan bisnis ini.

Seluruh keluarga besar SANTOSO Abiel, Rio, Rafi, Ghilman, Ery, Fardan, Ilham, Imam, Farras, Lutpi, Raihan, Fiqi, Okto, Kevin, Bakar yang telah menemani saya selama pengerjaan tugas akhir dan memberikan saya support tanpa henti.

Rekan rekan infinity yang telah memberikan motivasi serta dukungan selama masa perkuliahan.

Kepada para pihak yang terlibat dalam pengerjaan tugas akhir yang telah memberikan waktu dan tenaga sehingga pengerjaan tugas akhir dapat terselesaikan.

Annisa Fitria Huda yang telah menemani saya dari mulai SMA hingga saya menyelesaikan tugas akhir ini dan membantu saya secara emosional.

Bakar dan Farras yang telah sabar menghadapi keluh kesah saya selama pengerjaan tugas akhir saya baik itu yang berhubungan dengan tugas akhir atau tidak.

Dan orang orang yang tidak bisa saya sebutkan satu persatu dikarenakan terlalu banyak yang telah membantu saya selama masa perkuliahan ini.

HALAMAN MOTO

“Mau semengerikan apa pun dunia ini, mau sekejam apa pun dunia ini. Bertarunglah!
(Eren Yeager)

“Aku siap, aku siap!”

(Spongebob Squarepants)

"Jangan menyerah, Allah selalu ada untuk membantu."
(Laode Ghazy Naufal Iksyam)

KATA PENGANTAR

Assalamualaikum Wr. Wb.

Salam sejahtera dan atas izin rahmat Allah SWT, Dengan penuh rasa syukur, penulis mengucapkan puji syukur kehadirat Allah SWT atas segala rahmat, hidayah, dan karunia-Nya sehingga penulis dapat menyelesaikan perjalanan akademik dalam menyusun skripsi ini. Dengan penuh dedikasi dan semangat, skripsi ini berjudul "Implementasi Bisnis Aplikasi MECHA: Layanan Perbaikan Kendaraan Berbasis Android Menggunakan MVP sebagai Design Pattern" telah disusun sebagai salah satu wujud komitmen penulis dalam memahami dan mengaplikasikan pola desain Model-View-Presenter (MVP) dalam pengembangan aplikasi berbasis Android.

Penulisan skripsi ini tidak terlepas dari bimbingan, dukungan, dan dorongan dari berbagai pihak yang turut serta memberikan arahan dan ilmu pengetahuan selama proses penulisan. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada semua pihak yang telah memberikan kontribusi, baik secara langsung maupun tidak langsung dalam penyusunan skripsi ini. Semoga hasil dari perintisan bisnis ini dapat memberikan manfaat dan kontribusi yang signifikan dalam pengembangan teknologi informasi, khususnya dalam aplikasi layanan perbaikan kendaraan.

Melalui perintisan bisnis ini, penulis berharap dapat memberikan kontribusi terhadap pengembangan teknologi informasi, khususnya dalam penggunaan pola desain MVP dalam aplikasi berbasis Android. Semoga skripsi ini dapat menjadi sumber inspirasi dan pengetahuan bagi pembaca yang tertarik dalam mengembangkan aplikasi berbasis Android dengan pendekatan yang lebih efektif dan efisien. Akhir kata, penulis mengucapkan terima kasih sebesar-besarnya kepada semua pihak yang telah membantu dan mendukung terwujudnya skripsi ini. Semoga Allah SWT senantiasa memberikan rahmat dan hidayah-Nya dalam setiap langkah kehidupan kita. Amin.

Yogyakarta, 1 Oktober 2023



(Laode Ghazy Naufal Iksyam)

SARI

Penilitian ini mengangkat isu penting dalam industri bengkel, di mana masih banyak bengkel yang mengandalkan metode konvensional. Hal ini mengakibatkan menurunnya jumlah pelanggan dan kesulitan bagi pengguna kendaraan dalam mencari bengkel yang dapat diandalkan dan terdekat. Untuk mengatasi permasalahan ini, dirancanglah aplikasi MECHA. Aplikasi ini bertujuan untuk memperkuat daya tarik bisnis pemilik bengkel dengan meningkatkan jumlah pelanggan, sekaligus memudahkan pengguna kendaraan dalam mencari bengkel yang dapat dipercaya dan berlokasi terdekat. Pengembangan aplikasi MECHA didasari oleh prinsip-prinsip desain yang kokoh, dengan menerapkan pola desain Model View Presenter (MVP) sebagai tulang punggung utama. Pendekatan *Agile* digunakan dalam proses pengembangan untuk memastikan kelancaran dan responsivitas tim dalam mengatasi tantangan-tantangan yang muncul selama proses pengembangan. Metode pengujian *System Usability Testing* diterapkan untuk memastikan antarmuka pengguna aplikasi ini intuitif dan mudah digunakan. Hasil dari pengembangan ini adalah sebuah aplikasi mobile yang memadukan kebutuhan baik pemilik bengkel maupun pengguna kendaraan. MECHA memberikan solusi terbaik dengan menyediakan platform yang memungkinkan pemilik bengkel untuk menjangkau lebih banyak pelanggan, sementara pengguna kendaraan dapat dengan mudah menemukan bengkel terdekat dan tepercaya. Dengan pendekatan yang terstruktur dan metode pengujian yang teliti, MECHA memenuhi kebutuhan bisnis digital yang berkembang pesat dalam industri layanan perbaikan kendaraan.

Kata kunci: Bengkel, Aplikasi MECHA, *MVP*, *Agile*, *System Usability Testing*, Aplikasi Mobile, Layanan Perbaikan Kendaraan, Pengguna Kendaraan, Bisnis Digital

GLOSARIUM

Design Pattern	Merujuk pada solusi umum yang dapat digunakan untuk menangani masalah desain perangkat lunak tertentu. Mereka adalah panduan atau template untuk membantu pengembang dalam membuat struktur dan organisasi yang baik dalam kode mereka.
Sytem Usability Testing	System Usability Testing (SUT) adalah metode evaluasi dalam pengembangan perangkat lunak yang bertujuan untuk menilai sejauh mana sebuah sistem atau aplikasi dapat digunakan dengan efektif, efisien, dan memuaskan oleh pengguna akhir.
Model View Presenter	Model-View-Presenter (MVP) adalah sebuah pola desain yang memisahkan elemen utama dalam sebuah aplikasi. Model bertanggung jawab atas data dan logika aplikasi, View menangani tampilan antarmuka pengguna, sementara Presenter berperan sebagai penghubung antara Model dan View. Presenter mengatur alur logika, memproses data dari Model, dan menyampaikan informasi ke View. Dengan memisahkan elemen-elemen ini, MVP mempermudah pengujian, meningkatkan skalabilitas, dan memungkinkan pengembangan lebih terstruktur dan mudah dipelihara.
Minimum Viable Product	Minimum Viable Product (MVP) adalah versi awal dari sebuah produk atau aplikasi yang memiliki seperangkat fitur dan fungsionalitas yang cukup untuk memenuhi kebutuhan dasar pengguna.

Agile	Agile adalah suatu pendekatan dalam pengelolaan dan pengembangan proyek perangkat lunak yang menekankan kolaborasi, adaptasi terhadap perubahan, dan pengiriman produk secara bertahap

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 xiv
DAFTAR GAMBAR xv
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 3
1.3 Batasan Masalah 3
1.4 Tujuan 4
1.5 Manfaat 4
1.6 Sistematika penulisan 4
BAB II LANDASAN TEORI 6
2.1 Model View View Model 6
2.2 Model View Controller 6
2.3 Model View Presenter 6
2.4 Perbandingan *Design Pattern* 7
2.5 Agile 8
2.6 Waterfall 9
2.7 Lean 9
2.8 Perbandingan Pendekatan Pengembangan 10
BAB III METODOLOGI 12
3.1 Rancangan Pengembangan 12
3.1.1 Pendekatan Pengembangan 12

3.1.2 Pemilihan Teknologi 12
3.1.3 Alur Kerja Aplikasi 12
3.1.4 Struktur Aplikasi 13
3.2 Pendekatan Pembangan Aplikasi 14
3.2.1 Tahap Perencanaan 14
3.2.2 Tahap Development 14
3.2.3 Tahap Implementasi 14
3.2.4 Tahap Testing 14
3.3 Model-View-Presenter (*MVP*) sebagai *Design Pattern* 15
3.3.1 Konsep Dasar *MVP* 15
3.3.2 Keuntungan Penggunaan *MVP* 15
3.3.3 Implementasi *MVP* dalam Aplikasi MECHA 16
3.4 Metode Pengumpulan Data 16
3.4.1 Studi Literatur 16
3.4.2 Wawancara 16
3.4.3 Survei 16
3.5 *Prototype* Pada Mecha 17
3.6 *Minimum Viable Product* Pada Mecha 17
3.7 *System Usability Scale* Pada Mecha 18
BAB IV HASIL DAN PEMBAHASAN 21
4.1 Arsitektur Aplikasi 21
4.2 Implementasi Model 21
4.2.1 Register 21
4.2.2 Login 22
4.2.3 Pesan Layanan 22
4.2.4 Pembayaran 23
4.2.5 Riwayat Pemesanan 24
4.2.6 Detail Pesanan 24
4.3 Implementasi View 25
4.3.1 Register User 25
4.3.2 Register Mitra 27
4.3.3 Login 28
4.3.4 Pesan Layanan 30
4.3.5 Pembayaran 36

4.3.6 Riwayat Pesanan....42
4.4 Implementasi Presenter....43
4.4.1 Register....44
4.4.2 Login....46
4.4.3 Pesan Layanan....47
4.4.4 Pembayaran....50
4.4.5 Riwayat Pesanan....53
4.5 Pengujian Menggunakan Dua Mitra....54
4.6 Arsitektur Database....54
4.6.1 Struktur Basis Data....54
4.6.2 Relasi Antar Tabel....56
4.6.3 Implementasi API Dalam MECHA....57
4.7 *Adonis* dan *Ngrok* pada aplikasi MECHA....64
4.8 Implementasi Backend....64
4.8.1 Implementasi Backend Register....65
4.8.2 Implementasi Backend Login....66
4.8.3 Implementasi Pesan Layanan....67
4.8.4 Implementasi Ambil Pesanan....68
4.8.5 Implementasi History....69
4.9 Testing....70
4.9.1 Pengujian *System Usability Scale* (SUS)....70
4.9.2 Hasil *Testing* Mitra Bengkel(karena sample sedikit lebih baik kualit)....72
BAB V KESIMPULAN DAN SARAN....75
5.1 Kesimpulan....75
5.2 Saran....75
DAFTAR PUSTAKA....76
LAMPIRAN....79

DAFTAR TABEL

Tabel 2.1 Perbandingan Desain Arsitektur 7
Tabel 2.2 Tabel Perbandingan Pendekatan 10
Tabel 3.1 Tech Stack Mecha 17
Tabel 3.2 Tabel Pertanyaan *System Usability Testing* 19
Tabel 3.3 Peringkat Dalam *System Usability Scale* 20
Tabel 4.1 Tabel User 55
Tabel 4.2 Tabel Mekanik 55
Tabel 4.3 Tabel Order 55
Tabel 4.4 Tabel History 56
Tabel 4.5 Daftar Pernyataan 70
Tabel 4.6 Hasil Pengujian 70
Tabel 4.7 Hasil Usability Testing 71
Tabel 4.8 Hasil Pengujian Montir 72

DAFTAR GAMBAR

Gambar 4.1 Model Register 22
Gambar 4.2 Model Login 22
Gambar 4.3 Model Pesan Layanan 23
Gambar 4.4 Price Model 24
Gambar 4.5 Model Riwayat Pesanan 24
Gambar 4.6 Model Detail Pemesanan 25
Gambar 4.7 UI Daftar 26
Gambar 4.8 Kotlin Register 27
Gambar 4. 9 Register Mitra 27
Gambar 4.10 Kotlin Register Mitra 28
Gambar 4.11 UI Login 29
Gambar 4.12 Kotlin Login 30
Gambar 4.13 UI Map 31
Gambar 4.14 Kotlin Map 32
Gambar 4.15 UI Konfirmasi Order 33
Gambar 4.16 Kotlin Konfirmasi Order 34
Gambar 4.17 UI Ambil Pesan Montir 35
Gambar 4.18 Kotlin Ambil Pesan Montir 35
Gambar 4.19 UI Detail Pembayaran 37
Gambar 4.20 Kotlin Detail Pembayaran 38
Gambar 4.21 UI Konfirmasi Pembayaran Montir 39
Gambar 4.22 Kotlin Konfirmasi Pembayaran Montir 40
Gambar 4.23 UI Bayar Tagihan Montir 41
Gambar 4.24 Kotlin Bayar Tagihan 41
Gambar 4.25 UI Riwayat Pesanan 43
Gambar 4.26 Kotlin Riwayat Pesanan 43
Gambar 4.27 Presenter Register 45
Gambar 4.28 Presenter Login 47
Gambar 4.29 Presenter Pesan Layanan 49
Gambar 4.30 Presenter Ambil Pesan 50
Gambar 4.31 Presenter Pembayaran 51
Gambar 4.32 Presenter Pembayaran Montir 53

Gambar 4.33 Presenter Riwayat Pesanan 54
Gambar 4. 34 Function Delete Ambil Pesan 54
Gambar 4.35 Relasi Antar Tabel 56
Gambar 4.36 Network API 60
Gambar 4.37 Order API 62
Gambar 4.38 User API 64
Gambar 4.39 Backend Register 66
Gambar 4.40 Backend Login 67
Gambar 4. 41 Backend Pesan Layanan 68
Gambar 4. 42 Backend Ambil Pesanan 69
Gambar 4. 43 Backend History 70

BAB I
PENDAHULUAN

1.1 Latar Belakang

Berbagai aspek dunia dipengaruhi oleh kemajuan ilmu pengetahuan. Salah satu contohnya adalah perkembangan teknologi digital dan virtual yang terus berkembang seiring dengan kemajuan ilmu pengetahuan. Ini juga terkait dengan persaingan perusahaan untuk mengembangkan teknologi terbaru(Riza Aulia Putra, 2018). Penerapan ilmu pengetahuan dan pengetahuan lainnya merupakan bagian dari perkembangan teknologi. Karena teknologi dapat digunakan dalam banyak bidang kehidupan, itu pasti sangat bermanfaat (Dwi Martha dkk., 2018). *Android* merupakan sistem operasi yang telah disesuaikan dari varian *Linux* (Selvi Dwi Hartiyani dkk., 2023)Dalam hal ini, penggunaan *smartphone* pada pekerjaan sehari-sehari tidak terlepas dari penggunaan *internet* yang dibutuhkan untuk menunjang penggunaan *smartphone* itu sendiri. Dampak yang dapat kita rasakan dari penggunaan *smartphone* dengan berbagai aplikasi maupun *platform* didalamnya dapat kita rasakan dari berbagai bidang seperti Pendidikan, Kesehatan, hiburan hingga bidang bisnis. Dalam era digital yang semakin berkembang pesat ini, teknologi informasi (TI) telah menjadi komponen penting dalam perkembangan bisnis kontemporer(Anggraeni & Elan Maulani, 2023). Salah satu keunggulan perangkat seluler adalah kemampuannya untuk dibawa ke berbagai tempat tanpa perlu menghubungkannya ke jaringan telepon dengan menggunakan kabel (Hapsari et al., 2022).

Dalam bidang bisnis terdapat dua bagian yaitu layanan dan jasa yang mana permasalahan kali ini berfokus pada layanan. Bisnis layanan sendiri bertujuan untuk memberikan kenyamanan kepada masyarakat yang menggunakan layanan tersebut. Banyaknya jasa layanan memberikan kemudahan kepada masyarakat dalam melakukan kegiatan sehari hari salah satunya adalah jasa layanan perbaikan kendaraan. Salah satu fasilitas bisnis yang menyediakan jasa perbaikan purna jual kepada konsumen adalah bengkel (Ngurah dkk., 2017). Hal ini menyebabkan bengkel-bengkel menjadi lebih ramai, dengan pengendara harus mengantri untuk memperbaiki kendaraannya. Ketika pengendara mengalami masalah dengan kendaraannya di tengah perjalanan, seringkali sulit bagi mereka untuk menemukan bengkel terdekat yang dapat memperbaiki kendaraan mereka(Zaki &

Putra, 2018). Banyaknya lokasi bengkel terkadang masih membuat bingung para pengguna layanan bengkel dalam menemukan lokasi bengkel terdekat atau bengkel yang akan dicari karena pada umumnya bengkel hanya mencantumkan nama bengkel dan alamatnya saja terutama bagi bengkel resmi. Situasi ini dapat menciptakan kendala bagi pengguna layanan bengkel, terutama bagi mereka yang tidak familiar dengan nama dan alamat bengkel yang tertera dalam buku panduan(Ferico dkk., 2019).Banyaknya mahasiswa yogyakarta mengalami masalah mogok pada saat melakukan perjalanan menggunakan kendaraan roda dua, maka dari itu dibutuhkan aplikasi untuk membantu mahasiswa yogyakarta yang mengalami masalah kendaraan bermotor khususnya roda dua baik itu kempes ban maupun mogok kendaraan. Namun jika mengingat potensi dari penggunaan *smartphone* dengan *internet* yang menunjang seharusnya mampu mengatasi permasalahan tersebut.

Berdasarkan permasalahan diatas maka terbentuklah aplikasi MECHA, Empat jenis perawatan kendaraan yang biasanya dilakukan oleh pemilik motor adalah penggantian oli, tune up, kampas rem, dan penggantian suku cadang lainnya(Fitrahriansyah dkk., 2022). merupakan sebuah aplikasi penyedia jasa perbaikan kendaraan yang menggunakan *platform* mobile dengan menawarkan layanan untuk pengguna kendaraan roda dua berupa perbaikan mogok dijalan, tambal ban, ganti ban dan perbaikan sparepart motor. MECHA sendiri ada sebagai penghubung antara pengguna kendaraan bermotor dan pemilik bengkel yang mana pengguna kendaraan roda dua akan dimudahkan untuk menghubungi pemilik bengkel dan sebaliknya pemilik bengkel dimudahkan untuk mencari pelanggan yang memerlukan layanan nya.

Pengembangan aplikasi MECHA, pengembangan ini menggunakan metodologi *design pattern* . *Design pattern*s adalah unsur-unsur rancangan yang seringkali muncul pada berbagai sistem yang berbeda. Setiap kemunculan ini menguji pattern tersebut di berbagai situasi. Ketika suatu unsur rancangan semakin diuji, hal ini menandakan bahwa unsur tersebut semakin matang. Oleh karena itu, beberapa dari mereka dapat dianggap sebagai praktik terbaik dalam perancangan sistem(Kusnawi, 2010) .Pada desain pattern sendiri terdapat 3 opsi yaitu *MVC*, *MVVM*, dan *MVP* yang dimana memiliki tujuan yang sama yaitu memisahkan tanggung jawab antara visualisasi, pemrosesan, dan juga manajemen data untuk aplikasi UI. Sistem *MVP* (Model-View-Presenter) dirancang dengan prinsip bahwa antarmuka kode antara class view dan presenter harus terpisah. Hal ini memungkinkan

presenter untuk beroperasi dan menjalani proses pengujian tanpa mempengaruhi pengolahan kode pada view(Sahrial dkk., 2022).*MVC* sendiri adalah sebuah pola arsitektur dalam membuat sebuah aplikasi dengan cara memisahkan kode menjadi tiga bagian yaitu Model, View, Controller. *MVVM* sendiri adalah arsitektur pembuatan aplikasi berbasis GUI yang berfokus pada pemisahan antara kode untuk logika bisnis dan tampilan aplikasi. *MVP* adalah turunan dari *MVC* yang membantu menutupi kekurangan yang dimiliki oleh *MVC*. Menurut perintisan bisnis ini, status penggunaan energi dan memori *smartphone* pada tahap pengujian adalah heavy pada aplikasi tanpa pola perancangan dan medium pada aplikasi dengan pola perancangan. Memori yang digunakan pada aplikasi tanpa pola perancangan sebesar 92 MB, sedangkan pola perancangan untuk *MVC* sebesar 59,7 MB, *MVP* sebesar 59,7 MB, dan *MVVM* sebesar 73,2 MB(Rizki dkk., 2020). Setelah memanfaatkan keunggulan dan mempertimbangkan kekurangan yang kemudian akan diterapkan pada aplikasi MECHA akhirnya terpilih salah satu metodologi *design pattern* yang akan digunakan yaitu *MVP* (Model View Presenter).Penggunaan metodologi *MVP* (Model View Presenter) pada pengembangan MECHA dikarenakan komponen view dan presenter bisa di reuse, kode lebih mudah dipahami dan dikelola, *MVP* lebih mudah dipahami dan diimplementasikan namun *MVP* sendiri juga memiliki kekurangan yang mana ukuran kode akan lebih besar dari metode lainya. Metodologi *design pattern MVP* memiliki tahapan yaitu Antarmuka, Lapisan Muka, Lapisan Presenter. Tujuan dari penggunaan metodologi *design pattern MVP* ialah membuat baris kode terlihat terorganisir dengan lebih baik.

1.2 Rumusan Masalah

Rumusan masalah dalam perintisan bisnis ini adalah bagaimana merencanakan, merancang, dan mengembangkan sebuah aplikasi layanan bengkel online yang berbasis Android. Dalam konteks ini, aspek-aspek kunci yang akan diperhatikan meliputi antarmuka pengguna yang intuitif, fungsionalitas pengelolaan pesanan dan riwayat layanan, integrasi sistem pembayaran. Selain itu, juga akan dipertimbangkan faktor-faktor teknis dan infrastruktur yang memungkinkan aplikasi berjalan dengan lancar dan dapat diakses secara luas oleh pengguna.

1.3 Batasan Masalah

Adapun Batasan masalah yang akan dibahas yaitu sebagai berikut :

a. Aplikasi ini berfokus pada layanan untuk konsumen dalam mencari jasa *service* motor.

b. Aplikasi ini memiliki batasan pada platform Android, artinya dirancang khusus untuk perangkat yang menggunakan sistem operasi Android. Hal ini mengimplikasikan bahwa aplikasi tidak dapat diakses melalui platform lain seperti iOS atau sistem operasi lainnya.
c. Pembangunan aplikasi hanya sampai tahap pengujian.

