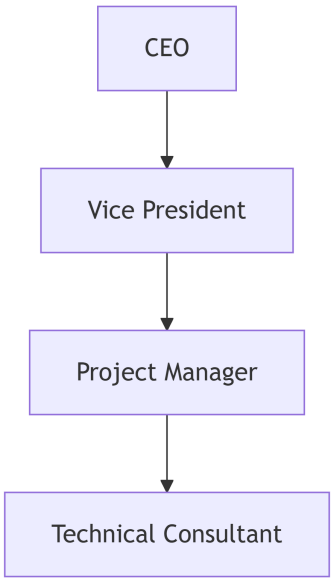
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Mahasiswa ditempatkan pada **Divisi Developer – Technical Consultant** di PT Global Innovation Technology (GIT). Secara garis besar, alur koordinasi berjalan sebagai berikut:

1. **Vice President of Operation** (VP Operation) berperan sebagai pengawas utama program magang dan pemberi arahan strategis.
2. **Project Manager** memecah kebutuhan bisnis menjadi tugas teknis, menugaskan prioritas sprint, serta memantau kemajuan harian melalui mêkanisme daily–stand-up.
3. **Technical Consultant** — tempat mahasiswa berkontribusi — fokus pada pengembangan aplikasi, *maintanance*, dan perbaikan bug.

Diagram alur singkat hubungan kerja dapat dilihat pada Gambar 3.1.



Gambar 3.1. Alur koordinasi mahasiswa di PT GIT.

3.2 Tugas yang Dilakukan

Tugas yang diberikan selama masa magang umumnya ditentukan oleh Project Manager sesuai dengan kebutuhan proyek yang sedang berjalan di PT Global Innovation Technology (PT GIT). Selama periode magang, penulis mendapatkan tugas utama yang meliputi eksplorasi, implementasi, serta pengembangan berbagai use case yang relevan dengan proyek-proyek yang ditangani oleh perusahaan.

1. **Eksplorasi**
   Tahap eksplorasi mencakup kegiatan pendalaman terhadap berbagai teknologi dan platform yang digunakan dalam proyek, termasuk instalasi, konfigurasi, dan pengaturan awal lingkungan kerja. Aktivitas ini berfokus pada pemahaman fitur utama, konsep fundamental, serta topologi sistem yang digunakan.

2. **Implementasi**
   Tahap implementasi mencakup penerapan use case spesifik yang berkaitan langsung dengan kebutuhan perusahaan, seperti pengembangan aplikasi, integrasi sistem pihak ketiga, serta pembuatan dashboard untuk visualisasi data dan monitoring. Dashboard yang dikembangkan bertujuan memberikan wawasan komprehensif mengenai operasional sistem dan mempermudah pengambilan keputusan strategis oleh tim terkait.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang di PT Global Innovation Technology (GIT) untuk setiap minggu diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Pengenalan aplikasi Tukerin, pembuatan ERD, pengembangan frontend barter dan reusable payment dengan JSON dummy.
2	Pembuatan template chat, penyelesaian state seller-buyer, instalasi Node Exporter dan Prometheus pada server Meotrik.
	Lanjut pada halaman berikutnya

Tabel 3.1: Pekerjaan yang dilakukan tiap minggu (lanjutan)

Minggu Ke -	Pekerjaan yang dilakukan
3	Pembuatan data generator KejarTugas, pembuatan panel Meotrik di Grafana, debugging panel bisnis KejarTugas, testing korelasi dan dashboard monitoring backup.
4	Penyelesaian executive dashboard, pengujian ulang korelasi, koneksi antar dashboard, dan stress testing infrastruktur APM.
5	Kembali ke tim development: pembuatan logic payment screen, persiapan infrastruktur backend Tukerin, deployment APK, pengumpulan feedback, dan bugfixing payment screen.
6	Bugfixing notifikasi Tukerin, perbaikan alur tukar tambah, penambahan fitur pemilihan metode pembayaran, perbaikan POV seller-buyer, dan pengembangan MVP Tukerin.
7	Penambahan fitur edit akun pada MVP Tukerin, perbaikan fitur tukar tambah, brainstorming chatbot, pembuatan topologi chatbot, dan riset teknologi Chatbot AI.
8	Eksplorasi bisnis cashierless tenant, eksplorasi chatbot Telegram, perbaikan KejarTugas, finalisasi kebutuhan chatbot, dan pengujian chatbot Telegram.
9	Pembuatan prototipe chatbot tanpa model, kembali ke Meotrik, dan eksplorasi tracing database menggunakan Grafana Tempo untuk KejarTugas.
10	Implementasi tracing database ke Grafana Tempo, studi lanjut Grafana Tempo, serta eksplorasi fitur Service Dependency Graph di Grafana.
11	Finalisasi infrastruktur chatbot, instalasi FAISS, optimisasi model chatbot, pengujian algoritma alternatif, dan eksplorasi backend chatbot.
12	Pengenalan Milvus untuk chatbot, implementasi backend Milvus, pengenalan proyek OCR, eksplorasi OCR dengan Layout-LLM dan Tesseract.
13	Pendalaman teknologi OCR, pembuatan desain frontend OCR di Figma, eksplorasi alternatif OCR dengan Donut, dan pembuatan workflow OCR.