1.4 Tujuan

Tujuan utama dari perintisan bisnis ini adalah untuk mengembangkan sebuah aplikasi layanan perbaikan kendaraan berbasis *Android* yang efisien dan mudah digunakan. Aplikasi ini bertujuan untuk menyediakan platform yang memungkinkan pengguna untuk dengan cepat dan tepat melakukan pemesanan layanan perbaikan kendaraan mereka. Dengan memanfaatkan teknologi *Android*, diharapkan aplikasi ini dapat memberikan pengalaman pengguna yang lebih baik dan mendukung pertumbuhan bisnis dalam industri jasa *service* motor.

1.5 Manfaat

Pengembangan ini diharapkan dapat memberikan manfaat sebagai berikut :

a. Manfaat untuk masyarakat
    1. Membantu masyarakat dalam mencari jasa *service* motor.
    2. Menghemat waktu masyarakat dalam mencari jasa *service* motor.
b. Manfaat untuk penyedia jasa.
    1. Mempermudah bengkel untuk mendapatkan pelanggan.
    2. Sebagai sarana promosi bengkel.
c. Bagi penulis
    1. Menambahkan pengetahuan dan wawasan mengenai membangun aplikasi berbasis *android*.

1.6 Sistematika penulisan

Dalam penulisan ini dilakukan secara sistematis untuk mengetahui isi dari uraian-uraian didalamnya. Bab-bab tersebut diantaranya:

Bab I : Pendahuluan, merupakan bab berisi latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika penulisan.

Bab II : Tinjauan Pustaka, meliputi berisi telaah perintisan bisnis dan landasan teori yang berkaitan dengan perintisan bisnis yang dilakukan.

Bab III : Metode Perintisan Bisnis, di dalam bab ini diuraikan tentang pendekatan teori dan analisis yang dibutuhkan untuk mendapatkan solusi dalam perancangan aplikasi.

Bab IV : Hasil dan Pembahasan, di dalam bab ini berisi hasil perintisan bisnis yang diimplementasi dan menampilkan hasil pengujian yang telah dibuat.

Bab V : Penutup, di dalam bab ini berisi kesimpulan hasil analisis dan saran.

BAB II
LANDASAN TEORI

2.1 Model View View Model

MVVM adalah singkatan dari Model-View-ViewModel, sebuah pola arsitektur perangkat lunak yang memisahkan Model (data dan logika bisnis), View (antarmuka pengguna), dan ViewModel (penghubung antara Model dan View) untuk mempermudah pengembangan aplikasi. Ini membantu mengorganisir kode, meningkatkan pengujian, dan memungkinkan pengembangan yang lebih efisien. MVVM sering digunakan dalam pengembangan aplikasi berbasis platform seperti Android, iOS, dan aplikasi desktop. Pola arsitektur MVVM memiliki manfaat signifikan dalam pengembangan aplikasi dengan memisahkan antara tampilan (View), logika bisnis (ViewModel), dan model data (Model). Pendekatan ini mempermudah pengembangan dan pemeliharaan aplikasi karena setiap komponen memiliki tanggung jawab yang terdefinisi dengan jelas dan terisolasi. Lebih lanjut, penerapan pola MVVM juga mampu meningkatkan pengalaman pengguna, meningkatkan kinerja aplikasi, serta memperbaiki efisiensi dalam proses pengembangan (Riyadhi dkk., 2023).

2.2 Model View Controller

Model-View-Controller (MVC) adalah pola arsitektur perangkat lunak yang memisahkan Model (data dan logika bisnis), View (antarmuka pengguna), dan Controller (pengatur interaksi). Dengan menerapkan MVC, aplikasi dapat diorganisir dengan lebih terstruktur, memudahkan pengujian, dan meningkatkan efisiensi pengembangan. MVC mampu mengurangi kompleksitas pengembangan aplikasi, sehingga dalam penelitian ini dibuat sebuah aplikasi identifikasi jam puncak arus lalu lintas pada simpang lima dengan MVC. Dengan demikian, MVC dijadikan pola dalam pengembangan aplikasi untuk mengatasi permasalahan kerumitan dalam pengolahan data jam puncak pada simpang lima jalan sebagai sumber konflik lalu lintas(Alkaff et al., 2021)002E

2.3 Model View Presenter

Model-View-Presenter (*MVP*) merupakan pola arsitektur perangkat lunak yang memisahkan komponen dalam aplikasi menjadi *Model* (pengelola data), *View* (tampilan pengguna), dan *Presenter* (pengontrol interaksi). *MVP* membantu memisahkan logika bisnis

dari antarmuka pengguna, memungkinkan pengembangan yang lebih terstruktur dan mudah diuji. *Model* mengatur data dan logika bisnis, *View* menampilkan informasi kepada pengguna, dan *Presenter* bertindak sebagai perantara yang mengambil data dari *Model*, memprosesnya, dan menyajikannya di *View*. Pendekatan ini mempermudah pemeliharaan, pengujian, dan skalabilitas aplikasi. *Model View Presenter* (*MVP*) adalah pola desain yang pertama kali muncul di IBM dan lebih terlihat di Taligent pada tahun 1990-an. Arsitektur ini, seperti pola desain lainnya, memiliki tujuan untuk memisahkan pengembangan secara efektif, memungkinkan beberapa pengembang bekerja dan menguji bersamaan. Dalam konteks pengembangan *Android*, terdapat masalah dengan fakta bahwa aktivitas *Android* terikat erat dengan mekanisme akses antarmuka dan data. Untuk mengatasi masalah ini dan membuat aplikasi mudah dikembangkan dan dipelihara, diperlukan definisi lapisan yang terpisah dengan baik. Oleh karena itu, dengan menerapkan arsitektur *MVP*, tugas latar belakang dipisahkan dari aktivitas/pandangan/fragmen untuk menghindari ketergantungan dengan sebagian besar kejadian terkait siklus hidup. Selain itu, tugas kompleks dipilah menjadi tugas yang lebih sederhana dan mudah dipecahkan, dan pandangan (view) dijadikan independen dari sumber data. *MVP* juga memfasilitasi pengujian unit otomatis. Kombinasi dari fitur-fitur tersebut membuat arsitektur *MVP* menjadi lebih nyaman bagi pengembang dengan keterampilan pemrograman *terbatas* atau minimal yang ingin mengembangkan aplikasi mobile (Yunus dkk., 2018)

2.4 Perbandingan *Design Pattern*

Dari *design pattern* yang telah dibahas sebelumnya, terbentuklah suatu tabel perbandingan yang umumnya digunakan dalam pengembangan aplikasi. Tabel 2.1 dibentuk untuk memperlihatkan perbandingan antara metode-metode dalam pengembangan aplikasi.

Tabel 2.1 Perbandingan Desain Arsitektur

No	Metode	Penulis dan Tahun Terbit	Maksud Penelitian	Kelebihan	Kekurangan
1	*MVVM*	Riyadhi IryaMuhammad, Intan Purnamasari, Kamal Prihandani (2023)	Pengembangan Aplikasi Pengaduan Masyarakat Berbasis Android dengan menggunakan MVVM	• Pisahan yang kuat antara tampilan dan logika bisnis • Binding data yang kuat • Memisahkan tampilan dan desain ui	• Memerlukan kode tambahan • Memerlukan pemahaman yang mendalam • Kompleksitas tambahan

2	*MVC*	Muhammad Alkaff, Iphan Fitrian Radam, Winarto Chandra (2021)	Pengembangan Aplikasi Identifikasi Jam Puncak Arus Lalu Lintas pada Simpang Lima menggunakan MVC	• Pisahan yang jelas antara logika aplikasi • Mendukung pengembangan parallel • Mendukung berbagai platform	• Pengelolaan kode yang rumit • Mungkin memiliki masalah dengan siklus hidup objek • Ketergantungan antar komponen
3	*MVP*	Ibrahim Yunus, Muhammad Fiqhan, Adam Hendra Brata, Agi Putra Kharisma (2018)	Pengembangan Aplikasi Pengingat Salat Dengan Konsep Context-Aware Menggunakan MVP Pada Platform Android	• Pisahan yang jelas antara logika bisnis dan UI • Fleksibilitas dalam memodifikasi tampilan • Mendukung Pengembangan Paralel	• Membutuhkan Kode Tambahan • Memerlukan Pemahaman yang lebih mendalam • Memerlukan lebih banyak waktu pengembangan awal

Berdasarkan Tabel 2.1 mengenai perbandingan metode pengembangan aplikasi, metode *MVP* menjadi pilihan metode dalam pengembangan aplikasi bergerak MECHA karena MVP menawarkan pemisahan yang jelas antara logika bisnis dan tampilan pengguna. Hal ini memiliki implikasi signifikan dalam konteks pengujian unit, karena memungkinkan pengujian logika bisnis secara terpisah tanpa terkait dengan UI. Selain itu, MVP juga memberikan fleksibilitas yang cukup tinggi dalam mengelola perubahan pada antarmuka pengguna. Jika terdapat perubahan atau penyesuaian pada tampilan, hal ini dapat diimplementasikan tanpa memengaruhi bagian logika bisnis, yang mempercepat proses pengembangan.

2.5 Agile

Pengembangan Agile adalah suatu pendekatan yang menekankan pada kecepatan pelaksanaan, responsibilitas yang cepat terhadap perubahan permintaan klien, dan partisipasi aktif dari semua pihak dalam proses, sehingga produk atau perangkat lunak yang dihasilkan merupakan kolaborasi dari semua yang terlibat(Firdha Pradana & Kuswinardi, 2020), kolaborasi tim, dan adaptasi terhadap perubahan. Dengan membagi proyek menjadi iterasi pendek, seperti sprint dalam Scrum, tim fokus pada komunikasi aktif, kerjasama dengan pemangku kepentingan, dan respons cepat terhadap perubahan kebutuhan. *Agile* adalah

model pengembangan perangkat lunak dalam jangka pendek, untuk kemudian diadaptasi secara cepat dalam mengatasi setiap perubahan. Nilai terpenting dari *Agile* development ini adalah memungkinkan sebuah tim dalam mengambil keputusan dengan cepat, kualitas dan prediksi yang baik, serta memiliki potensi yang baik dalam menangani setiap perubahan (Hikmah dkk., 2021).

2.6 Waterfall

Model Pengembangan Air Terjun (Waterfall), merupakan paradigma model pengembangan perangkat lunak paling tua, dan paling banyak dipakai(Riastuti & Chandra, 2022). Setiap tahap harus selesai sebelum memulai tahap berikutnya. Tahapan umum dalam model waterfall meliputi analisis, perancangan, implementasi, pengujian, dan pemeliharaan. Pendekatan ini kurang fleksibel terhadap perubahan dan seringkali tidak cocok untuk proyek-proyek yang kompleks atau berbasis kebutuhan yang cenderung berubah. Pengembangan ini bersifat linear dari awal pengembangan sistem yang berarti dari tahapperencanaan sampai dengan tahap akhir pengembangan sistem yaitu tahap pemeliharaan. Langkah berikutnya tidak akan dilakukan sebelum langkah sebelumnya selesai dilakukanselain itu juga tidak bisa kembali atau mengulang ke tahap sebelumnya(Deva Purwanto & Zukhri, 2023)

2.7 Lean

Pendekatan pengembangan *Lean Startup* adalah metode yang menekankan pengujian ide bisnis dengan cepat dan efisien untuk meminimalkan risiko kegagalan. Ini melibatkan siklus berulang dari membangun, mengukur, dan mempelajari iterasi produk untuk mendapatkan umpan balik pasar secepat mungkin. Lean Startup juga mendorong penggunaan *minimum viable product* (produk yang memenuhi kebutuhan dasar) untuk menguji hipotesis bisnis. Pendekatan ini membantu perusahaan untuk lebih responsif terhadap pasar dan mengoptimalkan penggunaan sumber daya. Belajar dari pengalamannya dalam membangun suatu startup yaitu, menghabiskan waktu bertahun-tahun lamanya untuk menyempurnakan teknologi tersebut dengan membuat minimum viable product tanpa menerima masukan dari konsumen. Selain itu, belum mengetahui proses apa yang akan dibuat dalam membangun startup juga menjadi salah satu mengapa metode ini muncul (Mernisiola Zipa dkk., 2020).

2.8 Perbandingan Pendekatan Pengembangan

Dari pendekatan pengembangan yang telah dibahas sebelumnya, terbentuklah suatu tabel perbandingan yang umumnya digunakan dalam pengembangan aplikasi. Tabel 2.2 dibentuk untuk memperlihatkan perbandingan antara metode-metode dalam pengembangan aplikasi.

Tabel 2.2 Tabel Perbandingan Pendekatan

No	Metode	Penulis dan Tahun Terbit	Maksud Penelitian	Kelebihan	Kekurangan
1	*Agile*	Hikmah Nur, Suradika Agus, Andi Ahmad Gunandi (2021)	Penggunaan Metode Agile Untuk Meningkatkan Kreativitas Guru Melalui Berbagi Pengetahuan (Knowledge Sharing) (Studi Kasus: SDN Cipulir 03 Kebayoran Lama, Jakarta)	• Fleksibilitas dalam perubahan • Keterlibatan pengguna yang lebih aktif • Pengiriman produk yang lebih cepat	• Memerlukan keterlibatan pengguna yang tinggi • Kesulitan dalam perencanaan jangka panjang • Memerlukan keterampilan manajemen proyek yang kuat
2	*Waterfall*	Arya Gemilang Deva Purwanto, Zainudding Zukhri (2023)	Pengembangan Perumahan Purimas Citra Gemilang 2 Berbasis Mobile menggunakan metode waterfall	• Struktur yang jelas dan tertata • Dokumentasi yang kuat • Cocok untuk Proyek dengan persyaratan stabil	• Keterbatasan dalam menanggapi perubahan • Resiko identifikasi terlambat • Penyelesaian proyek lama dan kemungkinan keterlambatan
3	*Lean*	Mutia Memisiola Zipa, Beni Suranto, Irving Vitra Paputungan (2018)	Penerapan Metode Lean Startup Pada Aplikasi EVORIA	• Validasi cepat ide bisnis • Penghematan sumber daya • Meminimalkan Risiko Keuangan	• Tidak cocok untuk Proyek besar dan kompleks • Memerlukan keterampilan evaluasi dan analisis yang kuat • Dapat membutuhkan waktu lebih lama untuk produk yang matang

Berdasarkan Tabel 2.2 mengenai perbandingan pendekatan pengembangan aplikasi, pendekatan *Agile* menjadi pilihan pendekatan dalam pengembangan aplikasi bergerak MECHA karena penggunaan pendekatan Agile mungkin paling sesuai karena memungkinkan penanganan perubahan kebutuhan dengan cepat, melibatkan pengguna secara aktif, dan memungkinkan pengiriman produk secara iteratif. Hal ini akan memungkinkan penelitian untuk mempertahankan fleksibilitas dan responsif terhadap kebutuhan dan umpan balik pengguna dalam pengembangan aplikasi.

BAB III
METODOLOGI

Model View Presenter (MVP) merupakan pola desain yang diadopsi penulis dalam pengembangan aplikasi MECHA. Tujuan dari MVP adalah memisahkan antara antarmuka pengguna (*View*) dengan logika bisnis (*Model*), dihubungkan oleh *presenter* sebagai perantara. Hal ini menghadirkan sejumlah keuntungan, termasuk kemudahan dalam pengujian, perawatan yang lebih baik, dan kode yang lebih terstruktur dan dapat dikelola secara efisien. Penulis juga menerapkan pendekatan *Agile* sebagai metode pengembangan..

3.1 Rancangan Pengembangan

Rancangan pengembangan ini menjelaskan pendekatan yang digunakan dalam pengembangan aplikasi MECHA. Penjelasan mengenai pendekatan pengembangan, pemilihan teknologi, serta alur kerja aplikasi akan diuraikan dalam bagian ini.

3.1.1 Pendekatan Pengembangan

Pendekatan pengembangan yang digunakan dalam perintisan bisnis ini adalah pendekatan berbasis siklus hidup pengembangan perangkat lunak (*software development life cycle*) yang menggabungkan prinsip-prinsip *Agile* dan iterasi. Pendekatan ini dipilih karena fleksibilitasnya dalam menangani perubahan kebutuhan dan kemampuanya untuk menghasilkan aplikasi yang adaptif dan berkualitas.

3.1.2 Pemilihan Teknologi

Pemilihan teknologi merupakan Langkah penting dalam pengembangan aplikasi MECHA. *Platform Android* dipilih sebagai *platform* utama untuk mengembangkan aplikasi ini karena dominasi *Android* di pasar *smartphone*. Selain itu, penggunaan Bahasa pemrograman *Kotlin* dipilih sebagai Bahasa utama untuk pengembangan aplikasi berbasis *Android*. *Android* Studio, sebagai *Integrated Development Environment* (IDE) resmi dari *Android*, digunakan sebagai alat pengembangan utama.

3.1.3 Alur Kerja Aplikasi

Alur kerja aplikasi MECHA terdiri dari beberapa tahapan yang mencakup proses pendaftaran pengguna, pemilihan layanan perbaikan kendaraan, hingga proses perbaikan. Pengguna akan mengakses aplikasi melalui antarmuka pengguna (UI) yang terdiri dari

berbagai tampilan yang terkait dengan masing-masing tahapan. *Presenter* dalam pola *MVP* akan mengelola logika bisnis dan interaksi antara *Model* (data) dan *View* (antarmuka pengguna).

3.1.4 Struktur Aplikasi

Struktur ini adalah kerangka kerja yang mendefinisikan bagaimana berbagai bagian aplikasi berinteraksi, berkomunikasi, dan bekerja sama untuk menciptakan fungsionalitas keseluruhan. Sebuah struktur aplikasi yang baik dirancang dengan tujuan untuk meningkatkan keterbacaan kode, memudahkan pemeliharaan, serta memfasilitasi pengembangan tim yang efisien. Dalam aplikasi MECHA terdapat 2 komponen utama, berikut penjelasanya.

a. Front-end Aplikasi MECHA

Aplikasi MECHA direncanakan akan terdiri dari komponen-komponen inti yang terdiri dari Model, View, dan Presenter sesuai dengan pola *MVP*. Bagian Model akan bertugas mengelola serta mengolah data, sedangkan View akan bertanggung jawab mengatur tampilan antarmuka pengguna. Di sisi lain, peran Presenter akan menjadi penghubung antara Model dan View. Dengan keterkaitan yang ada di antara ketiga komponen ini, aplikasi akan mampu memisahkan tanggung jawab dengan lebih jelas, mengakibatkan terciptanya pemisahan fungsionalitas yang lebih terstruktur dalam pengembangan aplikasi tersebut.

b. Back-end Aplikasi MECHA

Back-end diimplementasikan dengan menggunakan *Framework Adonis* yang berbasis JavaScript, yang menyediakan struktur kerangka kerja yang kuat untuk mengembangkan layanan *API*. Data disimpan dalam basis data *PostgreSQL*, yang divisualisasikan menggunakan *DBeaver*, alat manajemen basis data yang mempermudah administrasi dan visualisasi basis data. *API* yang digunakan oleh aplikasi MECHA diuji menggunakan *Postman*, memungkinkan pengujian dan pemantauan endpoint-endpoint *API* dengan mudah. Dengan menggunakan kombinasi teknologi ini, backend aplikasi MECHA dapat diatur dengan baik, menyediakan fungsionalitas yang handal dan responsif dalam mendukung interaksi antara pengguna dan layanan perbaikan kendaraan.

3.2 Pendekatan Pembangan Aplikasi

Pada pengembangan MECHA akan digunakan Agile sebagai pendekatan pengembangan. Metode Agile memungkinkan tim pengembang untuk bekerja secara iteratif, memecah proyek menjadi sprints singkat yang menghasilkan progres terukur. Setiap sprint menghasilkan bagian dari prototipe MECHA, yang dapat diuji oleh pengguna untuk mendapatkan umpan balik awal. Hal ini memungkinkan penyesuaian dan perbaikan berkelanjutan selama tahap pengembangan. Dengan fokus pada responsivitas terhadap perubahan dan kolaborasi tim yang kuat, pendekatan Agile telah membantu memastikan bahwa MECHA terus berkembang dengan tepat sesuai kebutuhan dan harapan pengguna.

3.2.1 Tahap Perencanaan

Tahap perencanaan mencakup definisi tujuan aplikasi, pengidentifikasian fitur dan fungsionalitas utama, serta penyusunan rencana Proyek. Dalam tahap ini, dilakukan analisis kebutuhan pengguna, pembuatan user stories, dan pemetaan alur kerja dalam aplikasi.

3.2.2 Tahap Development

Pada tahap ini, dilakukan perancangan antarmuka pengguna (UI) yang berdasarkan alur kerja yang telah didefinisikan. Desain antarmuka akan meliputi tampilan layer, navigasi, dan elemen-elemen grafis lainnya. Selain itu, perancangan struktur database dan diagram alur aplikasi juga dilakukan dalam tahap ini.

3.2.3 Tahap Implementasi

Implementasi melibatkan pengembangan kode aplikasi berdasarkan desain yang telah disusun sebelumnya. Bahasa pemrograman Kotlin akan digunakan untuk mengembangkan komponen-komponen aplikasi seperti Model, View, dan Presenter.

3.2.4 Tahap Testing

Tahap testing dalam pengembangan aplikasi merupakan proses krusial yang bertujuan untuk memastikan keandalan, kinerja, dan keamanan dari aplikasi yang dikembangkan.

3.3 Model-View-Presenter (*MVP*) sebagai *Design Pattern*

Model-View-Presenter (*MVP*) adalah sebuah pola desain (*design pattern*) arsitektur perangkat lunak yang bertujuan untuk memisahkan logika presentasi dari tampilan antarmuka pengguna dan logika bisnis. Pola ini membantu meningkatkan modularitas, pengujian dan pemeliharaan kode dengan memisahkan peran dan tanggung jawab setiap komponen dalam aplikasi.

3.3.1 Konsep Dasar *MVP*

MVP terdiri dari tiga komponen utama yang bekerja Bersama untuk mencapai tujuan tertentu dalam aplikasi. Ketiga komponen tersebut adalah:

a. Model: Komponen ini bertanggung jawab atas pengelolaan data, logika bisnis, dan pemrosesan informasi. Model menggambarkan bagaimana data disimpan, diperoleh, dan diolah sebelum diteruskan ke komponen lainya.
b. View: Komponen ini merupakan antarmuka pengguna (UI) yang menampilkan informasi kepada pengguna dan menerima masukin dari pengguna. View tidak memiliki logika bisnis, melainkan hanya berfungsi sebagai representasi visual dari data yang diberikan oleh presenter.
c. Presenter: Komponen ini berperan sebagai perantara antara Model dan View. Presenter mengambil data dari model, memprosesnya sesuai dengan logika bisnis, dan kemudian menyajikan hasilnya kepada View. Presenter juga merespons masukan dari pengguna dan memastikan tampilan antarmuka pengguna selalu sesuai dengan keadaan data.

3.3.2 Keuntungan Penggunaan *MVP*

Penggunaan pola *MVP* dalam pengembangan aplikasi MECHA memiliki beberapa keuntungan, antara lain:

a. Pemisahan Tanggung Jawab: *MVP* memungkinkan pemisahan yang jelas antara logika presentasi, tampilan, dan bisnis. Hal ini memudahkan pengembang untuk fokus pada satu aspek tanpa mengganggu yang lain, dan memudahkan pemeliharaan kode.
b. Pengujian Lebih Mudah: Dengan pemisahan yang jelas antara komponen, pengujian dapat dilakukan lebih efisien. Logika bisnis dapat diuji tanpa harus melibatkan antarmuka pengguna, dan sebaliknya, tampilan antarmuka pengguna dapat duji tanpa perlu mengakses data bisnis secara langsung.

c. Skalabilitas: Struktur *MVP* memudahkan pengembangan secara tim. Pengembang dapat bekerja pada komponen yang berbeda tanpa saling mengganggu, asalkan mereka mengikuti kontrak antara Presenter dan View.

3.3.3 Implementasi *MVP* dalam Aplikasi MECHA

Penerapan *MVP* dalam aplikasi akan mengikuti prinsip-prinsip yang dielaskan di atas. Presenter akan mengatur logika bisnis seperti pendaftaran layanan dan interaksi dengan model. View akan menampilkan informasi kepada pengguna dan mengirimkan masukan pengguna kepada Presenter. Model akan mengelola data yang diperlukan oleh aplikasi.

3.4 Metode Pengumpulan Data

Metode pengumpulan data merupakan langkah penting dalam mendapatkan informasi yang diperlukan untuk pengembangan aplikasi MECHA. Metode-metode yang digunakan dalam pengumpulan data akan diuraikan dalam bagian ini.

3.4.1 Studi Literatur

Studi literatur dilakukan untuk memahami konsep dasar tentang aplikasi perbaikan kendaraan, teknologi *Android*, dan pola desain Model-View-Presenter (*MVP*). Sumber-sumber literatur seperti jurnal dan artikel ilmiah digunakan untuk memperoleh pemahaman yang kokoh sebelum memulai pengembangan aplikasi.

3.4.2 Wawancara

Wawancara dilakukan dengan beberapa pihak terkait, termasuk pemilik bengkel perbaikan kendaraan, teknisi, dan pengguna potensial aplikasi. Wawancara digunakan untuk mendapatkan wawasan tentang proses perbaikan kendaraan, kebutuhan pengguna, serta tantangan yang dihadapi dalam layanan perbaikan kendaraan saat ini.

3.4.3 Survei

Survei dilakukan menggunakan google form yang diisi oleh calon pengguna aplikasi. Survei dilakukan untuk mendapatkan wawasan tentang keluhan yang dialami oleh calon pengguna aplikasi.

3.5 *Prototype* Pada Mecha

Prototipe Mecha dibuat sebagai representasi perangkat lunak yang sederhana tetapi berfungsi, memberikan gambaran visual dan interaktif tentang fitur utama yang akan ada dalam aplikasi final. Pada tahap ini, perhatian utama adalah memvalidasi konsep dan merancang antarmuka pengguna agar sesuai dengan kebutuhan pengguna. Fungsionalitas prototipe Mecha dirancang untuk memberikan pemahaman awal tentang navigasi aplikasi, tata letak elemen antarmuka, dan alur kerja dasar. Meskipun kemampuannya mungkin terbatas, prototipe ini memberikan gambaran yang jelas tentang pengalaman pengguna yang diinginkan. Dengan menggunakan prototipe, tim pengembangan dapat mengumpulkan umpan balik dari pengguna awal, mengevaluasi kemudahan penggunaan, dan mengidentifikasi area yang perlu diperbaiki atau ditingkatkan sebelum melanjutkan ke tahap pengembangan berikutnya. Oleh karena itu, bagian ini menekankan peran krusial prototipe dalam siklus pengembangan perangkat lunak Mecha, menjelaskan bagaimana implementasi awal ini membantu membentuk arah pengembangan berikutnya. Pengguna nantinya dapat menguji *Prototype* ini untuk mendapatkan masukan dan perbaikan lebih lanjut sebelum produk akhirnya dibuat (Pratama dkk., 2023) .

Tabel 3.1 Tech Stack Mecha

User Interface	Mobile App – Kotlin, XML,
Database	PostgreSql, Ngrok, AdonisJS, JavaScript
OS	Windows, Android
Network	Client Devices

Rancangan aplikasi Mecha dijelaskan dalam bentuk *Tech Stack* seperti yang sudah tertera pada Tabel 3.1.

3.6 *Minimum Viable Product* Pada Mecha

Minimum Viable Product (MVP) diterapkan dengan tujuan memberikan fungsionalitas inti yang memungkinkan pengguna melakukan pemesanan bengkel secara online. Fokus utama pengembangan MVP berperan sebagai pendekatan yang membantu pengembang dalam mengembangkan produk, baik fisik maupun digital, dengan menyajikan fitur minimum yang dapat memenuhi kebutuhan para pengguna *early adopters*. Produk final kemudian akan disempurnakan setelah mempertimbangkan umpan balik yang diperoleh dari versi awal produk (Ningsih dkk., 2023). Fitur pemesanan bengkel online menjadi inti dari MVP,

memungkinkan pengguna untuk dengan mudah menemukan bengkel terdekat, menjelajahi layanan yang ditawarkan, dan melakukan pemesanan layanan perbaikan kendaraan melalui aplikasi. Dengan mengutamakan fitur yang paling mendasar dan esensial, implementasi MVP Mecha tidak hanya mempercepat peluncuran aplikasi tetapi juga memungkinkan tim pengembang mendapatkan umpan balik awal dari pengguna. Melalui pengenalan Mecha sebagai MVP, tim dapat secara proaktif mengumpulkan masukan, memahami preferensi pengguna, dan melakukan perbaikan iteratif untuk meningkatkan kualitas dan responsivitas aplikasi.

3.7 *System Usability Scale* Pada Mecha

System Usability Scale (SUS) adalah sebuah skala Likert. Seringkali diasumsikan bahwa skala Likert adalah skala berdasarkan pertanyaan pilihan paksa, di mana pernyataan diberikan dan responden kemudian menunjukkan sejauh mana mereka setuju atau tidak setuju dengan pernyataan tersebut pada skala 5 (atau 7) poin (Brooke, 1995). Melibatkan sejumlah calon pengguna, evaluasi ini memanfaatkan kuesioner SUS yang terdiri dari sepuluh pertanyaan terstruktur. Kuesioner tersebut dirancang untuk mengukur berbagai aspek usabilitas, termasuk aspek kognitif dan afektif, sehingga menghasilkan skor yang mencerminkan tingkat kepuasan dan persepsi pengguna terhadap aplikasi. Dengan mempertimbangkan bahwa kegunaan dalam setiap situasi tertentu ditentukan oleh konteks penggunaan sistem, maka dapat diikuti bahwa umumnya cara untuk mengukur kegunaan juga akan terdefinisi oleh konteks tersebut. ISO 9241-11 memecah pengukuran kegunaan menjadi tiga komponen terpisah yang harus didefinisikan berdasarkan konteks penggunaan (Brooke, 2013):

a. Efektivitas (apakah orang dapat benar-benar menyelesaikan tugas mereka dan mencapai tujuan mereka
b. Efisiensi (sejauh mana mereka mengeluarkan sumber daya dalam mencapai tujuan mereka)
c. Kepuasan (tingkat kenyamanan yang mereka alami dalam mencapai tujuan tersebut).

Dalam penjelasan sebelumnya, muncul sepuluh pertanyaan yang memanfaatkan template yang dikembangkan oleh John Brooke, pencipta System Usability Scale (SUS).

Tabel 3.2 Tabel Pertanyaan *System Usability Testing*

No	Pernyataan	Skala
1	Saya akan ingin lebih sering menggunakan aplikasi ini	1 – 5
2	Saya merasa aplikasi ini tidak harus dibuat serumit ini	1 – 5
3	Saya piker aplikasi ini mudah untuk digunakan	1 – 5
4	Saya membutuhkan bantuan dari orang teknis untuk menggunakan aplikasi ini	1 – 5
5	Saya menemukan fitur pada aplikasi ini terintegrasi dengan baik	1 – 5
6	Saya piker ada ketidaksesuaian dalam aplikasi ini	1 – 5
7	Saya merasa kebanyakan orang mudah untuk mempelajari aplikasi dengan sangat cepat	1 – 5
8	Saya menemukan, aplikasi sangat rumit untuk digunakan	1 – 5
9	Saya percaya diri untuk menggunakan aplikasi ini	1 – 5
10	Saya perlu belajar sebelum saya menggunakan aplikasi ini	1 – 5

Sumber : (Ependi dkk., 2019)

Setelah mengumpulkan data dari hasil pengujian terhadap calon pengguna, langkah berikutnya adalah melakukan konversi data ke dalam skala Pertanyaan dengan nomor ganjil maka nilai akan dikurangi dengan 1.

a. Pertanyaan dengan nomor ganjil maka nilai akan dikurangi dengan 1 (2.1)

Skor $System\ Usability\ Scale$ (SUS)ganjil $= \sum$ Px – 1

Px = Jumlah skor pertanyaan ganjil

b. Pertanyaan dengan nomor genap maka skor akhir yang didapat dari nilai 5 dikurangi skor pertanyaan yang didapat (2.2)

Skor $System\ Usability\ Scale$ (SUS) genap $= \sum 5 -$ Pn

Pn = Jumlah skor pertanyaan genap (2.3)

c. Skor *System Usability Scale* (SUS) yang didapat dikalikan 2,5 (2.3)

$(\sum$ skor ganjil $- \sum$skor genap) x 2,5

d. Setelah mendapatkan skor dari setiap responden tahap selanjutnya menghitung hasil secara keseluruhan (2.4)

$$\frac{\sum x}{n}$$

$\bar{x} = skor\ rata - rata$

$\sum x = jumlah\ skor\ SUS$

$n = jumlah\ responden$

Setelah mendapatkan skor rata-rata $\sum x$ dari responden kemudian dilakukan perhitungan untuk menentukan grade ataupun peringkat penilaian. Untuk skala penilaian skor beserta peringkat yang didapat dapat dilihat pada Tabel mengenai System Usability Scale (SUS) *percentile rank* berikut:

Tabel 3.3 Peringkat Dalam *System Usability Scale*

Skor SUS	Nilai	Peringkat
>80,3	A	Bagus Sekali
74-80,3	B	Bagus
68-74	C	Baik
51-68	D	Buruk
<51	F	Buruk Sekali

Jika suatu sistem memungkinkan pengguna untuk menyelesaikan tugas, namun dengan penggunaan sumber daya yang besar dan waktu yang lama, serta memberikan pengalaman yang sangat tidak memuaskan bagi semua pihak, maka dapat disimpulkan bahwa sistem tersebut tidak dapat dianggap sebagai sistem yang dapat digunakan.

Penggunaan SUS pada Mecha tidak hanya sekadar mengukur kepuasan pengguna, tetapi juga berfungsi sebagai panduan untuk pengembangan berkelanjutan. Dengan menganalisis skor yang diperoleh, tim pengembang Mecha dapat mengidentifikasi area spesifik yang memerlukan perbaikan atau penyempurnaan. Hasil evaluasi ini memberikan pemahaman yang lebih baik tentang sejauh mana Mecha memenuhi harapan pengguna dalam hal usabilitas, dan membuka peluang untuk meningkatkan kualitas pengalaman pengguna secara menyeluruh. Dengan memanfaatkan SUS, Mecha dapat terus mengembangkan fitur dan antarmuka yang lebih responsif, sesuai dengan kebutuhan dan ekspektasi calon pengguna.

BAB IV
HASIL DAN PEMBAHASAN

Bab ini akan menjelaskan implementasi teknis dari aplikasi MECHA. Setiap komponen aplikasi akan dijelaskan secara rinci, termasuk bagaimana mereka berinteraksi sesuai dengan pola *MVP*.

4.1 Arsitektur Aplikasi

Implementasi aplikasi MECHA didasarkan pada arsitektur Model-View-Presenter (*MVP*). Arsitektur ini memisahkan peran dan tanggung jawab antara *Model*, *View*, dan *Presenter*, memungkinkan keterpisahan logika dan tampilan.

4.2 Implementasi Model

Komponen Model dalam aplikasi MECHA bertanggung jawab atas pengelolaan data dan logika. Data terkait pengguna, layanan perbaikan, dan jadwal layanan diimplementasikan dalam Model.

4.2.1 Register

Dalam aplikasi MECHA. Tampilan Register merupakan salah satu komponen yang membantu user untuk mendaftarkan diri pada aplikasi MECHA. Model bertanggung jawab atas pengelolaan data yang terkait dengan register ini . Kelas `RegisResponse` dalam paket `com.example.MECHAapp.data.Model` adalah representasi dari respons yang diterima dari server setelah proses pendaftaran pengguna di aplikasi MECHAApp. Properti `status` menunjukkan kode status dari respons, `message` berisi pesan deskriptif terkait respons, dan `verified` adalah boolean yang menandakan apakah pendaftaran telah diverifikasi atau tidak. Dengan `RegisResponse`, MECHAApp dapat mengelola respon pendaftaran dengan efisien dan memberikan umpan balik yang tepat kepada pengguna setelah proses pendaftaran berhasil dilakukan.

```
@JsonClass(generateAdapter = true )
data class RegisResponse(
    val status: Int,
    val message: String,
```

```
        val verified: Boolean
    )
```

Gambar 4.1 Model Register

4.2.2 Login

Dalam aplikasi MECHA. Tampilan Login merupakan salah satu komponen yang membantu user untuk masuk pada aplikasi MECHA. Model bertanggung jawab atas pengelolaan data yang terkait dengan register ini . Implementasi Model menggunakan data class `LoginResponse` di atas memfasilitasi representasi struktural dari respons yang diterima dari server saat pengguna berhasil melakukan login. Setiap properti dari model ini (seperti `status`, `id`, `token`, `nama`, `email`, `role`, dan `verified`) secara langsung memetakan elemen-elemen respons ke variabel-variabel yang dapat diakses dan diolah dalam aplikasi. Misalnya, `status` akan mengambil nilai status dari respons, `token` akan menyimpan token otentikasi, dan seterusnya. Anotasi `@JsonClass(generateAdapter = true)` menunjukkan bahwa adapter JSON akan dihasilkan secara otomatis, mempermudah proses marshalling dan unmarshalling data JSON ke dalam objek Kotlin. Dengan model ini, aplikasi dapat mengintegrasikan data dari respons login ke dalam logika bisnisnya dengan cara yang jelas dan mudah dimengerti.

```
    @JsonClass(generateAdapter = true)
    data class LoginResponse(
        val status: Int,
        val id: Int,
        val token: String,
        val nama: String,
        val email: String,
        val role: Int,
        val verified: Boolean
    )
```

Gambar 4.2 Model Login

4.2.3 Pesan Layanan

Dalam aplikasi MECHA. Tampilan pesan layanan merupakan salah satu komponen yang membantu user untuk melakukan pemesanan layanan pada aplikasi MECHA. Implementasi Model menggunakan struktur data class OrderModel telah dibuat dengan mempertimbangkan elemen-elemen yang diperlukan untuk merepresentasikan pesanan dalam aplikasi MECHA. Model ini mencakup informasi penting seperti ID pesanan, ID pengguna, ID layanan, nama pengguna, nama layanan, status pesanan, alamat pengiriman, URL peta

lokasi, tanggal pembuatan, jumlah pembayaran, URL gambar, dan daftar harga terkait. Dengan menggunakan Parcelable, model ini dapat dengan mudah dilewatkan antar komponen dalam aplikasi untuk memastikan integrasi data yang efisien. Dengan struktur ini, setiap pesanan dapat direpresentasikan dengan lengkap dan mudah diakses, memfasilitasi pengelolaan dan pemrosesan pesanan secara efektif dalam sistem MECHA.

```
data class OrderModel(
    val id: Int,
    val user_id: Int = 0,
    val id_service: String,
    val name: String,
    val name_service: String,
    val status: String?,
    val address: String,
    val map_url: String,
    val created_at: String,
    val sum: String?,
    val img_url: String,
    val prices: List<PriceModel> = emptyList()
):Parcelable
```

Gambar 4.3 Model Pesan Layanan

4.2.4 Pembayaran

Dalam aplikasi MECHA. Tampilan pembayaran merupakan salah satu komponen yang membantu user untuk melakukan pemesanan layanan pada aplikasi MECHA. Implementasi Model menggunakan model `PriceModel` adalah cara yang efektif untuk merepresentasikan data harga layanan dalam aplikasi MECHA. Model ini mengandung tiga properti utama, yaitu `price` untuk menyimpan informasi harga layanan, `id_service` untuk mengidentifikasi jenis layanan, dan `description_service` untuk menyimpan deskripsi terkait layanan tersebut. Dengan menerapkan Parcelable, objek dari `PriceModel` dapat dengan mudah dilewatkan antar komponen dalam aplikasi, seperti aktivitas dan fragmen, sehingga memungkinkan pengelolaan data harga secara efisien. Ini memungkinkan aplikasi untuk mengakses dan memanfaatkan informasi harga dengan cepat dan akurat, membantu pengguna dalam memilih layanan yang sesuai dengan kebutuhan mereka.

```
data class PriceModel(
    val price: String,
    val id_service: String = "",
    val description_service: String
): Parcelable
```

Gambar 4.4 Price Model

4.2.5 Riwayat Pemesanan

Tampilan riwayat pemesanan adalah salah satu fitur yang memungkinkan pengguna melihat riwayat pemesanan yang telah dilakukan. Model bertanggung jawab atas pengelolaan data yang terkait dengan riwayat pemesanan ini. Kelas `RiwayatItemModel` yang `com.example.MECHAap.features.data.model` adalah representasi dari item riwayat dalam aplikasi MECHAApp. Setiap objek dari kelas ini memiliki beberapa atribut penting seperti `id` (identifikasi unik), `title` (judul item), `pemesan` (nama pemesan), `date` (tanggal pemesanan), dan `status` (status item). Dengan demikian, `RiwayatItemModel` memungkinkan MECHAApp untuk menyimpan dan mengelola informasi riwayat dengan efisien, memberikan pengalaman yang lebih baik bagi pengguna.

```
@Parcelize
     class RiwayatItemModel (
         val id: String,
         val title: String,
         val pemesan: String,
         val date: String,
         val status: String
             ):Parcelable
```

Gambar 4.5 Model Riwayat Pesanan

4.2.6 Detail Pesanan

Tampilan Detail Pesanan adalah salah satu fitur yang memungkinkan pengguna melihat detail pemesanan yang telah dilakukan. Model bertanggung jawab atas pengelolaan data yang terkait dengan detail pemesanan ini. Kelas `DetailMontirModel` dalam package `com.example.MECHAapp.data.Model` adalah sebuah model data yang digunakan untuk merepresentasikan detail informasi terkait dengan montir dalam aplikasi MECHAApp. Kelas ini memiliki dua atribut, yaitu `juduldetail` dan `judulharga`, yang masing-masing digunakan untuk menyimpan judul detail informasi dan judul harga yang terkait dengan montir. Dengan demikian, `DetailMontirModel` berperan penting dalam menyediakan informasi yang relevan kepada pengguna tentang layanan montir yang tersedia dalam aplikasi.

```
@Parcelize
data class DetailMontirModel (
    val juduldetail: String,
    val judulharga: String
): Parcelable
```

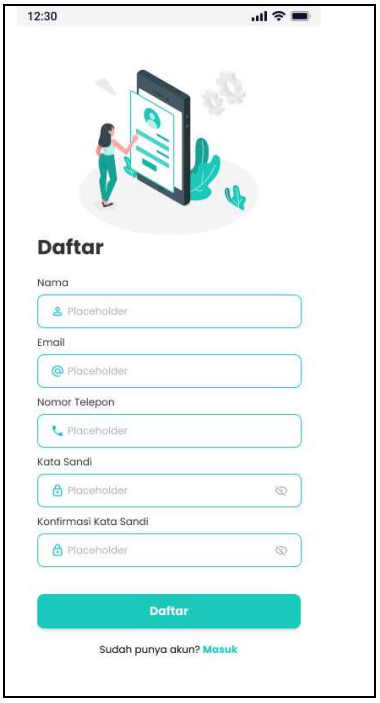
Gambar 4.6 Model Detail Pemesanan

4.3 Implementasi View

Komponen View dalam aplikasi MECHA bertanggung jawab atas tampilan antarmuka pengguna. Setiap layar dan elemen UI diimplementasikan dalam View. Penggunaan Activity layout untuk merancang tampilan layar dan koneksi antara View dengan Presenter dijelaskan dalam bagian ini.

4.3.1 Register User

Tampilan registrasi merupakan bagian dari antarmuka pengguna (View) yang memungkinkan pengguna untuk membuat akun baru di dalam aplikasi MECHA. Di tampilan ini, pengguna akan diminta untuk memasukkan informasi pribadi seperti nama, alamat email, nomor telepon, dan kata sandi. Selain itu, tombol "Daftar" akan disediakan untuk memulai proses registrasi. Presenter akan bertanggung jawab mengambil input pengguna dari tampilan ini dan mengirimkannya ke Model untuk diproses. Setelah registrasi berhasil, pengguna akan diberikan akses ke fitur-fitur lain dalam aplikasi MECHA. Pada implementasi yang ditunjukkan Gambar dapat dilihan penggunaan function onError, dan onErrorSuccess yang mana akan menampilkan pesan apabila pada pengisian formulir terdapat kesalahan.