Lanjut pada halaman berikutnya

Tabel 3.1: Pekerjaan yang dilakukan tiap minggu (lanjutan)

Minggu Ke -	Pekerjaan yang dilakukan
14	Pencarian dataset untuk pelatihan OCR, pelatihan model OCR, dan setup database serta API backend OCR.
15	Pelatihan lanjutan model OCR bersama tim, implementasi backend ke frontend Donut, pelatihan MiniMax, eksplorasi alternatif model, integrasi backend ke server SIEM.
16	Integrasi model AI ke backend server, pembuatan frontend inventaris KT, serta integrasi backend inventaris KT.
17	Finalisasi integrasi backend inventaris KT, debugging dan penyempurnaan inventaris, pengenalan konsep Identity Access Management (IAM), serta deployment backend OCR ke server Meotrik APM.
18	Persiapan integrasi frontend OCR dengan backend yang sudah dideploy, serta pendalaman lebih lanjut mengenai IAM.

3.3.1 Aplikasi Barter Tukerin

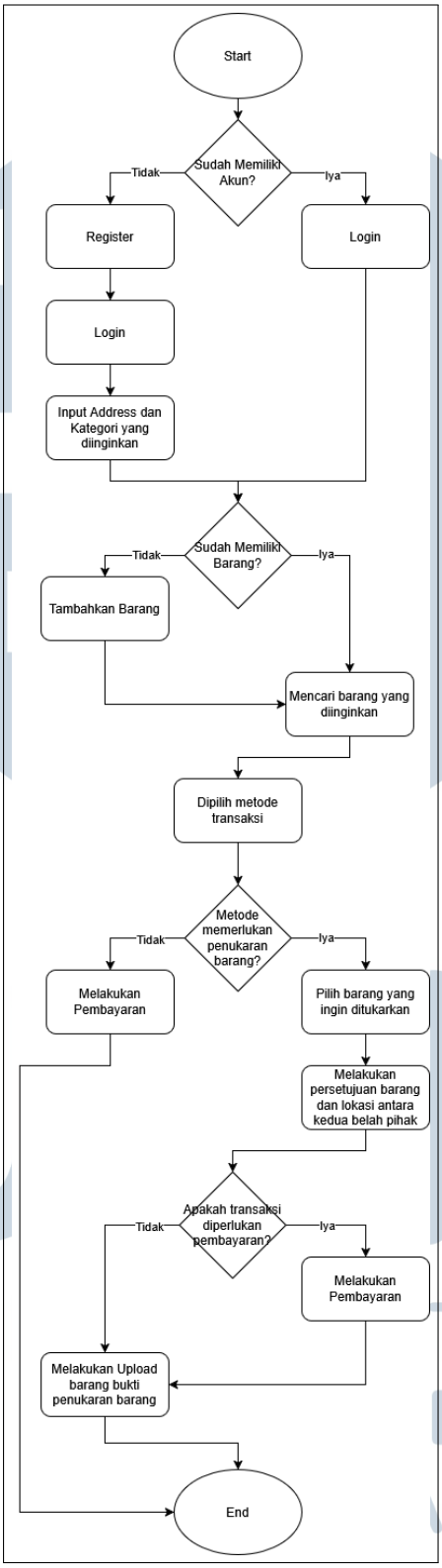
Gambar 3.2. Logo Sementara Tukerin

Proyek Tukerin merupakan aplikasi yang memfasilitasi pengguna untuk melakukan barter dan tukar tambah barang secara praktis, aman, dan efisien. Selama pelaksanaan kerja magang, penulis berperan aktif dalam pengembangan frontend, deployment, serta integrasi fitur-fitur utama lainnya pada aplikasi ini.