Gambar 4.7 UI Daftar

```
override fun onLoading() {
    TODO("Not yet implemented")
}

override fun onFinishedLoading() {
    TODO("Not yet implemented")
}

override fun onError(code: Int, message: String) {
    when(code) {
        1 ->binding.etEmail.error = message
        3 ->binding.etPassword.error = message
        4 ->binding.etPhone.error = message
        5 ->binding.etConpass.error = message
    }
}

override fun onSuccesRegister() {
    TODO("Not yet implemented")
}

override fun onErrorSignup() {
    TODO("Not yet implemented")
}

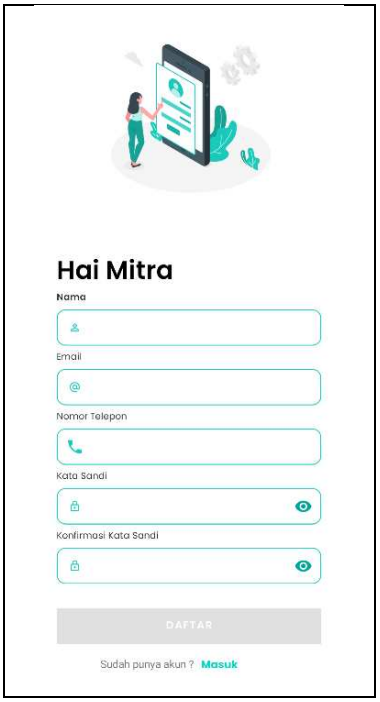
override fun onErrorSuccess(code: Int, message: String) {
    when(code) {
        2 ->binding.etPassword.isErrorEnabled = false
        6 ->binding.etPhone.error = ""
        1 ->binding.etEmail.error = ""
        7 ->binding.etConpass.error = ""
    }
}
```

```
        override fun onErrorAvailable(code: Int, message: String) {
            TODO("Not yet implemented")
        }
    }
```

Gambar 4.8 Kotlin Register

4.3.2 Register Mitra

Tampilan registrasi merupakan bagian dari antarmuka pengguna (View) yang memungkinkan pengguna untuk membuat akun baru di dalam aplikasi MECHA. Di tampilan ini, pengguna akan diminta untuk memasukkan informasi pribadi seperti nama, alamat email, nomor telepon, dan kata sandi. Selain itu, tombol "Daftar" akan disediakan untuk memulai proses registrasi. Presenter akan bertanggung jawab mengambil input pengguna dari tampilan ini dan mengirimkannya ke Model untuk diproses. Setelah registrasi berhasil, pengguna akan diberikan akses ke fitur-fitur lain dalam aplikasi MECHA. Pada implementasi yang ditunjukkan Gambar dapat dilihan penggunaan function onError, dan onErrorSuccess yang mana akan menampilkan pesan apabila pada pengisian formulir terdapat kesalahan.



Gambar 4. 9 Register Mitra

```
        override fun onLoading() {
            TODO("Not yet implemented")
        }

        override fun onFinishedLoading() {
            TODO("Not yet implemented")
        }

        override fun onError(code: Int, message: String) {
            when(code) {
                1 ->binding.etEmail.error = message
                3 ->binding.etPassword.error = message
                4 ->binding.etPhone.error = message
                5 ->binding.etConpass.error = message
            }
        }

        override fun onSuccesRegister() {
            TODO("Not yet implemented")
        }

        override fun onErrorSignup() {
            TODO("Not yet implemented")
        }

        override fun onErrorSuccess(code: Int, message: String) {
            when(code) {
                2 ->binding.etPassword.isErrorEnabled = false
                6 ->binding.etPhone.error = ""
                1 ->binding.etEmail.error = ""
                7 ->binding.etConpass.error = ""
            }
        }

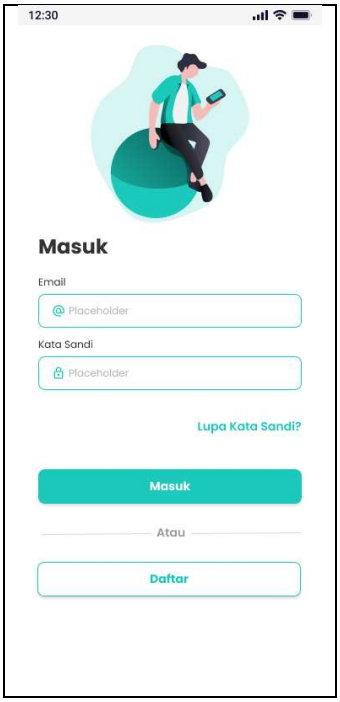
        override fun onErrorAvailable(code: Int, message: String) {
            TODO("Not yet implemented")
        }
    }
```

Gambar 4.10 Kotlin Register Mitra

4.3.3 Login

Tampilan login adalah antarmuka pengguna yang memungkinkan pengguna untuk melakukan proses autentikasi dalam aplikasi MECHA. Dalam tampilan ini, pengguna akan diminta untuk memasukkan informasi seperti nama pengguna (username) dan kata sandi (password) mereka. Tombol "Login" akan memicu aksi autentikasi, yang akan diambil oleh Presenter untuk memeriksa informasi pengguna dengan data yang ada di Model. Jika autentikasi berhasil, pengguna akan diarahkan ke tampilan utama aplikasi, sedangkan jika autentikasi gagal, pesan kesalahan akan ditampilkan kepada pengguna. Pada implementasi yang ditunjukkan Gambar dapat dilihan penggunaan function onError, onSuccesLogin,

onErrorEmpty, onErrorFalse yang mana akan menampilkan pesan apabila pada pengisian formulir terdapat kesalahan.



Gambar 4.11 UI Login

```
override fun onError(code: Int, message: String) {
    when(code){
        // nampilin error message
        2 -> binding.tilemailLogin.error = message
        4 -> binding.tilpwLogin.error = message
    }
}


override fun onSuccesLogin() {
    AlertDialogLoginSucces().show(supportFragmentManager,"test")
    Handler(Looper.getMainLooper()).postDelayed({

        startActivity(Intent(this, NavbarContainer::class.java))
        finish()
    }, 2000)
}

override fun onErrorLogin() {
    AlertDialogLoginGagal().show(supportFragmentManager,"test")
}
override fun onErrorEmpty(code: Int) {
    when(code){
        //ngilangin error message
        1 -> binding.tilemailLogin.error=""
        3 -> binding.tilpwLogin.error=""
    }
}
```

```
override fun onErrorFalse(code: Int, message: String) {
    when(code){
        5 -> binding.tilemailLogin.error=message
        6 -> binding.tilpwLogin.error=message
    }

}
```

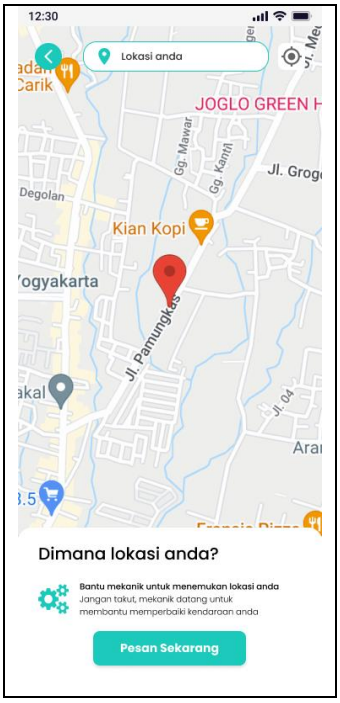
Gambar 4.12 Kotlin Login

4.3.4 Pesan Layanan

Tampilan pesan layanan akan dirancang agar pengguna dapat melakukan pemesanan layanan aplikasi MECHA mulai dari pengecekan lokasi pengguna hingga berhasil membuat sebuah pesanan.

a. Implementasi View Map

Implementasi view pada kelas `MapAddress` ini mengintegrasikan fungsionalitas peta dan lokasi. Pertama, tampilan inisialisasi menggunakan metode `onCreate` dan `setContentView` untuk mempersiapkan antarmuka pengguna. Lokasi pengguna diakses melalui `FusedLocationProviderClient`, dan LocationRequest disiapkan untuk memprioritaskan akurasi tinggi. Selanjutnya, peta diperoleh melalui `getMapAsync` dan ditangani oleh metode `onMapReady`. Kemudian, saat pengguna meminta pembaruan lokasi melalui tombol atau dengan memanggil `requestLocationUpdates`, aplikasi memeriksa dan meminta izin lokasi. Setelah izin diberikan, pembaruan lokasi ditangani oleh `locationCallback`. Data lokasi diperbarui dan digunakan untuk memperbarui peta, menyesuaikan tampilan kamera, dan menambahkan marker pada peta. Fungsi `getAddress` digunakan untuk mendapatkan alamat dari koordinat geografis. Jika berhasil, alamat diperbarui pada antarmuka pengguna. Fungsi `createMapUrl` membentuk URL peta dengan memanfaatkan informasi lokasi pengguna. Selain itu, terdapat pemrosesan untuk memastikan bahwa lokasi pengguna tetap terbarui secara real-time dan peta ditampilkan dengan benar. Fungsi-fungsi ini diintegrasikan dalam kelas `MapAddress` untuk memberikan pengalaman pengguna yang responsif dan memungkinkan pengguna untuk melakukan pemesanan dengan mudah berdasarkan lokasi mereka.



Gambar 4.13 UI Map

```
      private fun requestLocationUpdates() {
            val permissions = arrayOf(
                Manifest.permission.ACCESS_FINE_LOCATION,
                Manifest.permission.ACCESS_COARSE_LOCATION
            )
            val granted = permissions.all {
                ActivityCompat.checkSelfPermission(this, it) ==
PackageManager.PERMISSION_GRANTED
            }

            if (!granted) {
                ActivityCompat.requestPermissions(this, permissions,
LOCATION_PERMISSION_REQUEST_CODE)
                return
            }

            fusedLocationClient.requestLocationUpdates(locationRequest,
locationCallback, null)
        }


        private fun getAddress(context: Context, latitude: Double,
longitude: Double): String {
            val geocoder = Geocoder(context, Locale.getDefault())
            val addresses: List<Address>
            return try {
                addresses = geocoder.getFromLocation(latitude, longitude,
1) as List<Address>
                if (addresses.isNotEmpty()) {
                    addresses[0].getAddressLine(0) ?: ""
                } else {
                    "No address found"
```

```
                }
            } catch (e: IOException) {
                e.printStackTrace()
                "Error fetching address"
            }
        }
        private fun createMapUrl(latitude: Double, longitude:
Double,mapType: String = "roadmap"): String{
            //Create map URL
            val urlBuilder = StringBuilder("geo:")
            if (DataAddress.address.isNotBlank()) {
                urlBuilder.append("$longitude,
$latitude?q=(${DataAddress.address})")
            }
            if (mapType.isNotBlank()) {
                urlBuilder.append("&t=$mapType")
            }

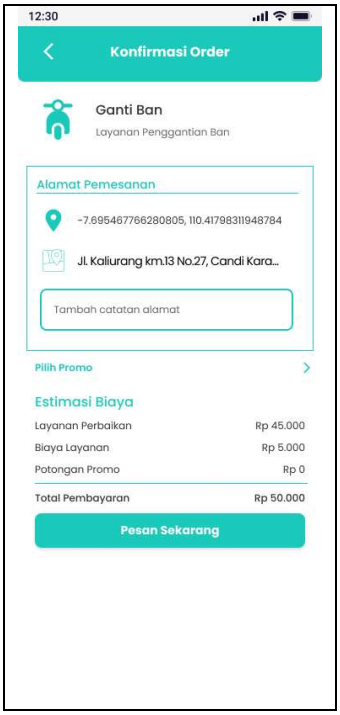
            return urlBuilder.toString()
        }

        companion object {
            private const val LOCATION_PERMISSION_REQUEST_CODE = 1
        }
    }
```

Gambar 4.14 Kotlin Map

b. Implementasi Pesan Layanan

Implementasi View dalam kelas `ConfirmationOrder` bertujuan untuk menyajikan antarmuka pengguna (UI) yang memungkinkan pengguna untuk mengonfirmasi pesanan mereka. Melalui layout XML yang diinflasi dari `ActivityConfirmationOrderBinding`, elemen-elemen seperti alamat, tautan peta, dan tombol pemesanan diperbarui dengan data yang sesuai. Pengguna juga diberi kemampuan untuk membuka tautan peta menggunakan `Intent`. Ketika tombol "Pesan Sekarang" ditekan, presenter akan dipanggil untuk mengirim pesanan ke backend dengan menggunakan informasi yang telah dimasukkan. Jika pengguna memilih untuk kembali (melalui tombol kembali), mereka akan diarahkan kembali ke layar sebelumnya. Inisialisasi presenter dan pengikatan antarmuka ini memastikan bahwa logika dan tampilan bekerja secara terintegrasi, menciptakan pengalaman pengguna yang mulus dalam mengonfirmasi pesanan mereka.



Gambar 4.15 UI Konfirmasi Order

```
        override fun onCreate(savedInstanceState: Bundle?) {
            binding =
ActivityConfirmationOrderBinding.inflate(layoutInflater)
            url.data = Uri.parse(DataAddress.mapUrl)
            super.onCreate(savedInstanceState)
            setContentView(binding.root)
            layanan = intent.getStringExtra("layanan").toString()
            presenter = OrderPresenter(this, OrderAPI()).apply {
                onAttach(this@ConfirmationOrder)
            }

            binding.apply{
                tvAlamat.text = DataAddress.address
                tvUrl.text = DataAddress.mapUrl
                tvUrl.setOnClickListener {
                    startActivity(url)
                }
            }
            binding.btnPesan.setOnClickListener {
                presenter.postOrder(layanan, "Menunggu",
binding.tvAlamat.text.toString(), binding.tvUrl.text.toString())
            }
            binding.ivbackConfirm.setOnClickListener {
                startActivity(Intent(this@ConfirmationOrder,
MapAddress::class.java))
            }
            runOnUiThread {
                var harga = 0
                when(layanan){
                    "Ganti Ban" -> harga = 15000
                    "Ganti Velg" -> harga = 15000
                    "Ganti Oli" -> harga = 30000
                    "Kendaraan Mogok" -> harga = 30000
```

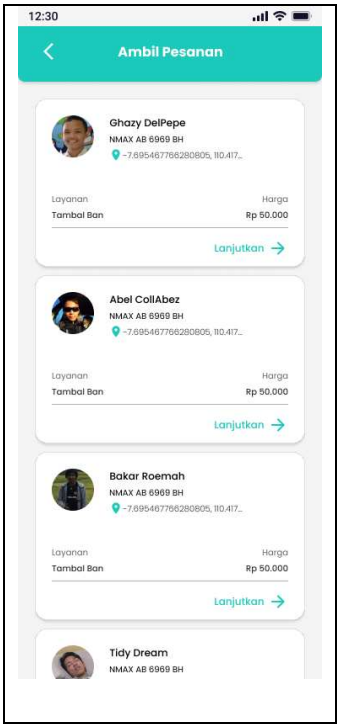
```
                    "Tambal Ban" ->harga = 15000
                    "Injeksi Motor" -> harga = 110000
                    "Karburator Motor" ->harga = 110000
                }
                binding.tvHarga.text = "Rp
${harga.formatDecimalSeparator()}"
                binding.tvBiaya.text = "Rp
${(harga*20/100).formatDecimalSeparator()}"
                binding.tvSumLayanan.text = "Rp
${(harga+(harga*20/100)).formatDecimalSeparator()}"
            }

        }
```

Gambar 4.16 Kotlin Konfirmasi Order

c. Implementasi Ambil Pesan Montir

Dalam kelas `AmbilPesan`, terjadi implementasi tampilan (View) yang mengatur antarmuka pengguna untuk fitur "Ambil Pesan". Pertama, layout dari aktivitas ini diisi menggunakan data dari file layout `activity_ambilpesan.xml`. Selanjutnya, inisialisasi presenter (`AmbilPesanPresenter`) dan adapter (`AmbilListAdapter`) dilakukan. Presenter bertanggung jawab untuk mengelola logika bisnis terkait ambil pesanan, sementara adapter digunakan untuk menghubungkan data ke tampilan daftar yang akan ditampilkan. Selanjutnya, ketika aktivitas dibuat (`onCreate`), fungsi-fungsi penting seperti meminta daftar pesanan dari presenter (`presenter.getOrder()`) dan mengatur fungsi klik kembali (`binding.impiwbackPesan.setOnClickListener`) diimplementasikan. Kemudian, tata letak dari daftar pesanan diatur dengan menggunakan `LinearLayoutManager` dan adapter (`binding.rvAmbil.adapter = adapterHistory` dan `binding.rvAmbil.layoutManager = layoutManager`). Terakhir, penanganan klik item di daftar diterapkan dengan menginisialisasi fungsi `rvClickListener` yang akan memulai aktivitas `DetailPesanan` dengan memberikan data pesanan yang terkait. Keseluruhan implementasi ini memastikan pengguna dapat melihat dan memilih pesanan dengan nyaman dalam antarmuka yang disediakan.



Gambar 4.17 UI Ambil Pesan Montir

```
override fun onCreate(savedInstanceState: Bundle?){
     binding = ActivityAmbilpesanBinding.inflate(layoutInflater)
     super.onCreate(savedInstanceState)
     setContentView(binding.root)

     presenter = AmbilPesanPresenter(this, OrderAPI()).apply {
         onAttach(this@AmbilPesan)
     }
     presenter.getOrder()

     binding.impiwbackPesan.setOnClickListener {
        onBackPressedDispatcher.onBackPressed()
     }

     val layoutManager = LinearLayoutManager(this)

     binding.rvAmbil.adapter = adapterHistory
     binding.rvAmbil.layoutManager = layoutManager
     adapterHistory.setOnclickItem(rvClickListener)

 }
 private val rvClickListener: ((OrderModel) -> Unit) = {
     item ->
     startActivity(Intent(this, DetailPesanan::class.java).apply {
         putExtra("order", item)
     })
 }
```

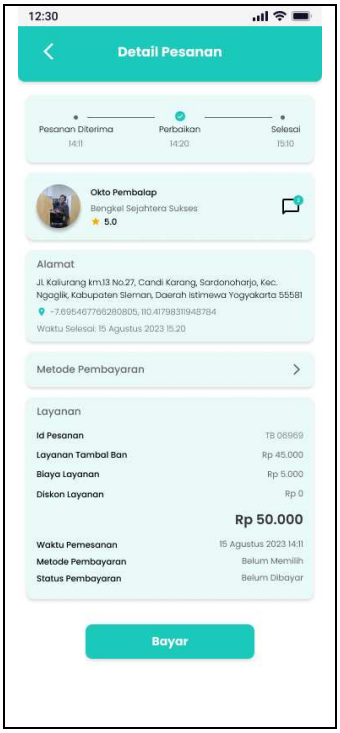
Gambar 4.18 Kotlin Ambil Pesan Montir

4.3.5 Pembayaran

Tampilan pembayaran akan dirancang agar pengguna dapat melakukan pembayaran setelah melakukan pemesanan layanan aplikasi MECHA.

a. Implementasi Pembayaran

Implementasi view di atas terjadi dalam aktivitas "DetailPesanan". Pertama, tata letak (layout) diinflasi menggunakan `ActivityDetailPesananBinding`. Selanjutnya, data pesanan diterima dari aktivitas sebelumnya. Presenter untuk aktivitas ini diinisialisasi dengan memanggil `DetailPesananPresenter` dan `OrderAPI`, dan kemudian di-attach ke aktivitas. Kemudian, konfigurasi RecyclerView dilakukan dengan menetapkan adapter dan layout manager. Informasi pengguna dan alamat pesanan diisi dengan memanfaatkan data dari objek `dataOrder`. Pengguna diberikan kemampuan untuk mengklik tombol untuk melakukan beberapa tindakan. Tombol "cvMetode" mengarahkan pengguna ke aktivitas "MetodePembayaran", tombol "ivBackdetail" kembali ke halaman sejarah, dan tombol "cvPayService" mengarahkan pengguna ke aktivitas "MenungguKonfirmasiPembayaran", dengan data pesanan yang dikirimkan bersama. Jika status pesanan adalah "Selesai", maka tombol "cvPayService" akan disembunyikan. Selanjutnya, informasi tambahan seperti nama pengguna, alamat, dan status pesanan ditampilkan, dan gambar yang terkait dengan pesanan dimuat menggunakan Glide. Status pesanan juga diperbarui dengan mengubah gambar sesuai dengan status pesanan yang berlaku (Menunggu, Diterima, Selesai). Implementasi ini memberikan pengguna antarmuka yang jelas dan fungsionalitas yang memungkinkan mereka untuk melihat dan mengelola detail pesanan dengan mudah.



Gambar 4.19 UI Detail Pembayaran

```
        override fun onCreate(savedInstanceState: Bundle?) {
            binding = ActivityDetailPesananBinding.inflate(layoutInflater)
            super.onCreate(savedInstanceState)
            setContentView(binding.root)
            val order = intent.getParcelableExtra<OrderModel>("order")
            if (order != null) {
                dataOrder = order
            }
            presenter = DetailPesananPresenter(this, OrderAPI()).apply {
                onAttach(this@DetailPesanan)
            }
            presenter.getPriceById(dataOrder.id.toString(),
dataOrder.id_service)

            val layoutManager = LinearLayoutManager(this)
            binding.rvDetailuser.adapter = adapterDetailuser
            binding.rvDetailuser.layoutManager = layoutManager
            binding.apply {
                tvNamaUser.text = dataOrder.name
                tvAlamatUser.text = dataOrder.address
            }

            binding.cvMetode.setOnClickListener {
                startActivity(Intent(this, MetodePembayaran::class.java ))
            }
            binding.ivBackdetail.setOnClickListener {
                startActivity(Intent(this, HistoryFragment::class.java ))
            }
            binding.cvPayService.setOnClickListener {
                startActivity(Intent(this,
MenungguKonfirmasiPembayaran::class.java).apply {
                    putExtra("order", order)
```

```
                })
            }
            if(dataOrder.status == "Selesai"){
                binding.cvPayService.visibility = View.GONE
            }

            binding.apply {
                if (order != null) {
                    tvNamaUser.text = order.name
                    tvAlamatUser.text = order.address

                    Glide
                        .with(binding.root.context)
                        .load(order.img_url)
                        .into(binding.ivGambardetail)
                    when(order.status){
                        "Menunggu" ->
binding.ivMenunggu.setImageDrawable(ContextCompat.getDrawable(this@DetailP
esanan, R.drawable.ic_progress_indicator))
                        "Diterima" ->
binding.ivDiterima.setImageDrawable(ContextCompat.getDrawable(this@DetailP
esanan, R.drawable.ic_progress_indicator))
                        "Selesai" ->
binding.ivSelesai.setImageDrawable(ContextCompat.getDrawable(this@DetailPe
sanan, R.drawable.ic_progress_indicator))
                    }
                }
            }
        }
```

Gambar 4.20 Kotlin Detail Pembayaran

b. Implementasi Konfirmasi Pembayaran Montir

Dalam kelas `DetailPembayaran`, implementasi tampilan (View) terfokus pada aktivitas ini. Pertama, layout di-inflasi menggunakan `ActivityDetailPembayaranBinding`, yang memungkinkan pengaksesan mudah ke elemen-elemen antarmuka pengguna. Presenter `BayarPresenter` juga diinisialisasi di sini, yang bertanggung jawab untuk mengelola logika bisnis dan berinteraksi dengan data. Pengaturan RecyclerView dan adapter juga terjadi di sini untuk menampilkan detail montir terkait pemesanan. Beberapa aksi pengguna seperti klik tombol "Kirim Resi" dan "Tambah Layanan Tambahan" diimplementasikan dengan memulai aktivitas baru. Selain itu, terdapat interaksi dengan presenter untuk mengirim dan memperbarui data terkait harga dan status pemesanan. Detail dari pesanan seperti nama pengguna, alamat, status, dan gambar ditampilkan melalui elemen-elemen tampilan yang sesuai. Status pesanan juga mempengaruhi ikon status visual. Keseluruhan implementasi ini memungkinkan pengguna untuk melihat dan mengelola detail pembayaran dengan mudah dalam aplikasi MECHA.



Gambar 4.21 UI Konfirmasi Pembayaran Montir

```
    override fun onCreate(savedInstanceState: Bundle?) {
            binding =
ActivityDetailPembayaranBinding.inflate(layoutInflater)
            super.onCreate(savedInstanceState)
            setContentView(binding.root)
            val order = intent.getParcelableExtra<OrderModel>("order")
            if (order != null) {
                dataOrder = order
                if(order.status == "Selesai"){
                    binding.cvKirimResi.visibility = View.GONE
                }else{
                    binding.cvKirimResi.visibility = View.VISIBLE
                }
            }
            presenter = BayarPresenter(this, OrderAPI(), UserAPI()).apply
{
                onAttach(this@DetailPembayaran)
            }

            val layoutManager = LinearLayoutManager(this)
            binding.rvDetailpembayaran.adapter = adapterDetailmontir
            binding.rvDetailpembayaran.layoutManager = layoutManager

            binding.cvKirimResi.setOnClickListener {
                startActivity(Intent(this,
KonfirmasiPembayaran::class.java ))
            }

            binding.cvTambah.setOnClickListener {
                startActivity(Intent(this, LayananTambahan::class.java
).apply {
                    putExtra("order", order)
```

```
                })
            }
            binding.cvKirimResi.setOnClickListener {
                DataPrice.priceList.forEach {

presenter.postPriceByName("${DataPelanggan.id}/${dataOrder.id_service}",
it.description_service, it.price)
                }
                presenter.updateStatus("Diterima", dataOrder.id_service)
            }

presenter.getPriceById(dataOrder.id.toString(),dataOrder.id_service)
            val intent = Intent(Intent.ACTION_VIEW,
Uri.parse(dataOrder.map_url))
            intent.setPackage("com.google.android.apps.maps")
            binding.cvSeeAlamat.setOnClickListener {
                if (intent.resolveActivity(packageManager) != null) {
                    startActivity(intent)
                } else {
                    val webIntent = Intent(Intent.ACTION_VIEW,
Uri.parse(dataOrder.map_url))
                    startActivity(webIntent)
                }

            }
```

Gambar 4.22 Kotlin Konfirmasi Pembayaran Montir

c. Implementasi Bayar Tagihan Montir

Implementasi view pada kelas `BayarTagihan` mengacu pada cara antarmuka pengguna (UI) diatur dan berinteraksi dalam aktivitas pembayaran tagihan. Pertama, dalam metode `onCreate()`, layout dari aktivitas ini diisi dengan menggunakan `ActivityBayarTagihanBinding`, yang memungkinkan pengaksesan elemen UI melalui binding. Nilai tagihan yang diterima dari intent diatur ke TextView `tvTotalTagihan`. Selanjutnya, terdapat implementasi OnClickListener untuk tombol "Salin" (`tvCopy`). Saat tombol ini ditekan, teks dari `tvsourceCopy` disalin ke papan klip, dan pengguna diberi notifikasi dengan Toast bahwa teks telah berhasil disalin. Terakhir, jika `cvSudah` (CardView untuk tombol "Sudah Bayar") ditekan, muncul sebuah dialog konfirmasi pembayaran sukses (`AlertBayarSukses`) dan pengguna akan diarahkan kembali setelah 2 detik. Dengan cara ini, antarmuka pengguna memberikan pengalaman interaktif yang jelas dan nyaman saat melakukan pembayaran tagihan.



Gambar 4.23 UI Bayar Tagihan Montir

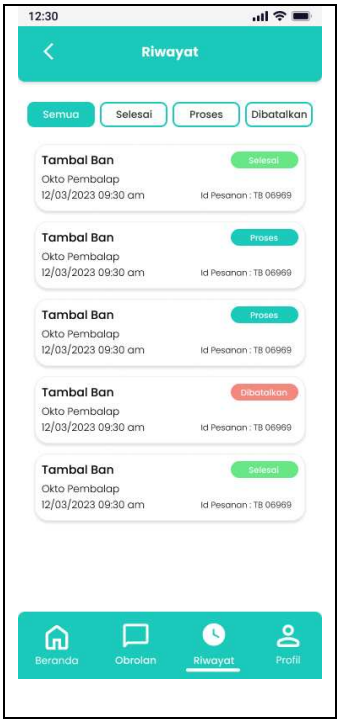
```
            binding.tvCopy.setOnClickListener(object :
View.OnClickListener {
                override fun onClick(v: View?) {
                    val textToCopy = binding.tvsourceCopy.text.toString()
                    val clipboardManager =
getSystemService(Context.CLIPBOARD_SERVICE) as ClipboardManager
                    val clipData = ClipData.newPlainText("Teks yang
disalin", textToCopy)
                    clipboardManager.setPrimaryClip(clipData)
                    Toast.makeText(
                        this@BayarTagihan,
                        "Teks berhasil disalin ke clipboard",
                        Toast.LENGTH_SHORT
                    ).show()
                }
            })

            binding.cvSudah.setOnClickListener {
                AlertBayarSukses().show(supportFragmentManager, "Sukses")
                Handler(Looper.getMainLooper()).postDelayed({
                    onBackPressedDispatcher.onBackPressed()
                },2000)
            }
        }
    }
```

Gambar 4.24 Kotlin Bayar Tagihan

4.3.6 Riwayat Pesanan

Dalam komponen antarmuka pengguna (View) aplikasi MECHA, tampilan riwayat pesanan dirancang untuk memberikan pengguna informasi yang jelas mengenai sejarah pemesanan layanan perbaikan kendaraan mereka. Tampilan ini akan menampilkan daftar pesanan dalam format yang terstruktur, mencakup detail seperti nomor pemesanan, tanggal pemesanan, jenis layanan yang dipilih, dan status pemesanan. Pengguna dapat dengan mudah melihat riwayat pemesanan mereka dalam urutan kronologis, yang memungkinkan mereka untuk melacak dan memahami proses layanan yang telah mereka lakukan. Dengan adanya tampilan riwayat pesanan, pengguna dapat merasa lebih terinformasi dan memiliki akses cepat ke informasi yang mereka butuhkan terkait pemesanan layanan perbaikan kendaraan. Implementasi view pada kelas `HistoryFragment` bertujuan untuk mengatur tampilan dan interaksi antarmuka pengguna di layar riwayat pesanan. Pada tahap awal (`onCreateView`), tata letak XML `FragmentHistoryBinding` dihubungkan ke fragmen ini. Selanjutnya, dalam metode `onViewCreated`, presenter `HistoryPresenter` diinisialisasi dengan memanggil konstruktor dan method `onAttach`. Terdapat juga konfigurasi untuk RecyclerView (`rvHistory`) yang digunakan untuk menampilkan daftar riwayat pesanan. Adapter `HistoryListAdapter` diinisialisasi dan diset pada RecyclerView, dan juga pengaturan tata letak manajer linear. Presenter kemudian memanggil metode `getHistory` dengan ID pengguna untuk mengambil data riwayat. Selain itu, terdapat empat card view (`cvSemuaHistory`, `cvProsesHistory`, `cvDibatalkanHistory`, `cvSelesaiHistory`, dan `cvMenungguHistory`) yang mewakili filter status pesanan. Saat salah satu card view ditekan, data riwayat akan difilter berdasarkan status yang sesuai. Hasil filtrasi akan diperbarui pada RecyclerView menggunakan `adapterHistory.submitList()`. Selain itu, tata letak dan warna card view akan diperbarui untuk menunjukkan filter yang aktif. Keseluruhan implementasi ini bertujuan untuk menyajikan riwayat pesanan secara terstruktur dan memungkinkan pengguna untuk memilah data berdasarkan statusnya. Ini meningkatkan pengalaman pengguna dengan memberikan cara yang jelas untuk menavigasi dan menganalisis riwayat pesanan mereka.



Gambar 4.25 UI Riwayat Pesanan

```
  override fun onCreateView(
            inflater: LayoutInflater, container: ViewGroup?,
            savedInstanceState: Bundle?
        ): View {
            binding = FragmentHistoryBinding.inflate(layoutInflater,
container, false)
            return binding.root
        }

        override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
            super.onViewCreated(view, savedInstanceState)
            presenter = HistoryPresenter(this, OrderAPI()).apply {
                onAttach(this@HistoryFragment)
            }

            val layoutManager = LinearLayoutManager(activity)
            binding.rvHistory.adapter = adapterHistory
            binding.rvHistory.layoutManager = layoutManager
            adapterHistory.setOnclickItem(rvClickListener)
            presenter.getHistory(DataToken.idUser)
```

Gambar 4.26 Kotlin Riwayat Pesanan

4.4 Implementasi Presenter

Komponen Presenter dalam aplikasi MECHA mengelola logika. Masing-masing fitur dalam aplikasi memiliki presenter yang mengatur alur kerja dan memproses data sesuai

dengan kebutuhan. Presenter menerima masukan dari view dan memperbarui tampilan serta mengambil data dari Model.

4.4.1 Register

Dalam bagian Presenter, Presenter akan menerima input ini dan memvalidasi data tersebut sebelum mengirimkannya ke Model. Presenter juga dapat melakukan langkah-langkah tambahan seperti memeriksa ketersediaan alamat email dalam basis data atau menerapkan kebijakan validasi tertentu. Setelah data valid, Presenter akan menginstruksikan Model untuk memasukkan data pengguna baru ke dalam basis data. Jika proses pendaftaran berhasil,. Presenter Register ini bertanggung jawab untuk memediasi antara tampilan dan model dalam proses registrasi pengguna. Presenter ini memvalidasi data yang dimasukkan oleh pengguna, termasuk email, password, nomor telepon, dan konfirmasi kata sandi. Pertama, presenter memastikan bahwa format email sesuai dengan pola yang benar dan memberikan umpan balik sesuai. Selanjutnya, presenter memvalidasi kata sandi untuk memastikan memiliki minimal 8 karakter dengan kombinasi huruf dan angka. Jika validasi berhasil, presenter memberikan umpan balik ke tampilan. Presenter juga memverifikasi panjang nomor telepon dan memberikan pesan kesalahan jika format tidak sesuai. Selain itu, presenter memastikan konfirmasi kata sandi sesuai dengan kata sandi yang dimasukkan sebelumnya. Jika terdapat kesalahan dalam validasi, presenter memberikan pesan kesalahan yang sesuai. Presenter juga memeriksa ketersediaan email untuk memastikan tidak ada duplikasi. Akhirnya, presenter memanggil metode untuk mendaftarkan pengguna melalui API dan memberikan umpan balik ke tampilan berdasarkan hasilnya. Presenter ini memanfaatkan manajemen coroutine untuk menjalankan tugas secara asinkronus dan memastikan tanggung jawabnya terhadap proses ini.

```
            fun validateEmail(email: String): Boolean {
                val isEmailValid = email.contains("^[\\w-\\.]+@([\\w-
]+\\.)+[\\w-]{2,4}\$".toRegex())
                if (!isEmailValid) {
                    view?.onError(1, "Format email tidak sesuai")
                } else {
                    view?.onErrorSuccess(1,"")
                }
                return isEmailValid
            }

            fun validatePassword (password: String): Boolean {
                val isPasswordValid = password.contains ("^(?=.*[0-
9])(?=.*[a-zA-Z])[a-zA-Z0-9]{8,}\$".toRegex())
```

```
                if (isPasswordValid) {
                    view.onErrorSuccess(2,"")
                } else {
                    view.onError(3,"Password minimal 8 huruf dan kata
sandi harus terdiri dari angka dan huruf")
                }
                return isPasswordValid
            }

            fun validateTelephone (phone: String): Boolean {
                this.isTelephoneValid = phone.length in 9..13
                if (isTelephoneValid) {
                    view.onErrorSuccess(6,"")
                } else {
                    view.onError(4,"Nomor telepon terdiri dari 10-12
angka")
                }
                return isTelephoneValid
            }

            fun validateRepassword (password: String, repassword:
String) : Boolean {
                isValidateRepassword = repassword.equals(password)
                if (isValidateRepassword) {
                    view.onErrorSuccess(7, "")
                } else {
                    view.onError(5,"Password tidak sama")

                }
                return isValidateRepassword
            }

            fun emailAvailable (email: String) : Boolean {
                isEmailAvailable = email == "ghazygg@gmail.com"
                if (email == "ghazygg@gmail.com") {
                    view.onErrorAvailable(8,"Email Sudah Dipakai")
                }
                return isEmailAvailable
            }
        fun regisUser(nama: String, email: String, notelp: String,
password: String){
            api.regisUser(nama,email,notelp,password) {
                scope.launch {
                    when(it){
                        is ResponseStatus.Success ->
view.onSuccesRegister(it.data)
                        is ResponseStatus.Failed ->
view.onErrorSignup(it.message)
                    }
                }
            }
        }
    }
```

Gambar 4.27 Presenter Register

4.4.2 Login

Dalam konteks bagian Presenter, implementasi login merupakan komponen yang mengatur proses otentikasi pengguna ke dalam aplikasi. Presenter login bertanggung jawab atas penerimaan input berupa informasi login dari pengguna, seperti nama pengguna dan kata sandi. Setelah menerima input, Presenter akan memvalidasi informasi tersebut dengan menggunakan Model yang berhubungan dengan data pengguna terdaftar. Jika informasi yang dimasukkan valid, Presenter akan memberikan akses kepada pengguna ke fitur-fitur tertentu dalam aplikasi, seperti tampilan utama atau riwayat pemesanan. Namun, jika informasi tidak valid, Presenter akan memberikan umpan balik kepada pengguna berupa pesan kesalahan yang sesuai. Dengan mengatur alur login secara efektif, Presenter memastikan bahwa hanya pengguna yang sah yang dapat mengakses fungsionalitas yang ada dalam aplikasi MECHA. Presenter `MainActivityPresenter` bertanggung jawab untuk mengelola logika bisnis terkait antarmuka pengguna utama dalam aplikasi. Pertama, presenter ini memvalidasi email dan password yang dimasukkan pengguna dengan menggunakan metode `validateEmail` dan `validatePassword`. Pada validasi, presenter memeriksa apakah format email sesuai dan apakah password memiliki panjang minimal dan mengandung kombinasi huruf dan angka yang benar. Selain itu, presenter juga mengatur perilaku UI terkait error dan pesan validasi. Selain validasi, presenter mengelola proses login pengguna melalui metode `loginUser`. Di dalamnya, presenter berkomunikasi dengan API `UserAPI` untuk melakukan proses autentikasi. Hasil dari proses ini akan memicu respon yang akan diteruskan kembali ke antarmuka pengguna melalui `view`. Presenter menggunakan `CoroutineScope` untuk mengelola proses asinkron dan menangani kemungkinan exception yang terjadi selama proses. Terakhir, presenter dapat melekatkan atau memisahkan antarmuka pengguna melalui metode `onAttach`. Namun, dalam kode yang diberikan, implementasi dari metode `onAttach` terlihat belum lengkap, sehingga tidak memiliki pengaruh pada fungsionalitas presenter.

```
         fun validateEmail(email: String): Boolean {
             //validasi format email
             isEmailValid = email.contains("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-
]{2,4}\$".toRegex())

             if (isEmailValid) {
                 view.onErrorEmpty(1)
             }else {
                 view.onError(2, "Format email yang anda masukkan salah.")
             }
```

```
            return isEmailValid
        }
        fun validatePassword(password: String): Boolean {
            //validasi email lebih dari 7 karakter
            isPasswordValid = password.contains ("^(?=.*[0-9])(?=.*[a-zA-
Z])[a-zA-Z0-9]{8,}\$".toRegex())


            if (isPasswordValid) {
                view.onErrorEmpty(3)
            }else {
                view.onError(4, "Password minimal 8 huruf dan kata sandi
harus terdiri dari angka dan huruf.")
            }
            return isPasswordValid
        }

        fun loginUser(email: String, password: String){
            api.loginUser(email, password){
                scope.launch {
                    when(it){
                        is            ResponseStatus.Success            ->
view.onSuccesLogin(it.data)
                        is ResponseStatus.Failed -> view.onErrorLogin()
                    }
                }
            }
        }
    }
```

Gambar 4.28 Presenter Login

4.4.3 Pesan Layanan

Dalam sub-sub bab implementasi presenter pesan layanan, fokus utamanya adalah pada kelas `OrderPresenter`. Presenter ini bertindak sebagai perantara antara tampilan "Konfirmasi Order" (ConfirmationOrder) dan model data, khususnya API Order. Presenter ini menyediakan sejumlah metode untuk memproses pesanan yang dibuat oleh pengguna. Misalnya, fungsi `postOrder` memungkinkan presenter untuk mengirimkan data pesanan seperti jenis layanan, status, alamat, dan URL peta. Ketika operasi ini selesai, presenter menanggapi hasil dari API Order, memastikan bahwa jika pesanan berhasil terkirim, tampilan akan menerima pemberitahuan melalui metode `onSuccesOrder`. Sebaliknya, jika terjadi kegagalan dalam pengiriman pesanan, metode `onErrorOrder` akan diaktifkan untuk menangani situasi tersebut. Dengan demikian, presenter memainkan peran kunci dalam memastikan bahwa interaksi pengguna dengan antarmuka "Konfirmasi Order" berjalan lancar dan memadukan data yang diperlukan dengan API Order.

a. Implementasi Presenter Pesan Layanan

Presenter Order yang telah diimplementasikan adalah komponen kunci dalam arsitektur MV (Model-View) dari aplikasi MECHA. Presenter ini bertanggung jawab untuk memproses logika bisnis terkait pemesanan layanan, interaksi dengan API Order, dan memediasi komunikasi antara tampilan (View) dan model data. Presenter ini memanfaatkan Kotlin Coroutines untuk menangani tugas-tugas jaringan secara asinkron.

Presenter ini menyediakan serangkaian fungsi, masing-masing menangani jenis permintaan yang berbeda terhadap API Order. Misalnya, fungsi `postOrder` mengirimkan pesanan baru dengan parameter yang diteruskan dari tampilan, dan mengelola hasil respons dari API. Jika respons dari API adalah sukses, maka presenter akan memanggil metode `onSuccessOrder` di tampilan. Sebaliknya, jika respons dari API adalah gagal, maka metode `onErrorOrder` akan dipanggil untuk menangani situasi tersebut.