Penulis akan memaparkan berbagai aktivitas yang telah dilakukan selama menjalani masa kerja magang. Pada periode ini, aplikasi Tukerin masih berada

dalam tahap *prototype* dan belum tersedia untuk dirilis secara publik. Hingga akhir masa magang, fitur yang berhasil dikembangkan adalah fitur barter dan tukar tambah. Sementara itu, fitur lain seperti jual, lelang, dan *giveaway* belum sempat direalisasikan karena proyek mengalami penghentian sementara oleh pihak perusahaan.

Secara garis besar, alur penggunaan aplikasi dimulai dari proses pembuatan akun oleh pengguna. Namun demikian, pengguna tetap dapat melakukan pencarian tanpa harus memiliki akun terlebih dahulu. Ilustrasi dari keseluruhan alur penggunaan yang dijadikan dasar dalam proses pengembangan aplikasi dapat dilihat pada Gambar 3.3.



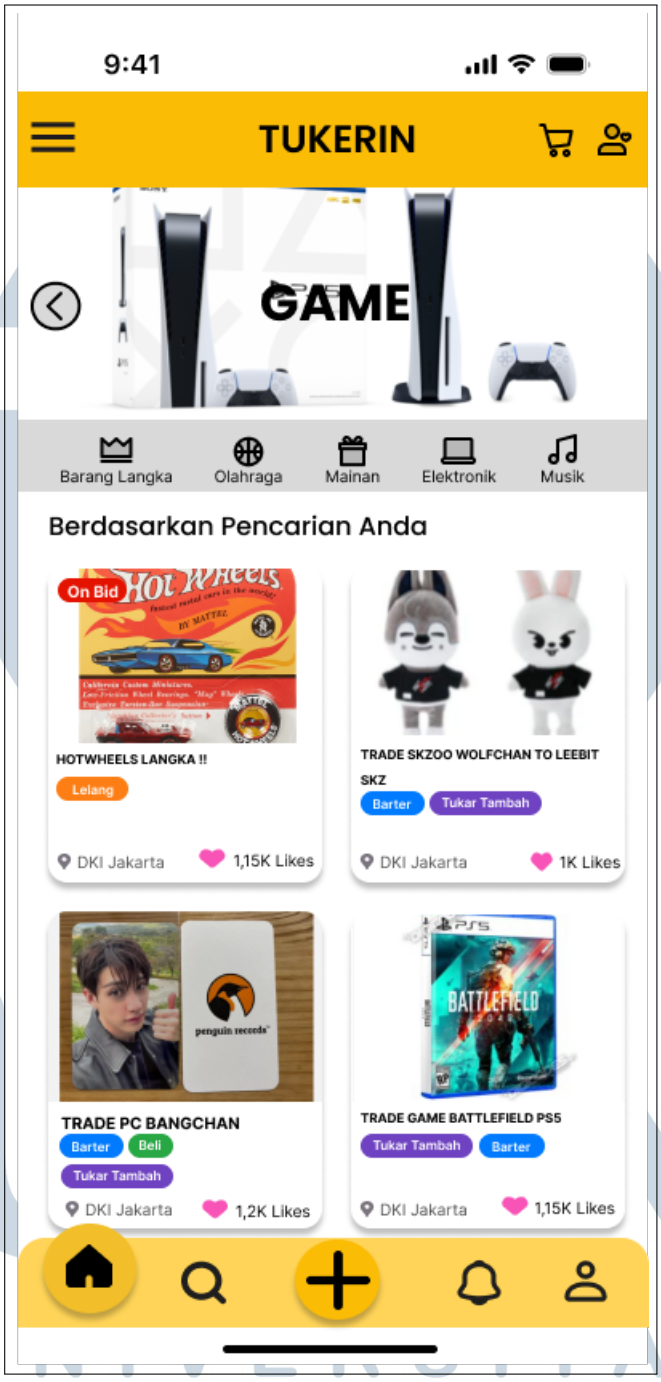
Gambar 3.3. Alur Pemakaian Aplikasi untuk Melakukan Transaksi [3]

A Desain Awal (Mockup)