Fungsi-fungsi serupa juga ada untuk kasus-kasus lain seperti memposting riwayat pesanan (`postHistory`), memposting harga pesanan (`postPriceOrder` dan `postPriceHistory`), serta mengambil pesanan berdasarkan ID (`getOrderById`). Semua fungsi ini menggunakan CoroutineScope untuk memastikan operasi-operasi jaringan berjalan secara asinkron dan tidak memblokir antarmuka pengguna. Dengan cara ini, aplikasi MECHA dapat memberikan responsifitas tinggi kepada pengguna selama interaksi dengan API Order.

```
fun postOrder(service: String, status: String, address: String, mapurl: String){
        api.postOrder(service, status, address, mapurl){
            scope.launch {
                when(it){
                    is              ResponseStatus.Success              -> view.onSuccesOrder(it.data)
                    is               ResponseStatus.Failed              -> view.onErrorOrder(it.message)
                }
            }
        }
    }

    fun postHistory(name: String, service: String, status: String, address: String, mapurl: String, id_service: String){
        api.postHistory(name, service, status, address, mapurl, id_service){
            scope.launch{
                when(it){
```

```
                    is              ResponseStatus.Success              ->
view.onSuccesHistory(it.data)
                    is               ResponseStatus.Failed              ->
view.onErrorhistory(it.message)
                }
            }
        }
    }

    fun postPriceOrder(id_service: String, desc: String, price: String){
        api.postPriceOrder(id_service,desc,price){
            scope.launch {
                when(it){
                    is              ResponseStatus.Success              ->
view.onSuccessPrice(it.data)
                    is               ResponseStatus.Failed              ->
view.onErrorPrice(it.message)
                }
            }
        }
    }

    fun  postPriceHistory(id_service:  String,  desc:  String,  price:
String){
        api.postPriceHistory(id_service, desc, price){
            when(it){
                is ResponseStatus.Success -> view.onSuccessPrice(it.data)
                is ResponseStatus.Failed -> view.onErrorPrice(it.message)
            }
        }
    }

    fun getOrderById(id: String){
        api.getOrderById(id){
            scope.launch {
                when(it){
                    is              ResponseStatus.Success              ->
view.onSuccesGetOrder(it.data)
                    is               ResponseStatus.Failed              ->
view.onErrorgetOrder(it.message)
                }
            }
        }
    }
}
```

Gambar 4.29 Presenter Pesan Layanan

b. Implementasi Ambil Pesan Presenter

Implementasi presenter `AmbilPesanPresenter` bertujuan untuk memfasilitasi interaksi antara tampilan (View) dan model (API). Presenter ini memiliki tanggung jawab untuk memproses logika bisnis terkait ambil pesan dalam aplikasi MECHA. Pertama, presenter ini memiliki fungsi `onAttach` yang seharusnya menerima sebuah view dari tipe

`AmbilPesanContract`. Kemudian, terdapat fungsi `getOrder` yang memanggil API `getAllOrder`. Hasil dari pemanggilan API akan dievaluasi, dan berdasarkan responsnya, presenter akan memberi tahu view apakah ambil pesan berhasil atau gagal melalui pemanggilan fungsi `onSuccessGet` atau `onFailedGet` di dalam scope Coroutine. Keseluruhannya, presenter ini bertanggung jawab untuk memisahkan logika bisnis dari tampilan, memastikan bahwa view hanya menangani tampilan data dan respons yang diberikan oleh presenter.

```
    class AmbilPesanPresenter(
        private val view: AmbilPesanContract,
        private val api: OrderAPI,
        uiContext: CoroutineContext = Dispatchers.Main
    ) {
        private val supervisorJob: Job = SupervisorJob()
        private val scope = CoroutineScope(supervisorJob + uiContext)

        fun onAttach(view: AmbilPesanContract){
            this.view
        }

        fun getOrder(){
            api.getAllOrder {
                scope.launch {
                    when(it){
                        is ResponseStatus.Success ->
view.onSuccessGet(it.data)
                        is ResponseStatus.Failed ->
view.onFailedGet(it.message)
                    }
                }
            }
        }
    }
```

Gambar 4.30 Presenter Ambil Pesan

4.4.4 Pembayaran

a. Implementasi Presenter pembayaran

Presenter `DetailPesananPresenter` bertanggung jawab untuk mengatur logika bisnis terkait tampilan "Detail Pesanan" dalam aplikasi MECHA. Dalam implementasi ini, presenter menghubungkan tampilan (`view`) dengan sumber daya data dari API layanan pesanan (`api`). Presenter ini juga memanfaatkan coroutines untuk melakukan tugas-tugas jaringan dan pemrosesan data secara asinkron.

Metode `getPriceById` digunakan untuk mengambil harga berdasarkan ID pesanan dan ID layanan. Ketika pemanggilan ke API selesai, hasil respons akan dievaluasi. Jika

respons adalah sukses (`ResponseStatus.Success`), presenter akan memberi tahu tampilan untuk menangani data dengan memanggil `view.onSuccesGetPrice(it.data)`. Namun, jika terjadi kegagalan, pesan error akan diteruskan ke tampilan melalui `view.onErrorGetPrice(it.message)`.

```
        fun getPriceById(id: String, id_service: String){
            api.getPriceId(id, id_service){
                when(it){
                    is ResponseStatus.Success ->
view.onSuccesGetPrice(it.data)
                    is ResponseStatus.Failed ->
view.onErrorGetPrice(it.message)
                }
            }
        }

        fun getHistory(id: String){
            api.getHistory(id){
                scope.launch {
                    when(it){
                        is ResponseStatus.Success ->
view.onSuccesGetStatus(it.data)
                        is ResponseStatus.Failed ->
view.onErrorGetStatus(it.message)
                    }
                }
            }
        }
    }
```

Gambar 4.31 Presenter Pembayaran

b. Implementasi Konfirmasi Pembayaran Montir

Implementasi Presenter pada kelas `BayarPresenter` bertujuan untuk memisahkan logika bisnis dari tampilan dalam aplikasi MECHA. Presenter ini bekerja sebagai perantara antara antarmuka pengguna dan sumber data atau API yang diperlukan. Pertama, presenter menerima referensi tampilan (`view`) dan akses ke API untuk pesanan dan pengguna. Presenter menggunakan struktur Coroutine untuk menangani operasi berbasis jaringan atau IO yang mungkin membutuhkan waktu. SupervisorJob dan CoroutineScope digunakan untuk memastikan bahwa pekerjaan yang dimulai dapat dipantau dan dihentikan jika diperlukan. Metode `onAttach` yang didefinisikan, sepertinya tidak melakukan apa pun selain memetakan referensi tampilan, dan sepertinya ada kesalahan karena penggunaan syntax yang tidak sesuai. Selanjutnya, terdapat beberapa fungsi yang mengelola operasi terkait harga, pengguna, dan pembaruan status.

Misalnya, fungsi `postPriceByName` mengirimkan data harga berdasarkan nama layanan ke API. Hasil dari operasi ini kemudian akan diproses, dan berdasarkan statusnya, tampilan akan diberi tahu tentang keberhasilan atau kesalahan operasi. Fungsi-fungsi lainnya seperti `postPrice`, `getPriceById`, `getAllUser`, dan `updateStatus` bekerja dengan prinsip yang serupa. Mereka memanggil API yang relevan dan menangani respons yang diterima. Jika operasi berhasil, hasilnya akan diteruskan ke tampilan melalui metode yang sesuai di kontrak antarmuka (`BayarContract`). Dengan menggunakan presenter seperti ini, aplikasi MECHA dapat mempertahankan prinsip desain arsitektural yang baik dan memungkinkan pengujian yang lebih baik dari komponen-komponen tertentu. Presenter memainkan peran penting dalam memisahkan logika aplikasi dari tampilan, sehingga membuat aplikasi lebih mudah dipelihara dan diperbarui..

```
        fun postPriceByName(id_service: String, desc: String, price:
String){
            api.postPriceByName(id_service, desc, price){
                scope.launch {
                    when(it){
                        is ResponseStatus.Success ->
view.onSuccesPrice(it.data)
                        is ResponseStatus.Failed ->
view.onErrorPrice(it.message)
                    }
                }
            }
        }

        fun postPrice(id_service: String, desc: String, price: String){
            api.postPriceById(id_service, desc, price){
                scope.launch {
                    when(it){
                        is ResponseStatus.Success ->
view.onSuccesPrice(it.data)
                        is ResponseStatus.Failed ->
view.onErrorPrice(it.message)
                    }
                }
            }
        }

        fun getPriceById(id: String, id_service: String){
            api.getPriceId(id, id_service){
                when(it){
                    is ResponseStatus.Success ->
view.onSuccesGetPrice(it.data)
                    is ResponseStatus.Failed ->
view.onErrorGetPrice(it.message)
                }
            }
```

```
        }

        fun getAllUser(){
            apiUser.getAllUser {
                scope.launch {
                    when(it){
                        is ResponseStatus.Success ->
view.onSuccesuser(it.data)
                        is ResponseStatus.Failed ->
view.onError(it.message)
                    }
                }
            }
        }

        fun updateStatus(status: String, id_service: String){
            api.updateStatus(id_service, status){
                scope.launch{
                    when(it){
                        is ResponseStatus.Success ->
view.onSuccesUpdate(it.data)
                        is ResponseStatus.Failed ->
view.onError(it.message)
                    }
                }
            }
        }
    }
```

Gambar 4.32 Presenter Pembayaran Montir

4.4.5 Riwayat Pesanan

Presenter Riwayat Pesanan adalah komponen kunci dalam struktur aplikasi MECHA yang bertanggung jawab atas mengatur logika bisnis terkait riwayat pesanan pengguna. Fungsinya meliputi pengambilan data riwayat pesanan dari Model, memprosesnya, dan mempersiapkannya untuk ditampilkan di antarmuka pengguna. Presenter ini juga berinteraksi dengan tampilan (View) terkait, memastikan bahwa informasi riwayat pesanan disajikan dengan jelas dan sesuai dengan kebutuhan pengguna. Presenter Riwayat Pesanan adalah komponen kunci dalam struktur aplikasi MECHA yang bertanggung jawab atas mengatur logika bisnis terkait riwayat pesanan pengguna. Fungsinya meliputi pengambilan data riwayat pesanan dari Model, memprosesnya, dan mempersiapkannya untuk ditampilkan di antarmuka pengguna. Presenter ini juga berinteraksi dengan tampilan (View) terkait, memastikan bahwa informasi riwayat pesanan disajikan dengan jelas dan sesuai dengan kebutuhan pengguna.

```
    fun getHistory(id: String){
            api.getHistory(id) {
                scope.launch {
                    when(it){
```

```
                    isResponseStatus.Success>view.onSucces(it.data)
                    isResponseStatus.Failed->view.onFailed(it.message)
                     }
                }
```

Gambar 4.33 Presenter Riwayat Pesanan

4.5 Pengujian Menggunakan Dua Mitra

Pada bagian ini, akan dijelaskan bagaimana jika ada 2 mitra dan hanya ada satu pesanan maka bagaimana penanganan yang dilakukan aplikasi.

```
            binding.btnAccept.setOnClickListener {
                if (order != null) {
                    presenter.deleteOrder(order.id.toString())
                }
            }
```

Gambar 4. 34 Function Delete Ambil Pesan

Pada Gambar 4.32 terdapat btnAccept pada halaman ambil pesan yang mana akan melakukan delete order saat ada montir yang telah mengambil pemesanan, berdasarkan penjelasan diatas sudah menanggulangi masalah jika terjadinya pemesanan oleh dua orang mitra atau lebih.

4.6 Arsitektur Database

Pada bagian ini, akan dijelaskan tentang bagaimana struktur dan komponen dari basis data aplikasi MECHA dirancang dan diimplementasikan. Basis data PostgreSQL dipilih sebagai sistem manajemen basis data (DBMS) karena kemampuannya yang kuat dalam mengelola data yang berkaitan dengan layanan perbaikan kendaraan. Pengelolaan basis data dilakukan melalui DBeaver, sementara interaksi dengan basis data melalui aplikasi diatur oleh Framework Adonis yang berbasis JavaScript.

4.6.1 Struktur Basis Data

Struktur basis data aplikasi MECHA terdiri dari beberapa tabel utama yang mencakup entitas-entitas kunci dalam konteks layanan perbaikan kendaraan.

a. Tabel User

Tabel user digunakan untuk menyimpan informasi tentang pengguna. Tabel user dapat dilihat pada Tabel 4.1, peran (role) di sini mengacu pada perbedaan antara pengguna umum dan montir saat melakukan proses registrasi dalam aplikasi Mecha.

Tabel 4.1 Tabel User

No	Nama	Tipe	Keterangan
1	Id	Int	Id table (primary key)
2	name	Varchar	
3	No Telepon	Varchar	
4	Email	Varchar	
5	Password	Varchar	
6	Role	Int	
7	Img_Url	String	
8	No Rek	String	
9	Tagihan	String	

b. Tabel Mekanik

Tabel mekanik digunakan untuk menyimpan informasi tentang pengguna. Tabel mekanik dapat dilihat pada Tabel 4.2, peran (role) di sini mengacu pada perbedaan antara pengguna umum dan montir saat melakukan proses registrasi dalam aplikasi Mecha.

Tabel 4. 2 Tabel Mekanik

No	Nama	Tipe	Keterangan
1	Id	Int	Id table (primary key)
2	name	Varchar	
3	No Telepon	Varchar	
4	Email	Varchar	
5	Password	Varchar	
6	Role	Int	
7	No Rek	String	

c. Tabel Order

Tabel order adalah tabel yang memuat informasi terkait order yang dilakukan oleh user, yang nantinya akan diterima oleh mekanik. Order menggunakan 3 method yaitu GET, POST, DELETE. GET akan digunakan oleh montir, POST akan digunakan oleh user, DELETE akan digunakan oleh montir

Tabel 4.3 Tabel Order

No	Nama	Tipe	Keterangan
1	Id	Int	Id table (primary key)
2	User_id	Int	Foreign key ke table user
3	Id_service	Varchar	
4	Nama Layanan	Varchar	
5	Status	Varchar	
6	Harga Layanan	Varchar	
7	Alamat User	Varchar	
8	Lokasi User	Varchar	

d. Tabel History

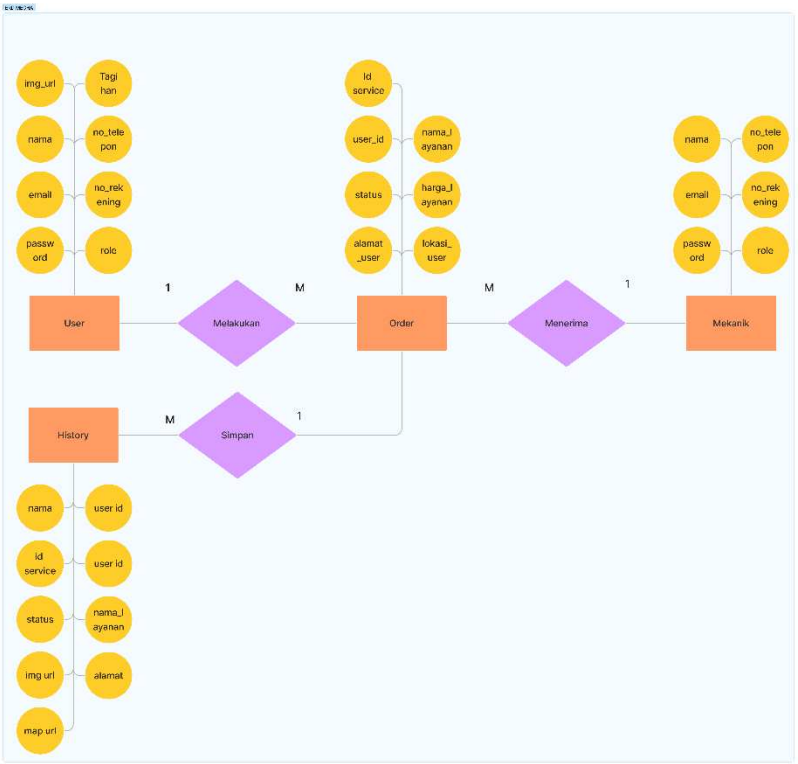
Tabel history Adalah tabel yang memuat informasi terkait history pemesanan yang dilakukan oleh user

Tabel 4.4 Tabel History

No	Nama	Tipe	Keterangan
1	Id	Int	Id table (Primary Key)
2	Nama	Varchar	
3	Id service	Varchar	
4	User id	Int	Foreign Key
5	Nama Layanan	Varchar	
6	Status	String	
7	Alamat	Varchar	
8	Img url	Varchar	
9	Map Url	Varchar	

4.6.2 Relasi Antar Tabel

Dalam basis data aplikasi MECHA, relasi antar tabel dibentuk untuk memungkinkan pengelolaan data yang lebih efisien dan konsisten.



Gambar 4.35 Relasi Antar Tabel

Pada Gambar, rancangan ERD menggambarkan hubungan antar tabel dalam sebuah basis data yang digunakan pada aplikasi MECHA. Selain itu terdapat 4 entitas dengan atributnya

masing-masing. Entitas-entitas tersebut yaitu user, order, mekanik,history. Setiap entitas memiliki relasi dengan entitas lainya, penggunaan role diantara 2 user berguna untuk membedakan saat melakukan pendaftaran agar pendaftaran lebih mudah dilakukan. Dalam aplikasi MECHA, hubungan antara pengguna yang melakukan pemesanan (User) dengan pemesanan (Order) dapat dikategorikan sebagai One-to-Many (1:M). Hal ini disebabkan oleh struktur alur logika bisnis di mana setiap pengguna memiliki kemampuan untuk melakukan banyak pemesanan, namun setiap pemesanan hanya dapat terkait dengan satu pengguna. Hal ini sesuai dengan kebutuhan aplikasi, di mana pengguna dapat melakukan berbagai pemesanan layanan perbaikan kendaraan sesuai dengan kebutuhan mereka. Oleh karena itu, dalam konteks ini, hubungan ini dapat dijelaskan sebagai One-to-Many, memungkinkan pengguna untuk memiliki sejumlah pemesanan sesuai dengan preferensi dan kebutuhan individual mereka. Selanjutnya hubungan antara Mekanik yang melakukan pengambilan order dan pemesanan (Order) dapat dijelaskan sebagai One-to-Many. Hal ini karena seorang Mekanik dapat mengambil banyak pesanan dari berbagai pelanggan, namun setiap pesanan hanya dapat diambil oleh satu Mekanik. Dengan kata lain, satu sisi hubungan (Mekanik) memiliki banyak entitas terkait di sisi lain (Order), tetapi sebaliknya, setiap pesanan hanya akan diambil oleh satu Mekanik. Dengan struktur hubungan ini, sistem dapat melacak dengan akurat siapa yang bertanggung jawab atas setiap pesanan, memungkinkan untuk manajemen dan pelacakan yang efisien. Selajutnya, penyimpanan pesanan dalam riwayat pada aplikasi MECHA dapat dijelaskan sebagai hubungan One-to-Many. Setiap pengguna dalam aplikasi memiliki kemampuan untuk melakukan banyak pemesanan kendaraan seiring berjalannya waktu. Dengan kata lain, satu pengguna dapat memiliki beberapa pesanan terkait dengan akun mereka. Namun, setiap pemesanan hanya dapat terkait dengan satu pengguna secara spesifik. Oleh karena itu, struktur penyimpanan order dalam riwayat aplikasi MECHA mencerminkan hubungan One-to-Many, di mana setiap pengguna memiliki banyak pesanan tetapi setiap pesanan hanya terkait dengan satu pengguna. Hal ini memungkinkan pengguna untuk dengan mudah melacak dan mengelola riwayat pesanan mereka melalui aplikasi MECHA.

4.6.3 Implementasi API Dalam MECHA

Dalam sub-sub bab ini, kita akan mengulas implementasi API dalam aplikasi MECHA yang memungkinkan komunikasi antara frontend dan backend. Tiga API kunci yang akan

dibahas adalah NetworkAPI, OrderAPI, dan UserAPI. Kami akan merinci bagaimana setiap API berfungsi dan memberikan contoh hasil codingan untuk setiap satu di antaranya.

a. NetworkAPI

Kelas `NetworkClient` merupakan bagian kunci dari infrastruktur jaringan dalam aplikasi MECHA. Kelas ini menyediakan metode-metode untuk membuat berbagai jenis permintaan HTTP ke server basis data menggunakan OkHttp, serta menangani autentikasi melalui header token Bearer. Pertama, dalam `NetworkClient` terdapat definisi konstan `BASE_URL` yang menyimpan URL dasar dari API server MECHA. Selanjutnya, ada `headerInterceptor` yang merupakan objek `Interceptor` untuk menangani header dalam setiap permintaan. Di dalamnya, `Content-Type` ditetapkan sebagai `application/json`. Metode `client` adalah lazy property yang menginisialisasi dan mengonfigurasi klien OkHttp. Ini termasuk penambahan `headerInterceptor` dan logging interceptor yang memungkinkan debug jika aplikasi berada dalam mode Debug. Selanjutnya, ada serangkaian metode yang menghasilkan `Request` dengan berbagai jenis metode HTTP (GET, POST, PUT, DELETE) untuk berbagai end point. Misalnya, `requestBuilder`, `getWithBearerToken`, `requestById`, dll. Masing-masing metode ini membangun dan mengembalikan objek `Request` sesuai dengan endpoint yang ditentukan. Misalnya, `requestLogin` adalah metode untuk mengirim permintaan login dengan menyediakan alamat email dan kata sandi pengguna. Ini menghasilkan permintaan POST dengan badan formulir yang sesuai. Ada juga metode `updateTagihan`, `updateReqName`, `updateRequest`, `requestPrice`, dan `deleteRequest` yang masing-masing digunakan untuk memperbarui tagihan, nama, status, harga, dan menghapus permintaan. Penting untuk dicatat bahwa metode-metode ini memanfaatkan OkHttp dan membangun objek `Request` dengan berbagai atribut seperti URL, method, header, dan body sesuai dengan kebutuhan permintaan tertentu. Seluruh kelas `NetworkClient` ini bertindak sebagai antarmuka antara aplikasi MECHA dan server basis data, memungkinkan aplikasi untuk berkomunikasi dengan server secara efisien melalui API yang disediakan.:

```
    private const val BASE_URL = "https://c645-103-95-7-66.ngrok-
free.app/api"

    fun requestLogin(endpoint: String,email: String, password: String,
method: METHOD = METHOD.POST, jsonBody: String? = null): Request {
                val requestBody = FormBody.Builder()
                    .add("email", email)
```

```
                    .add("password", password)
                    .build()
                val request = Request.Builder()
                    .url("$BASE_URL$endpoint")
                    .post(requestBody)

                if(jsonBody != null){
                    request.method(method.name, jsonBody.toRequestBody())
                }

                return request.build()
            }

            fun requestResgis(endpoint: String, nama: String, no_telp:
String, email: String, password: String, method: METHOD = METHOD.POST,
jsonBody: String? = null): Request{
                val requestBody = FormBody.Builder()
                    .add("name", nama)
                    .add("no_telp", no_telp)
                    .add("email", email)
                    .add("password", password)
                    .build()
                val request = Request.Builder()
                    .url("$BASE_URL$endpoint")
                    .post(requestBody)

                if (jsonBody != null){
                    request.method(method.name, jsonBody.toRequestBody())
                }

                return request.build()
            }

       fun requestOrder(endpoint: String, token: String, service: String,
status: String, address: String, mapUrl: String, method: METHOD =
METHOD.POST, jsonBody: String? = null): Request{
                val requestBody = FormBody.Builder()
                    .add("name_service", service)
                    .add("status", status)
                    .add("address", address)
                    .add("map_url", mapUrl)
                    .build()
                val request = Request.Builder()
                    .url("$BASE_URL$endpoint")
                    .header("Authorization", "Bearer $token")
                    .post(requestBody)

                if (jsonBody != null)
                    request.method(method.name, jsonBody.toRequestBody())

                return request.build()
            }

            fun requestHistory(endpoint: String, token: String, name:
String, service: String, status: String, address: String, mapUrl: String,
id_service: String, method: METHOD = METHOD.POST, jsonBody: String? =
null): Request{
                val requestBody = FormBody.Builder()
                    .add("name", name)
                    .add("name_service", service)
```

```
                .add("status", status)
                .add("address", address)
                .add("map_url", mapUrl)
                .add("id_service", id_service)
                .build()
            val request = Request.Builder()
                .url("$BASE_URL$endpoint")
                .header("Authorization", "Bearer $token")
                .post(requestBody)
            if (jsonBody != null)
                request.method(method.name, jsonBody.toRequestBody())
            return request.build()}
```

Gambar 4.36 Network API

b. OrderAPI

Kelas `OrderAPI` adalah bagian dari sistem yang bertanggung jawab untuk melakukan berbagai operasi terkait pesanan, riwayat, dan harga. Terdapat sejumlah endpoint yang berbeda yang digunakan untuk berinteraksi dengan server. Misalnya, `orderEndpoint` digunakan untuk mendapatkan semua pesanan, `historyEndpoint` digunakan untuk mendapatkan riwayat, dan sebagainya. Metode `getAllOrder` mengirim permintaan GET untuk mendapatkan semua pesanan. Hasil dari permintaan ini kemudian diolah dan diteruskan melalui pemanggilan `onResponse`. Metode `getOrderById` mengirim permintaan GET untuk mendapatkan pesanan berdasarkan ID tertentu. Sama seperti sebelumnya, hasilnya diolah dan diteruskan melalui pemanggilan `onResponse`. Metode `getHistory` digunakan untuk mendapatkan riwayat berdasarkan ID pengguna. Permintaan GET dikirimkan dan hasilnya diolah. Metode `getPrice` digunakan untuk mendapatkan harga berdasarkan ID layanan. Permintaan GET dikirimkan dan hasilnya diolah. Selanjutnya, terdapat serangkaian metode yang digunakan untuk mengirimkan data pesanan, riwayat, dan harga baru ke server. Misalnya, `postOrder` digunakan untuk membuat pesanan baru dengan mengirimkan data seperti jenis layanan, status, alamat, dan URL peta. Hasil dari permintaan ini kemudian diolah dan diteruskan melalui pemanggilan `onResponse`. Selain itu, terdapat juga metode-metode untuk menghapus pesanan, dan memperbarui status, nama, dan tagihan. Akhirnya, terdapat metode `getToken` yang digunakan untuk mendapatkan token dari penyimpanan bersama aplikasi. Setiap metode memanfaatkan klien jaringan (`NetworkClient`) untuk mengirimkan permintaan ke server, dan mengelola respons dari server. Jika permintaan berhasil, data dari respons akan diolah dan diteruskan ke pemanggilan `onResponse`. Jika terjadi kegagalan dalam proses, respons akan diolah sebagai respons gagal. Metode-metode ini memanfaatkan callback

yang diimplementasikan di dalamnya untuk menangani hasil dari permintaan jaringan. Selain itu, terdapat manajemen kesalahan baik pada tingkat jaringan maupun pada tingkat pemrosesan respons. Misalnya, jika terjadi kegagalan dalam mengirimkan permintaan atau dalam memproses respons, maka informasi kesalahan akan dikemas dalam respons gagal yang diteruskan melalui pemanggilan `onResponse`:

```
        fun getAllOrder(onResponse: (ResponseStatus<OrderGetResponse?>)-
> Unit){
            val request =
NetworkClient.getWithBearerToken(orderEndpoint, DataToken.token)
            NetworkClient
                .client.newCall(request)
                .enqueue(object : Callback{
                    override fun onFailure(call: Call, e: IOException) {
                        onResponse.invoke(
                            ResponseStatus.Failed(
                                code = -1,
                                message = e.message.toString(),
                                throwable = e
                            )
                        )
                    }

                    override fun onResponse(call: Call, response:
Response) {
                        Log.d("Response", "Status Code: ${response.code}
Msg: ${response.body.toString()}")
                        if (response.isSuccessful){
                            val data =
deserializeJson<OrderGetResponse>(response.body?.string() ?: "") ?:
OrderGetResponse(0)
                            onResponse.invoke(ResponseStatus.Success(
                                data = data,
                                method = "GET",
                                status = true
                            ))
                        } else {
                            onResponse.invoke(
                                mapFailedResponse(response)
                            )
                        }
                        response.body?.close()
                    }

                })
        }

    fun getOrderById(id: String, onResponse:
(ResponseStatus<OrderGetResponse?>) -> Unit){
            val request = NetworkClient.requestById(historyEndpoint,
DataToken.token, DataToken.idUser)
            NetworkClient
                .client.newCall(request)
                .enqueue(object : Callback{
                    override fun onFailure(call: Call, e: IOException) {
                        onResponse.invoke(
```

```
                            ResponseStatus.Failed(
                                code = -1,
                                message = e.message.toString(),
                                throwable = e
                            )
                        )
                    }

    fun postOrder(service: String, status: String, address: String,
mapUrl: String, onResponse: (ResponseStatus<OrderResponse?>) -> Unit){
            val request = NetworkClient.requestOrder(orderEndpoint,
DataToken.token, service, status, address, mapUrl)
            NetworkClient
                .client
                .newCall(request)
                .enqueue(object: Callback{
                    override fun onFailure(call: Call, e: IOException) {
                        onResponse.invoke(
                            ResponseStatus.Failed(
                                code = -1,
                                message = e.message.toString(),
                                throwable = e
                            )
                        )
                    }
```

Gambar 4.37 Order API

c. UserAPI

Kelas `UserAPI` bertanggung jawab untuk mengelola komunikasi antara aplikasi MECHA dan server menggunakan API terkait pengguna. Terdapat beberapa metode yang digunakan dalam kelas ini. Metode `getAllUser` digunakan untuk mengambil daftar pengguna dari server. Fungsi ini mengirimkan permintaan GET ke endpoint `/users` dengan token otorisasi Bearer. Hasil respons dari server akan diuraikan menjadi objek `UserResponse` dan diteruskan ke pemanggil melalui `onResponse`. Metode `loginUser` memungkinkan pengguna untuk masuk ke aplikasi dengan mengirimkan email dan kata sandi. Fungsi ini membuat permintaan POST ke endpoint `/login` dengan kredensial pengguna. Hasil respons dari server akan diuraikan menjadi objek `LoginResponse` dan diteruskan ke pemanggil melalui `onResponse`. Metode `regisUser` digunakan untuk mendaftarkan pengguna baru. Fungsi ini mengirimkan permintaan POST ke endpoint `/register` dengan informasi pendaftaran pengguna. Hasil respons dari server akan diuraikan menjadi objek `RegisResponse` dan diteruskan ke pemanggil melalui `onResponse`. Setiap metode juga menangani respons dari server, termasuk penanganan situasi ketika permintaan gagal atau tidak berhasil. Jika terjadi kegagalan, kesalahan akan

dikirim kembali kepada pemanggil melalui `onResponse` bersama dengan informasi tentang kesalahan yang terjadi. Kelas `UserAPI` membentuk jembatan vital antara antarmuka pengguna dan backend server, memastikan bahwa operasi yang berkaitan dengan pengguna berjalan dengan lancar dan aman.:

```
    fun loginUser(email:String, password: String, onResponse:
(ResponseStatus<LoginResponse?>) -> Unit){
            val request = NetworkClient.requestLogin(loginEndpoint,
email, password)
            NetworkClient
                .client
                .newCall(request)
                .enqueue(object: Callback{
                    override fun onFailure(call: Call, e: IOException) {
                        onResponse.invoke(
                            ResponseStatus.Failed(
                                code = -1,
                                message = e.message.toString(),
                                throwable = e
                            )
                        )
                    }

        fun regisUser(nama: String, email: String, notelp: String,
password: String, onResponse: (ResponseStatus<RegisResponse?>) -> Unit )
{
            val request = NetworkClient.requestResgis(registerEndpoint,
nama, email, notelp, password)
            NetworkClient
                .client
                .newCall(request)
                .enqueue(object: Callback{
                    override fun onFailure(call: Call, e: IOException) {
                        onResponse.invoke(
                            ResponseStatus.Failed(
                                code =  +1,
                                message = e.message.toString(),
                                throwable = e
                            )
                        )
                    }

                    override fun onResponse(call: Call, response:
Response) {
                        if(response.isSuccessful) {
                            val moshi = Moshi.Builder().build()
                            val adapter: JsonAdapter<RegisResponse> =
moshi.adapter(RegisResponse::class.java)
                            val user =
adapter.fromJson(response.body?.string() ?:"")
                            onResponse.invoke(
                                ResponseStatus.Success(
                                    data = user,
                                    method = "POST",
                                    status = true
                                )
```

```
                            )
                        } else {
                            onResponse.invoke(
                                mapFailedResponse(response)
                            )
                        }
                        response.body?.close()
                    }

                })
        }
```

Gambar 4.38 User API

4.7 Adonis dan *Ngrok* pada aplikasi MECHA

Adonis merupakan sebuah framework web yang digunakan sebagai kerangka pengembangan untuk membangun aplikasi backend pada aplikasi MECHA. Framework ini terpilih karena menyediakan struktur dan alat yang kuat untuk mempermudah pengembangan, serta mengadopsi desain pattern MVC (Model-View-Controller). Penerapan desain pattern MVC memungkinkan MECHA untuk memisahkan logika bisnis dari presentasi data. Model bertanggung jawab untuk mengelola data, View menangani tampilan antarmuka pengguna, dan Controller mengontrol alur logika aplikasi. Hal ini memudahkan tim pengembang untuk bekerja secara terstruktur, meningkatkan keterbacaan kode, dan memungkinkan skalabilitas aplikasi dengan lebih baik. Dengan menggunakan Adonis dan desain pattern MVC, MECHA dapat dikembangkan secara efisien dan dapat diadaptasi dengan lebih mudah dalam menghadapi perubahan kebutuhan atau peningkatan fitur di masa mendatang.

Selanjutnya, untuk mengakses server lokal pada backend saat pengembangan, digunakan Ngrok. Ngrok berperan sebagai alat untuk membuat server lokal yang dapat diakses melalui internet, memungkinkan tim pengembang untuk menguji dan mengevaluasi aplikasi MECHA secara real-time. Hal ini sangat berguna dalam pengujian fungsionalitas dan pengoptimalan aplikasi sebelum diluncurkan secara publik. Penggabungan Adonis dengan Ngrok membuktikan kehandalannya dalam memfasilitasi proses pengembangan dan pengujian aplikasi MECHA sebelum dirilis secara resmi kepada pengguna akhir.

4.8 Implementasi Backend

Backend aplikasi MECHA adalah inti dari seluruh sistem, mengurus semua proses logika, pengolahan data, dan interaksi dengan basis data. Di bagian ini, diuraikan penggunaan

alat dan teknologi penting seperti Postman API, DBeaver, PostgreSQL, JavaScript, dan Framework Adonis. Postman berfungsi sebagai platform untuk menguji dan mengelola antarmuka pemrograman aplikasi (API), memastikan komunikasi yang efisien antara bagian depan dan belakang sistem. DBeaver berperan sebagai alat untuk mengelola dan memantau basis data PostgreSQL, memungkinkan pengelolaan struktur dan kueri database. Penggunaan bahasa pemrograman JavaScript adalah inti dari pengembangan logika aplikasi, sementara Framework Adonis menyediakan kerangka kerja yang kokoh untuk mengatur rute, pengontrol, dan model, mempercepat pengembangan dan memastikan keamanan sistem. Dengan demikian, sub-bab ini memberikan gambaran yang komprehensif tentang teknologi dan infrastruktur yang digunakan dalam backend aplikasi MECHA.

4.8.1 Implementasi Backend Register

Fungsi di bawah adalah implementasi dari endpoint registrasi pada sistem backend. Fungsi ini menangani permintaan registrasi yang masuk, mengekstrak informasi relevan (seperti nama, nomor telepon, email, kata sandi, peran, dan nomor rekening bank) dari permintaan, dan membuat objek pengguna baru. Objek pengguna kemudian diisi dengan informasi ini dan disimpan ke dalam basis data. Selain itu, fungsi ini menghasilkan token untuk otentikasi, memungkinkan pengguna mengakses rute yang diotentikasi. Jika berhasil, ia mengembalikan respons dengan kode status 200, bersama dengan token, pesan keberhasilan, dan status verifikasi pengguna baru. Dalam kasus terjadi kesalahan selama proses registrasi, ia akan melempar sebuah kesalahan.

```
    public async register({ request, auth }: HttpContextContract) {
        try {
            const name = request.input("name")
            const no_telp = request.input("no_telp")
            const email = request.input("email")
            const password = request.input("password")
            const role = request.input("role")
            const no_rek = request.input("no_rek")

            const newUser = new User()

            newUser.name = name
            newUser.no_telp = no_telp
            newUser.email = email
            newUser.password = password
            newUser.role = role
            newUser.no_rek = no_rek

            await newUser.save()
```

```
                const token = await auth.use("api").login(newUser, {
                    expiresIn: "1 days",
                })

                // await sendVerificationOTPEmail(email)

                return {
                    status: 200,
                    token: token.token,
                    message: "Registration is successful",
                    verified: newUser.verified
                }
            } catch (error) {
                throw error
            }
        }
```

Gambar 4.39 Backend Register

4.8.2 Implementasi Backend Login

Fungsi dibawah adalah implementasi dari proses autentikasi saat pengguna melakukan login dalam sistem backend. Fungsi ini menerima permintaan dari pengguna dengan input berupa alamat email dan kata sandi. Selanjutnya, sistem akan mencoba untuk melakukan autentikasi dengan menggunakan alamat email dan kata sandi yang diberikan. Jika autentikasi berhasil, sistem akan menghasilkan sebuah token yang akan digunakan untuk otorisasi selama 1 hari. Selain itu, sistem akan mencari informasi pengguna berdasarkan alamat email. Jika informasi pengguna tidak ditemukan, maka akan dianggap bahwa alamat email yang dimasukkan tidak valid. Jika autentikasi berhasil dan informasi pengguna ditemukan, sistem akan mengembalikan respons dengan status 200, ID pengguna, token, nama pengguna, alamat email, peran pengguna, dan status verifikasi pengguna. Namun, jika terdapat kesalahan selama proses autentikasi, sistem akan melemparkan pesan kesalahan yang diterima.

```
    public async login({ request, auth }: HttpContextContract) {
        try {
            const email = request.input("email")
            const password = request.input("password")

            const token = await auth.use("api").attempt(email, password,
{
                expiresIn: "1 days",
            })

            const fetchedUser = await User.findBy("email", email)

            if (!fetchedUser) {
                throw Error("Invalid email entered!")
```

```
            }

            return {
                status: 200,
                id: fetchedUser.id,
                token: token.token,
                nama: fetchedUser.name,
                email: fetchedUser.email,
                role: fetchedUser.role,
                verified: fetchedUser.verified
            }
        } catch (error) {
            throw error
        }
    }
```

Gambar 4.40 Backend Login

4.8.3 Implementasi Pesan Layanan

Fungsi di bawah adalah bagian dari implementasi backend untuk membuat pesanan baru dalam aplikasi. Pertama-tama, fungsi ini memastikan bahwa pengguna telah terautentikasi menggunakan token API. Kemudian, ia mengambil informasi penting dari permintaan, seperti nama layanan, status pesanan, alamat, URL peta, dan total biaya. Selanjutnya, pesanan baru dibuat dengan mengisi informasi yang diperlukan dan disimpan ke dalam basis data. Setelah itu, server memberikan respons dengan kode status 200 yang menunjukkan keberhasilan pembuatan pesanan bersama dengan informasi terkait seperti ID layanan, pesan sukses, dan rincian pesanan. Namun, jika terjadi kesalahan selama proses ini, server akan memberikan respons dengan kode status 404 dan pesan kesalahan yang sesuai.

```
public async createOrder({ request, response, auth} :
HttpContextContract) {
        try {
            await auth.use("api").authenticate()

            const user = auth.use('api').user
            const { name_service, status, address, map_url, sum } =
request.all()

            const newOrder = new Order()
            newOrder.fill({
                name: user?.name,
                id_service: generateRandomValue(),
                name_service,
                status,
                address,
                img_url: user?.img_url,
                map_url,
                sum,
                user_id: user?.id
            })

            await newOrder.save()
```

```
            response.status(200).json({
                status: 200,
                id_service: newOrder.id_service,
                msg: "Order created successfully",
                order: newOrder
            })
        } catch (error) {
            response.status(404).json({
                status: 404,
                msg: error.message
            })
        }
    }
```

Gambar 4. 41 Backend Pesan Layanan

4.8.4 Implementasi Ambil Pesanan

Fungsi yang dibawah merupakan bagian dari backend aplikasi dan bertanggung jawab untuk mengembalikan data pesanan berdasarkan ID pengguna tertentu. Pada awalnya, fungsi melakukan autentikasi pengguna melalui token API. Setelah itu, ID pengguna diambil dari parameter yang diterima. Kemudian, fungsi melakukan query pada database untuk mencari pesanan yang terkait dengan ID pengguna tersebut. Selama query, data harga juga dimuat terkait dengan setiap pesanan. Hasil dari query adalah data pesanan yang memuat informasi seperti ID, nama pengguna, ID layanan, nama layanan, status pesanan, alamat, tautan peta, tanggal pembuatan, total harga, dan tautan gambar. Jika operasi berjalan dengan sukses, fungsi mengembalikan respons dengan kode status 200 dan data pesanan. Namun, jika terjadi kesalahan selama proses ini, fungsi akan memberikan respons dengan kode status 401 dan pesan kesalahan yang terkait.

```
public async getOrderId({ response, auth, params }: HttpContextContract)
{
        try {
            await auth.use("api").authenticate()

            const userId = params.user_id

            const order = await Order.query()
                .where('user_id', userId)
                .preload("prices", (query) => {
                    query.select("id",   "price",   "description_service",
"id_service")
                    })
                .select("id",   "name",   "id_service",   "name_service",
"status", "address", "map_url", "created_at", "sum", "img_url")

            response.status(200).json({
                id: userId,
```

```
                status: 200,
                msg: "Order hit successfully",
                order: order
            })
        } catch (error) {
            response.status(401).json({
                status: 401,
                msg : error.message
            })
        }
    }
```

Gambar 4. 42 Backend Ambil Pesanan

4.8.5 Implementasi History

Fungsi yang diberikan merupakan implementasi dari endpoint untuk mendapatkan riwayat pesanan berdasarkan ID pengguna. Pertama-tama, fungsi ini mencoba untuk mengotentikasi pengguna dengan menggunakan metode otentikasi API. Selanjutnya, ID pengguna diperoleh dari parameter permintaan. Kemudian, fungsi ini melakukan query ke basis data untuk mendapatkan riwayat pesanan yang terkait dengan ID pengguna tersebut. Selama proses ini, data harga dari layanan juga dimuat bersama. Hasil dari query ini diformat dan hanya informasi terpilih yang diambil, seperti ID, harga, dan deskripsi layanan. Selanjutnya, hasil query diubah sedikit untuk memastikan ID pengguna tetap ada dalam respons. Akhirnya, respons dengan status 200 dan data riwayat pesanan dikirimkan kembali dalam format JSON. Jika terjadi kesalahan selama proses ini, maka respons dengan status 401 dan pesan kesalahan akan dikirimkan kembali.

```
    public async getHistoryId({ response, auth, params }:
HttpContextContract) {
        try {
            await auth.use("api").authenticate()

            const userId = params.user_id

            const output = await HistoryOrder.query()
                .where('user_id', userId)
                .preload("prices", (query) => {
                    query.select("id", "price", "description_service")
                })
                .select("id", "name", "id_service", "user_id",
"name_service", "status", "address", "map_url", "created_at", "sum",
"img_url")

            const historiesWithUserId = output.map((history) => {
                return {
                    ...history.toJSON(),
                    user_id: history.user_id,
                };
```

```
            });

            response.status(200).json({
                status: 200,
                history: historiesWithUserId
            })
        } catch (error) {
            response.status(401).json({
                status: 401,
                msg : error.message
            })
        }
    }
```

Gambar 4. 43 Backend History

4.9 Testing

Tahap pengujian merupakan fase akhir setelah implementasi dari aplikasi MECHA. Dalam tahap ini, pengujian dilakukan dengan melibatkan sejumlah calon pengguna, termasuk pengguna layanan dan mitra bengkel. Evaluasi dilakukan menggunakan metode *SUS* (*System Usability Scale*).

4.9.1 Pengujian *System Usability Scale* (SUS)

Setelah responden mencoba aplikasi MECHA, responden diminta untuk mengisi *google form* mengenai *System Usability Scale* (SUS) yang terdiri dari 10 (sepuluh) pertanyaan yang dimana jawaban yang digunakan skala 1 hingga 5. Daftar pertanyaan yang digunakan tertera pada Tabel 4. 4. .

Tabel 4.5 Daftar Pernyataan

No	Pernyataan	Skala
1	Saya akan ingin lebih sering menggunakan aplikasi ini	1 – 5
2	Saya merasa aplikasi ini tidak harus dibuat serumit ini	1 – 5
3	Saya piker aplikasi ini mudah untuk digunakan	1 – 5
4	Saya membutuhkan bantuan dari orang teknis untuk menggunakan aplikasi ini	1 – 5
5	Saya menemukan fitur pada aplikasi ini terintegrasi dengan baik	1 – 5
6	Saya piker ada ketidaksesuaian dalam aplikasi ini	1 – 5
7	Saya merasa kebanyakan orang mudah untuk mempelajari aplikasi dengan sangat cepat	1 – 5
8	Saya menemukan, aplikasi sangat rumit untuk digunakan	1 – 5
9	Saya percaya diri untuk menggunakan aplikasi ini	1 – 5
10	Saya perlu belajar ssebelum saya menggunakan aplikasi ini	1 – 5

Berikut hasil pengujian yang dilakukan oleh responded yang tertera pada Tabel 4.6. 2.