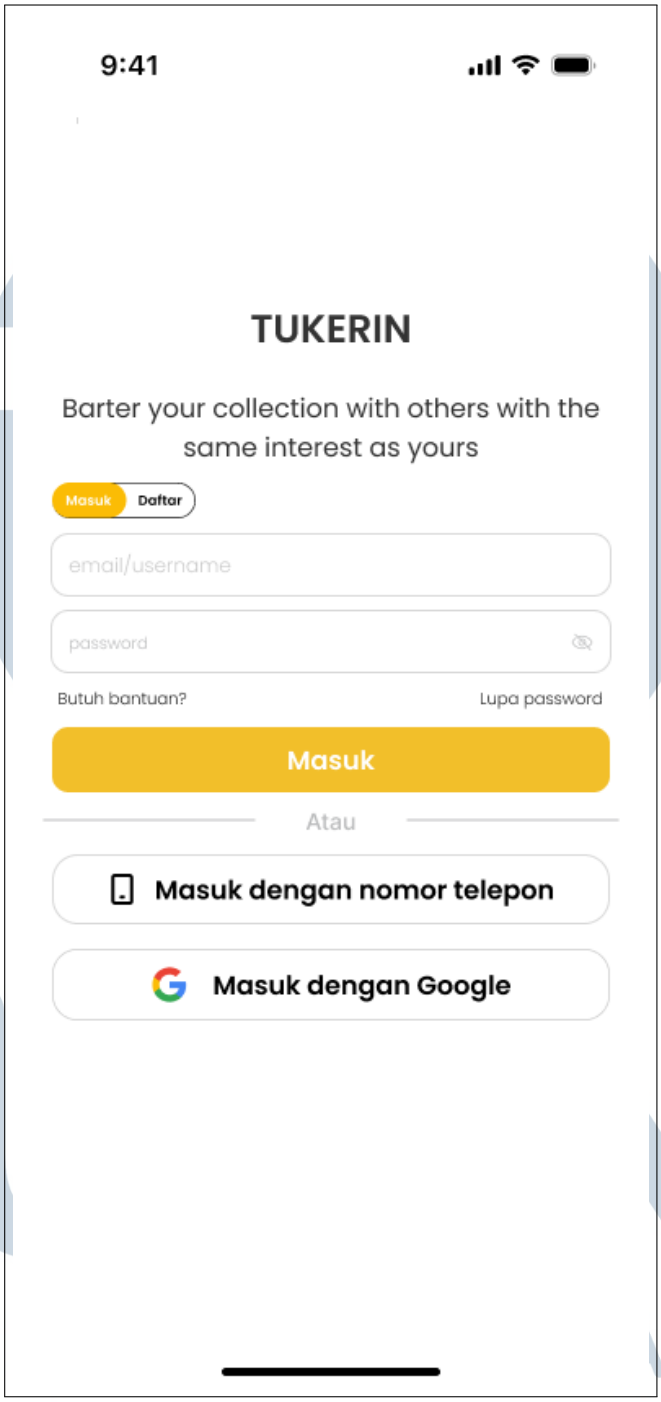
Tahap perancangan dan analisis sistem dilakukan secara sistematis untuk memastikan aplikasi yang dikembangkan tepat sasaran sesuai kebutuhan pengguna serta kondisi pasar. Setiap aktivitas dilakukan secara iteratif agar setiap fitur yang dikembangkan memenuhi ekspektasi pengguna dan tetap relevan dengan dinamika pasar yang ada. Aktivitas pada tahap ini meliputi:

A.1 Perancangan Awal (Frontend)

Pada tahap ini, penulis melakukan studi mendalam terkait alur interaksi pengguna terhadap frontend aplikasi Tukerin. Perancangan dilakukan dengan menggunakan perangkat desain seperti Figma, yang mencakup perancangan tampilan antarmuka utama seperti halaman *homepage*, halaman detail produk, fitur transaksi barter dan tukar tambah, fitur percakapan (*chat*) antar pengguna, serta halaman profil pengguna. Mockup awal ini diuji coba menggunakan data simulasi dalam format JSON untuk menggambarkan secara realistis bagaimana interaksi pengguna berlangsung. Hasil rancangan ini kemudian menjadi acuan utama dalam proses pengembangan frontend, membantu tim pengembang untuk memahami dengan jelas setiap kebutuhan teknis serta fungsional dari masing-masing fitur, untuk apa yang sudah dilakukan sebelumnya oleh tim frontend sebelumnya dilihat pada gambar 3.4 dan juga gambar 3.5.



Gambar 3.4. Main Page yang masih hardcoded [3]

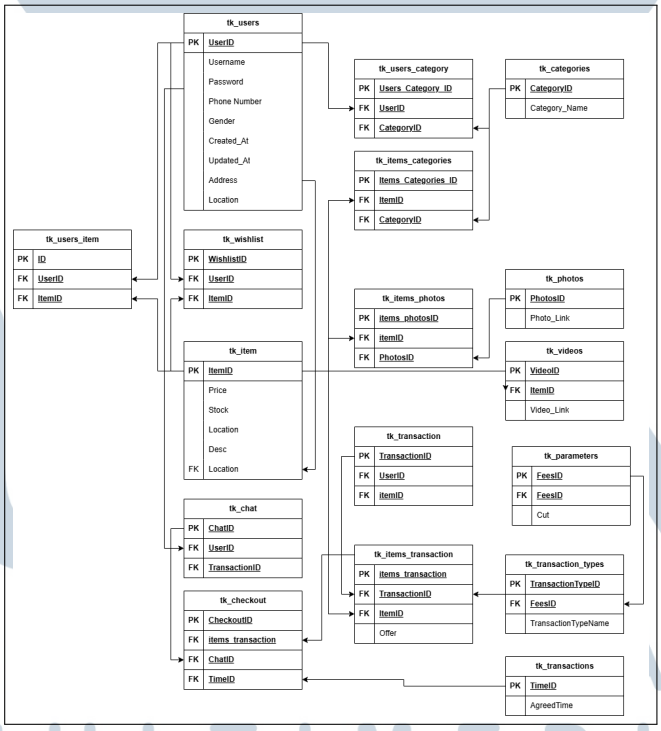


Gambar 3.5. Halaman login yang masih hardcoded [3]

A.2 Perancangan Backend (Barter)

Perancangan backend untuk aplikasi Tukerin dilakukan dengan membuat Entity Relationship Diagram (ERD) yang detail untuk merepresentasikan struktur database secara menyeluruh. Pada tahap awal implementasi, fitur barter menjadi prioritas utama dalam pengembangan sistem backend ini. Oleh karena itu, ERD dirancang secara khusus agar dapat mendukung seluruh kebutuhan transaksi barter, termasuk penawaran antar pengguna, negosiasi melalui fitur chat, hingga mekanisme checkout dan konfirmasi transaksi secara real-time. Namun, ERD yang dibuat tidak hanya terbatas pada barter saja. Diagram ini juga dirancang secara fleksibel, mempertimbangkan kemungkinan pengembangan fitur-fitur lain di masa depan seperti jual beli, lelang, maupun giveaway.

Visualisasi lengkap mengenai ERD aplikasi Tukerin untuk transaksi barter dan tukar tambah dapat dilihat pada Gambar 3.6.



Gambar 3.6. ERD Aplikasi untuk melakukan kegiatan transaksi barter

Diagram ini secara jelas memperlihatkan hubungan antar entitas penting seperti data pengguna (*tk_users*), data barang (*tk_item*), wishlist pengguna (*tk_wishlist*), hingga data transaksi barter yang melibatkan berbagai tabel tambahan seperti tabel checkout (*tk_checkout*), chat antar pengguna (*tk_chat*), tabel parameter

transaksi (*tk_parameters*), serta jenis transaksi yang dapat diperluas di masa depan (*tk_transaction_types*). Perancangan ERD ini dimaksudkan untuk memastikan integritas data, konsistensi hubungan antar entitas, serta memberikan kemudahan dalam mengakomodasi berbagai skenario transaksi yang akan berkembang pada aplikasi Tukerin ke depannya.

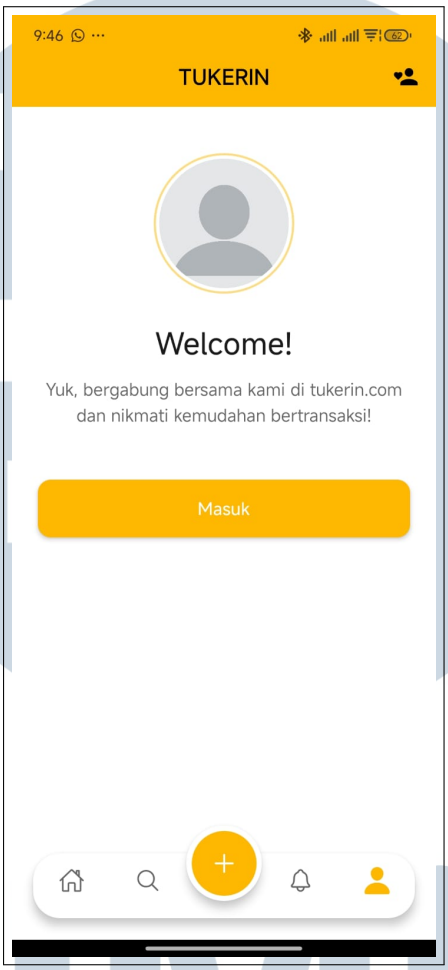
B Pembuatan Frontend Aplikasi Tukerin

Pengembangan *frontend* pada aplikasi Tukerin bertujuan untuk menghadirkan antarmukanya yang intuitif, responsif, serta mudah digunakan oleh pengguna dalam berbagai aktivitas transaksi. Aplikasi ini dibangun menggunakan *framework React Native* yang dilengkapi dengan Expo, dengan pertimbangan bahwa teknologi tersebut mampu mendukung pengembangan aplikasi lintas platform secara efisien, baik untuk Android maupun iOS hanya dengan satu basis kode.