Tabel 4.6 Hasil Pengujian

Responded (Tester)	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10

R1	4	2	4	2	4	3	4	2	4	2
R2	5	2	4	2	4	1	4	1	4	2
R3	5	2	5	1	5	2	5	2	5	1
R4	5	2	4	1	3	2	4	2	3	2
R5	5	2	4	2	5	2	4	2	4	2
R6	5	2	4	2	5	4	3	2	4	4
R7	5	2	4	2	5	1	5	1	5	1
R8	4	2	4	1	4	2	3	2	3	4
R9	5	2	4	2	5	1	4	3	5	4
R10	4	3	4	2	4	3	4	2	4	2

Setelah menerima jawaban dari responden yang telah mengisi *google form* yang diberikan, penulis melakukan perhitungan dengan rumus yang ada pada metode *System Usability Scale*. Berikut hasil perhitungan yang tertera pada Tabel 4.6 :

Tabel 4.7 Hasil Usability Testing

Responded (Tester)	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**	**Nilai (Total)**	**Nilai Akhir (Total x 2,5)**
R1	3	3	3	3	3	2	3	3	3	3	29	72,5
R2	4	3	3	3	3	4	3	4	3	3	33	82,5
R3	4	3	4	4	4	3	4	3	4	4	37	92,5
R4	4	3	3	4	2	3	3	3	2	3	30	75
R5	4	3	3	3	4	3	3	3	3	3	32	80
R6	4	3	3	3	4	1	2	3	3	1	27	67,5
R7	4	3	3	3	4	4	4	4	4	4	37	92,5
R8	3	3	3	4	3	3	2	3	2	1	27	67,5
R9	4	3	3	3	4	4	3	2	4	1	31	77,5
R10	3	2	3	3	3	2	3	3	3	3	28	70
Nilai Skor Akhir												77,75

Dengan memanfaatkan rumus *System Usability Scale*, desain aplikasi MECHA memperoleh skor akhir sekitar 77,75. Sesuai dengan Subbab 2.4 yang membahas mengenai *System Usability Scale*, desain aplikasi MECHA telah mencapai klasifikasi Bagus ketika skornya berada di kisaran 68 hingga 80,3. Maka dari itu, dengan nilai akhir 77,75, dapat disimpulkan bahwa desain aplikasi MECHA memperoleh klasifikasi B, menandakan bahwa desain tersebut sudah dapat dianggap "Bagus".

4.9.2 Hasil *Testing* Mitra Bengkel(karena sample sedikit lebih baik kualit)

Setelah melaksanakan tahap implementasi *Model View Presenter* pada aplikasi MECHA dengan fokus pada fitur utama, langkah selanjutnya adalah melakukan pengujian dengan mitra bengkel. Hal ini dilakukan untuk memverifikasi kesesuaian konsep bisnis yang telah dikembangkan dan memastikan bahwa fitur utama dapat beroperasi dengan lancar tanpa mengalami kendala. Melalui proses pengujian ini, tim pengembang berharap dapat meningkatkan kualitas aplikasi MECHA di masa depan, berdasarkan masukan dan umpan balik yang diberikan oleh mitra-mitra bengkel.

Tabel 4.8 Hasil Pengujian Montir

No	Pertanyaan	Jawaban
Nama : Mas Doni, Profesi: Pemilik bengkel tambal ban		
1	Pertanyaan terkait konsep ide aplikasi MECHA	Ide yang Anda sampaikan sangat menarik. Aplikasi ini berpotensi untuk memperluas jangkauan bengkel Anda, memungkinkan masyarakat untuk mengetahui keberadaan dan layanan yang ditawarkan di bengkel milik Anda. Dengan aplikasi ini, diharapkan dapat meningkatkan jumlah pelanggan yang mempercayakan kendaraan mereka untuk diperbaiki di bengkel Anda.
2	Pertanyaan terkait sistem pembayaran pada aplikasi dan model bisnis	Sistem penagihan dalam aplikasi MECHA tidak menimbulkan beban berat karena biaya ditanggung oleh pelanggan, sehingga tidak terlalu mengganggu. Standarisasi harga layanan juga memberikan bantuan signifikan, sehingga tidak ada kebutuhan untuk menentukan harga secara independen.
3	Pertanyaan terkait konsep tampilan dan fitur yang tersedia	Tampilan aplikasinya dirancang dengan sederhana dan mudah dipahami. Fitur-fiturnya mencakup kemampuan untuk mengambil pesanan, melihat lokasi, dan melakukan pembayaran, yang sangat membantu saya sebagai pemilik bengkel. Namun, saya juga berharap ada opsi untuk menunda pengambilan pesanan sementara saya memberi tahu pelanggan bahwa saya akan tiba dalam waktu sekitar 5 menit.
4	Pertanyaan terkait ketersediaan untuk menggunakan aplikasi ini	Tentu, aplikasi ini memiliki potensi untuk meningkatkan jumlah pelanggan yang datang ke bengkel saya, karena orang akan lebih mudah menemukan lokasi bengkel saya melalui aplikasi ini.
5	Pertanyaan terkait pengembangan untuk aplikasi MECHA	Apakah mungkin untuk mengembangkan fitur di mana pelanggan dapat memeriksa

		ketersediaan bengkel dan status kehadiran saya sebelum datang? Misalnya, apakah bengkel sedang buka atau apakah saya sedang berada di sana?
Nama : Mas Iky, Profesi: Pemilik bengkel TW Racing		
1	Pertanyaan terkait konsep ide aplikasi MECHA	Ide ini sangat inovatif dan belum pernah ada sebelumnya. Terutama, aplikasi ini akan menjadi penyelamat bagi bengkel-bengkel kecil yang menghadapi kesulitan bersaing dengan bengkel besar. Selain itu, aplikasi ini dapat menjadi alat yang sangat efektif untuk menarik lebih banyak pelanggan, terutama pelanggan mahasiswa, karena mayoritas dari mereka pasti akan senang jika dapat memesan layanan dengan mudah melalui aplikasi.
2	Pertanyaan terkait sistem pembayaran pada aplikasi dan model bisnis	Saya sedikit bingung mengenai apakah tagihan ini termasuk dalam biaya layanan yang harus dibayar kepada montir. Jika ini adalah biaya tambahan, saya harap hal ini tidak akan membuat biaya layanan menjadi terlalu tinggi, karena ini mungkin membuat pelanggan enggan untuk menggunakan jasa layanan ini.
3	Pertanyaan terkait konsep tampilan dan fitur yang tersedia	Desainnya menarik dan saya tidak mengalami kesulitan karena saya sudah terbiasa menggunakan smartphone. Mungkin perlu sedikit waktu untuk beradaptasi dengan tata letaknya.
4	Pertanyaan terkait ketersediaan untuk menggunakan aplikasi ini	Tentu, jika situasinya padat dengan lalu lintas, kemungkinan saya tidak akan menggunakan aplikasinya.
5	Pertanyaan terkait pengembangan untuk aplikasi MECHA	Dapat ditambahkan fitur pemberitahuan untuk memberitahu saat ada pesanan baru masuk, mengingat bahwa montir mungkin tidak selalu dapat memantau ponsel mereka saat sedang bekerja.
Nama : Pak Susilo, Profesi: Pemilik bengkel Lancar Motor		
1	Pertanyaan terkait konsep ide aplikasi MECHA	Saya sangat senang dengan konsep ini, terutama karena usaha bengkel saya telah beroperasi selama beberapa waktu tanpa mendapatkan banyak pelanggan. Saya berharap bahwa dengan menggunakan aplikasi ini, akan ada peningkatan minat dari mahasiswa untuk melakukan perbaikan kendaraan di bengkel saya.
2	Pertanyaan terkait sistem pembayaran pada aplikasi dan model bisnis	Saya agak terkejut, saya pikir aplikasinya gratis, tetapi ternyata ada tagihan yang perlu saya bayar. Mungkin bisa lebih jelas apakah tagihan ini seharusnya ditanggung oleh pelanggan dan bukan oleh saya.
3	Pertanyaan terkait konsep tampilan dan fitur yang tersedia	Saya merasa bingung dengan antarmukanya, terasa agak sulit dipahami

		bagi saya. Menurut pendapat saya, antarmuka dapat disederhanakan agar lebih mudah dipahami, tanpa perlu kompleks seperti ini. Mungkin lebih baik jika langsung menampilkan daftar pesanan yang masuk.
4	Pertanyaan terkait ketersediaan untuk menggunakan aplikasi ini	Saya bersedia, namun saya agak kesulitan untuk menjemput kendaraan pelanggan karena saya adalah satu-satunya yang mengurus bengkel ini. Mungkin kedepannya bisa dipertimbangkan untuk meminta pelanggan datang ke bengkel sendiri.
5	Pertanyaan terkait pengembangan untuk aplikasi MECHA	Lebih optimal jika tim pengembang memiliki anggota yang secara proaktif mendekati pelanggan atau menyederhanakan antarmukanya sendiri.

Berdasarkan data yang tercantum pada Tabel 4.7, terdapat tiga sampel responden yang merupakan pemilik bengkel. Total jumlah responden adalah tiga orang. Mengenai konsep ide aplikasi MECHA, seluruh responden sepakat dengan konsep yang telah disajikan beserta nilai dan manfaat yang dapat diberikan kepada pengguna. Namun, terkait sistem pembayaran MECHA, dua dari responden masih merasa bingung dengan kejelasan proses dan cara kerja dari tagihan. Mereka mengakui bahwa mereka perlu mempelajarinya lebih lanjut untuk memahaminya dengan baik. Meskipun begitu, salah satu responden pemilik bengkel menyatakan bahwa standarisasi harga untuk setiap layanan sangat menguntungkan bagi pemilik bengkel, karena mereka tidak perlu bingung dalam menentukan harga perbaikan kendaraan. Dua dari tiga responden pemilik bengkel berpendapat bahwa antarmuka dari aplikasi sudah cukup sederhana dan mudah untuk digunakan, serta sudah memenuhi kebutuhan mereka. Akan tetapi, satu dari tiga responden pemilik bengkel menganggap bahwa antarmuka dari aplikasi MECHA masih tergolong rumit dan sulit untuk dipahami alurnya. Oleh karena itu, responden tersebut merasa perlu untuk lebih terbiasa dengan alur aplikasi sebelum dapat menggunakannya dengan lancar. Pertanyaan mengenai ketersediaan responden untuk menggunakan produk menunjukkan bahwa keseluruhan responden menyatakan minat untuk menggunakan produk. Terakhir, dalam hal pengembangan aplikasi MECHA, kesimpulan dari semua responden adalah bahwa beberapa fitur tambahan sangat diharapkan untuk membantu para montir, seperti adanya notifikasi saat ada pesanan masuk, kemampuan untuk menandai status bengkel jika sedang tutup, dan opsi untuk mendatangkan pelanggan ke bengkel tanpa harus menjemput mereka terlebih dahulu.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil pengembangan aplikasi MECHA berbasis mobile menggunakan MVP sebagai desain pattern dapat dapat disimpulkan sebagai berikut:

a. Dalam pengembangan MECHA menggunakan *tools* adalah *Android Studio,* serta menggunakan Bahasa pemograman kotlin.
b. Pengembang berhasil mengimplementasikan layanan bengkel berbasis mobile.
c. Pengembang berhasil mengimplementasikan desain pattern MVP pada pengembangan MECHA.

5.2 Saran

Berdasarkan hasil pengembangan aplikasi MECHA, masih terdapat kekurangan pada *mobile* yang telah dihasilkan dan kekurangan tersebut masih dapat diperbaiki untuk mengembangkan *mobile* lebih baik lagi. Oleh karena itu terdapat beberapa saran yang dapat dilakukan untuk pengembangan selanjutnya:

a. Menambahkan feature daftar admin agar admin dapat mendaftarkan diri sendiri.
b. Menggunakan core banking system agar pembayaran lebih mudah dilakukan.
c. Menambahkan web admin agar bisa dilakukan manajemen pengguna, dukungan pelanggan, pelaporan dan analisis.
d. Notifikasi jika ada pesanan masuk
e. Memberikan status bengkel jika sedang tutup

DAFTAR PUSTAKA

Alkaff, M., Radam, I. F., & Chandra, W. (2021). *Jurnal Sistem dan Teknologi Informasi Penerapan Pattern MVC (Model View Controller) dalam Pengembangan Aplikasi Identifikasi Jam Puncak Arus Lalu Lintas pada Simpang Lima*. *6*(2). http://jurnal.unmuhjember.ac.id/index.php/JUSTINDO

Anggraeni, R., & Elan Maulani, I. (2023). Pengaruh Teknologi Informasi Terhadap Perkembangan Bisnis Modern. *Jurnal Sosial Teknologi*, *3*(2), 94–98. https://doi.org/10.59188/jurnalsostech.v3i2.635

Brooke, J. (1995). *SUS: A quick and dirty usability scale Fault diagnosis training View project Decision Making in General Practice View project*. https://www.researchgate.net/publication/228593520

Brooke, J. (2013). *SUS: a retrospective System Usability Scale View project Fault diagnosis training View project SUS: A Retrospective* (Vol. 8).

Deva Purwanto, A. G., & Zukhri, Z. (2023). Sistem Informasi Perumahan Purimas Citra Gemilang 2 Berbasis Mobile. *Edusaintek: Jurnal Pendidikan, Sains dan Teknologi*, *10*(2), 646–658. https://doi.org/10.47668/edusaintek.v10i2.820

Dwi Martha, Z., Pramono Adi, E., & Soepriyanto, Y. (2018). *Ebook Berbasis Mobile Learning*.

Ependi, U., Kurniawan, T. B., & Panjaitan, F. (2019). System Usability Scale Vs Heuristic Evaluation: A Review. *Simetris: Jurnal Teknik Mesin, Elektro dan Ilmu Komputer*, *10*(1), 65–74. https://doi.org/10.24176/simet.v10i1.2725

Ferico, A., Pasaribu, O., Darwis, D., & Surahman, A. (2019). Sistem Informasi Geografis Untuk Pencarian Lokasi Bengkel Mobil Di Wilayah Kota Bandar Lampung. *Jurnal TEKNOKOMPAK*, *13*(2), 1–6. http://maps.google.com.

Firdha Pradana, Y., & Kuswinardi, W. (2020). *Jurnal Terapan Sains & Teknologi Rancang Bangun Aplikasi Android Penilaian Kinerja Guru Dengan Metode Agile Development Di Man 1 Kabupaten Malang*. *2*(1).

Fitrahriansyah, I., Haerul Jaman, J., Ilmu Komputer, F., Singaperbangsa Karawang, U., & Hs Ronggowaluyo Telukjambe Timur, J. (2022). *Pemanfaatan Application Programming Interface (API) pada Aplikasi Layanan Jasa Perbaikan Kendaraan Bermotor*. *15*(1), 1979–276. https://doi.org/10.30998/faktorexacta.v15i1.9848

Hapsari, E. D., & Yuda, Y. P. (2022). Efektivitas Penggunaan Media Pembelajaran Menulis Aksara Jawa Berbasis Mobile Terhadap Prestasi Siswa. *EDUSAINTEK: Jurnal*

Pendidikan, Sains dan Teknologi, *9*(2), 351–361. https://doi.org/10.47668/edusaintek.v9i2.475

Hikmah, N., Suradika, A., Andi, R., & Gunadi, A. (2021). Metode Agile Untuk Meningkatkan Kreativitas Guru Melalui Berbagi Pengetahuan (Knowledge Sharing).

Kusnawi. (2010). Penerapan Design Patterns Untuk Perancangan Berbasiskan Objek Oriented. 11.

Mernisiola Zipa, M., Suranto, B., & Vitra Paputungan, I. (2020). Penerapan Metode Lean Startup Pada Aplikasi Evoria.

Ngurah, G., Widhiananda, A., Made, I., Suyadnya, A., & Saputra, K. O. (2017). Rancang Bangun Aplikasi Reservasi Service Untuk Bengkel Sepeda Motor Berbasis Web (Vol. 4, Nomor 2). Komang Oka Saputra.

Ningsih, A. A., Rambe, R. S., Munthe, Y. N., Sialalahi, P. R., William, J., Ps, I. V, Estate, M., Percut, K., Tuan, S., & Serdang, K. D. (2023). Pendekatan Lean Startup Pada Desain Produk Dan Teknik Minimum Viable Product Dalam Menyikapi Skeptisisme Pada Iklim Bisnis (Vol. 2, Nomor 1).

Pratama, E. B., Hendini, A., & Fristian, A. (2023). Pendekatan Metode Prototype Pada Aplikasi Presensi Berbasis Mobile (Studi Kasus: Kantor Desa Mekar Jaya). Dalam *Jurnal Sistem Informasi Akuntansi* ❒ (Vol. 33, Nomor 1). http://jurnal.bsi.ac.id/index.php/justian

Riastuti, M., & Chandra, Y. I. (2022). Perancangan Aplikasi Pelayanan Service Bengkel Motor ABS Menggunakan Model Sequential Linier Berbasis Android. Dalam *Jurnal Esensi Infokom* (Vol. 6, Nomor 1).

Riyadhi, I. M., Intan Purnamasari, & Kamal Prihandani. (2023). Penerapan Pola Arsitektur Mvvm Pada Perancangan Aplikasi Pengaduan Masyarakat Berbasis Android. *INFOTECH journal*, *9*(1), 147–158. https://doi.org/10.31949/infotech.v9i1.5246

Riza Aulia Putra. (2018). Peran Teknologi Digital Dalam Perkembangan Dunia Perancangan Arsitektur. Dalam *Elkawnie: Journal of Islamic Science and Technology* (Vol. 4, Nomor 1). www.jurnal.ar-raniry.com/index.php/elkawnie

Rizki, B., Surya, P., Putra Kharisma, A., & Yudistira, N. (2020). Perbandingan Kinerja Pola Perancangan MVC, MVP, dan MVVM Pada Aplikasi Berbasis Android *(Studi kasus : Aplikasi Laporan Hasil Belajar Siswa SMA BSS)* (Vol. 4, Nomor 11). http://j-ptiik.ub.ac.id

Sahrial, R., Fauzi, D. F., & Susilawati, E. (2022). Pemanfaatan Json Untuk Menampilkan Data Realtime Covid-19 Dengan Model View Presenter. Dalam *Jurnal TEKNOINFO* (Vol. 16, Nomor 1).

Selvi Dwi Hartiyani, Agung Prayogo, & Erlinawati. (2023). Aplikasi Multimedia Pembelajaranaksara Jawa Berbasis Android Untuk Siswa Sekolah Dasar.

Yunus, I., Fiqhan, M., Hendra Brata, A., & Kharisma, A. P. (2018). Pengembangan Aplikasi Pengingat Salat Dengan Konsep Context-Aware Menggunakan MVP Pada Platform Android (Vol. 2, Nomor 10). http://j-ptiik.ub.ac.id

Zaki, B., & Putra, S. D. (2018). Aplikasi Bengkel Online Menggunakan Global Positioning System (Gps) Berbasis Android Pada Cv. Rumah Otomotif.

LAMPIRAN



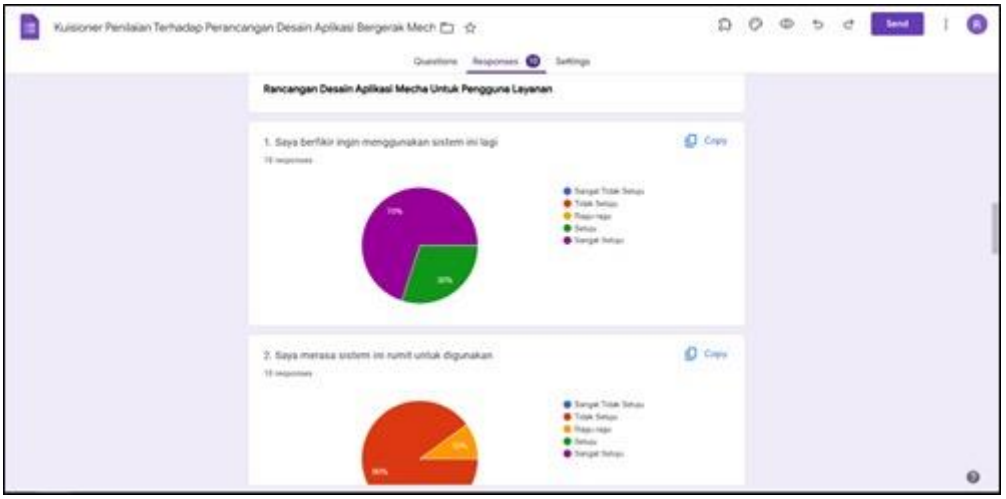
Lampiran Testing Aplikasi



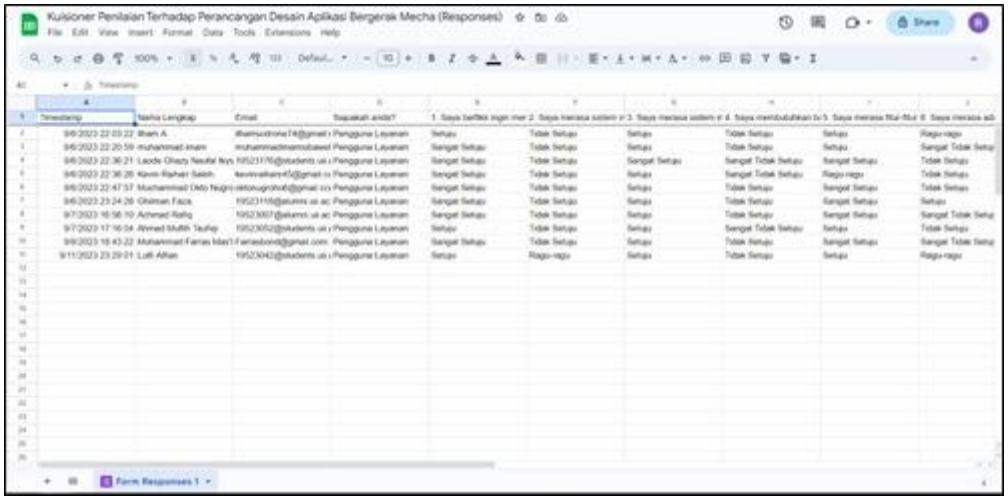
Lampiran Testing Aplikasi



Lampiran Testing Aplikasi



Lampiran Testing Aplikasi



Lampiran Testing Aplikasi