Pemilihan React Native dilakukan karena keunggulannya dalam mempercepat proses pengembangan aplikasi mobile, kemampuan modularisasi kode yang baik, serta dukungan dokumentasi dan komunitas yang kuat. Framework ini memungkinkan reuse kode hingga 90% antara platform Android dan iOS, melalui arsitektur berbasis komponen, hot-reloading, serta ekosistem plugin dan pustaka yang luas. Selain itu, React Native mendukung praktik modularisasi struktur aplikasi, yang memudahkan pengelolaan kode, pengujian, dan kolaborasi tim, khususnya dalam proyek skala besar [5]. Dengan kemampuan untuk membangun antarmuka native dan menulis logika aplikasi sekali saja untuk berbagai platform, React Native membantu tim developer menekan waktu dan biaya pengembangan, sekaligus menjaga performa dan pengalaman pengguna yang konsisten.

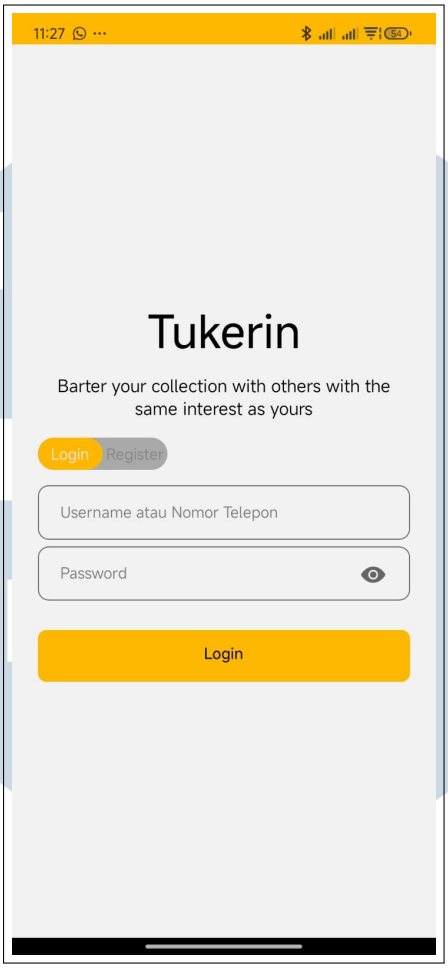
Selain itu, Expo digunakan sebagai framework pendukung untuk menyederhanakan proses pembangunan aplikasi. Expo menyediakan konfigurasi lingkungan yang terintegrasi, akses mudah ke API native, dan dukungan untuk Over-The-Air (OTA) updates—fitur yang memungkinkan pengiriman pembaruan kode JavaScript secara langsung ke pengguna tanpa perlu menunggu persetujuan App Store atau Google Play [6]. Kombinasi React Native dan Expo memungkinkan tim developer untuk fokus pada pengembangan logika aplikasi dan tampilan antarmuka tanpa terbebani konfigurasi native yang kompleks.

B.1 Pembuatan Halaman Login dan Register



Gambar 3.7. Tampilan awal untuk melakukan *login* atau *register*

Pengguna yang baru pertama kali mengakses aplikasi atau belum melakukan *login* akan disambut dengan tampilan awal aplikasi Tukerin, seperti yang terlihat pada Gambar 3.7. Layar ini dirancang secara khusus untuk memberikan pengalaman awal yang positif bagi pengguna baru. Dari tampilan ini, pengguna dapat melanjutkan dengan memilih tombol ”Masuk” untuk berpindah ke halaman berikutnya yang menyediakan opsi untuk melakukan *login* atau registrasi.

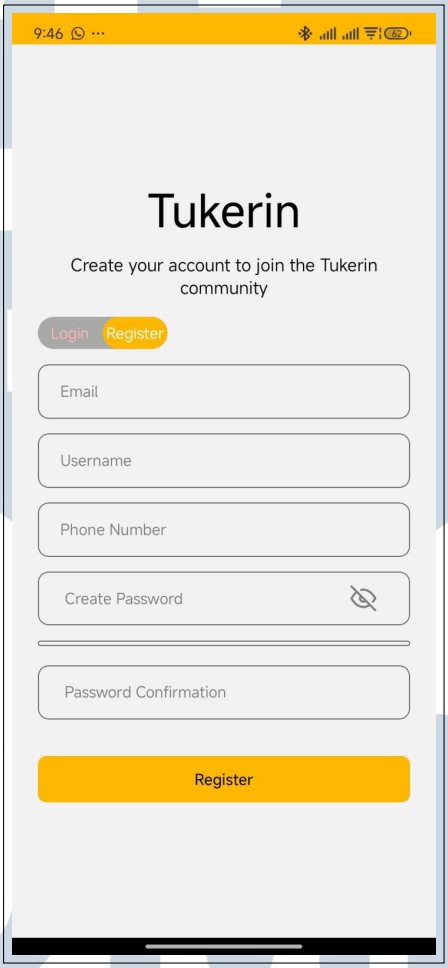


Gambar 3.8. Halaman *login*

Setelah pengguna menekan tombol ”Masuk”, aplikasi akan menampilkan halaman *login*, yang menjadi akses utama ke fitur-fitur utama aplikasi Tukerin seperti transaksi atau melihat detail barang. Halaman *login* ini dirancang dengan pendekatan minimalis yang mengutamakan kejelasan informasi dan kemudahan penggunaan. Seperti ditampilkan pada Gambar 3.8, halaman ini menyediakan elemen input untuk mengisi *username*, nomor telepon, atau email, serta elemen input khusus untuk *password* yang disertai dengan ikon mata untuk menampilkan atau menyembunyikan teks *password*.

Secara teknis, elemen input menggunakan komponen *TextInput* bawaan dari React Native, sedangkan tombol *”Login”* dibuat menggunakan komponen *TouchableOpacity*. Setelah pengguna menekan tombol tersebut, fungsi autentikasi akan dijalankan dengan mengirimkan permintaan (*request*) ke backend menggunakan Axios dalam format JSON. Apabila autentikasi berhasil, pengguna

akan dialihkan ke halaman utama aplikasi. Penanganan untuk input yang kosong atau tidak valid juga diterapkan untuk meningkatkan pengalaman pengguna. Jika pengguna belum memiliki akun, navigasi ke halaman registrasi dapat dilakukan dengan menekan tombol *tab "Login/Register"* yang terletak di bagian atas kiri layar. Tampilan halaman registrasi ditampilkan pada Gambar 3.9.



Gambar 3.9. Halaman *Register*

Halaman registrasi dirancang khusus bagi pengguna yang belum memiliki akun di aplikasi Tukerin. Pada halaman ini terdapat beberapa kolom input, antara lain: *Email*, *Username*, nomor telepon, *Password*, dan konfirmasi *Password*. Setiap kolom input dilengkapi validasi awal di sisi *frontend* untuk memastikan format dan ketentuan input yang dimasukkan pengguna telah sesuai, seperti format email yang valid, ketentuan panjang dan karakter pada *password*, serta memastikan bahwa *password* dan konfirmasinya saling cocok.

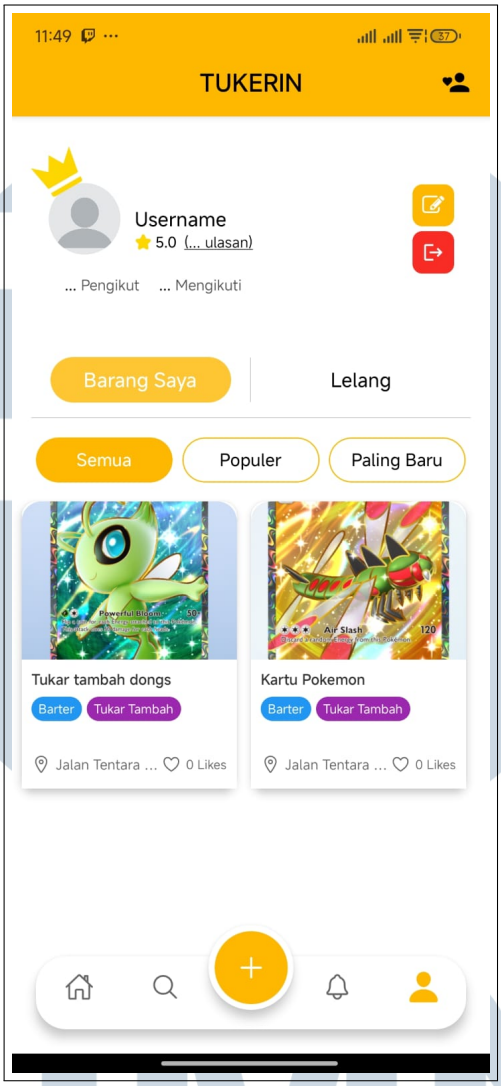
Implementasi teknis kolom input ini menggunakan komponen *TextInput*

yang sudah dikustomisasi sesuai dengan kebutuhan validasi dari masing-masing tipe data yang diinputkan. Ketika pengguna menekan tombol *"Register"*, aplikasi akan menjalankan fungsi yang bertanggung jawab mengirimkan data pendaftaran ke *endpoint backend* khusus registrasi. Validasi awal pada sisi *frontend* ini dimaksudkan untuk meminimalkan potensi error yang mungkin muncul dari sisi server. Jika proses registrasi sukses dilakukan, pengguna secara otomatis akan diarahkan kembali ke halaman *login* untuk masuk ke aplikasi.

Dari perspektif antarmuka, navigasi antara halaman *"Login"* dan *"Register"* dirancang dalam satu layar yang sama, menggunakan mekanisme *tab-switch* yang dikendalikan berdasarkan *state* yang aktif. Konsistensi gaya desain visual aplikasi Tukerin tetap dijaga dengan penggunaan warna dominan kuning untuk menegaskan aksi-aksi utama pengguna.

B.2 Pembuatan Halaman Profil Pengguna

Setelah melakukan penambahan barang, barang-barang tersebut dapat dilihat pada bagian halaman profil yang dapat diakses melalui ikon profil pada bagian kanan bawah *bottom navigation bar*. Halaman ini dirancang sebagai pusat kendali personal untuk pengguna agar dapat melihat informasi akun, barang milik pribadi, serta aktivitas lelang yang sedang atau telah dilakukan. Halaman ini dapat dilihat pada Gambar 3.10

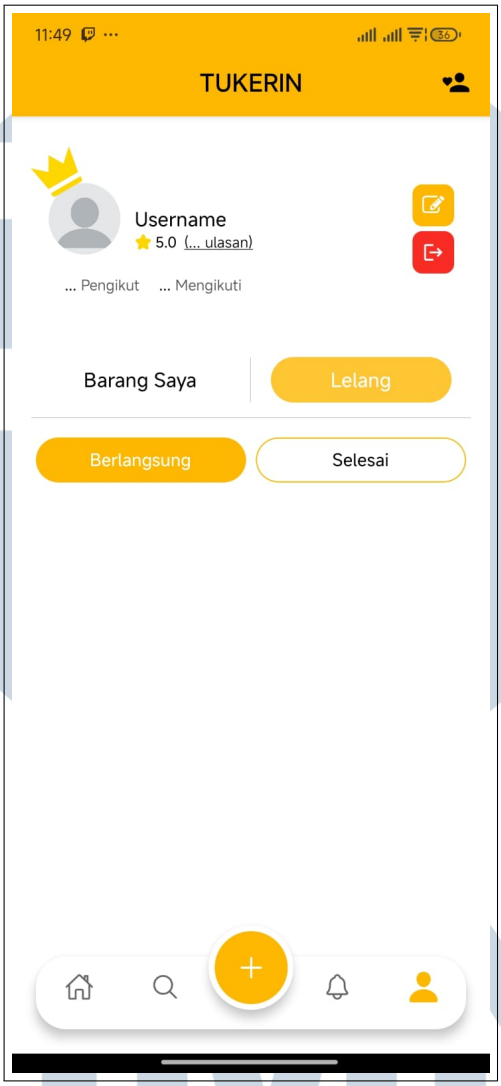


Gambar 3.10. Tampilan awal halaman profil pengguna

Pada bagian atas halaman profil, pengguna dapat melihat informasi dasar akun, seperti foto profil, nama pengguna, dan rating, jumlah pengikut, dan yang mengikuti, serta tombol untuk melakukan edit profil ataupun melakukan *logout*. Gambar mahkota pada halaman profil merupakan sebuah fitur yang sedang dirancang untuk menggambarkan seberapa bagus rating pengguna dalam melakukan kegiatan transaksi.

Pada bagian bawah informasi akun, terdapat dua *tab* utama, yaitu ”Barang Saya” dan ”Lelang”. Dalam *tab* ”Barang Saya” akan menampilkan semua daftar barang yang telah diunggah oleh pengguna tersebut. Barang-barang tersebut kemudian dibagi lagi ke dalam kategori semua, populer, serta paling baru. Barang-barang akan ditampilkan dalam bentuk kartu yang berisikan gambar, nama barang,

metode transaksi, lokasi, serta jumlah *likes*.



Gambar 3.11. Tampilan daftar barang pengguna dan tab lelang

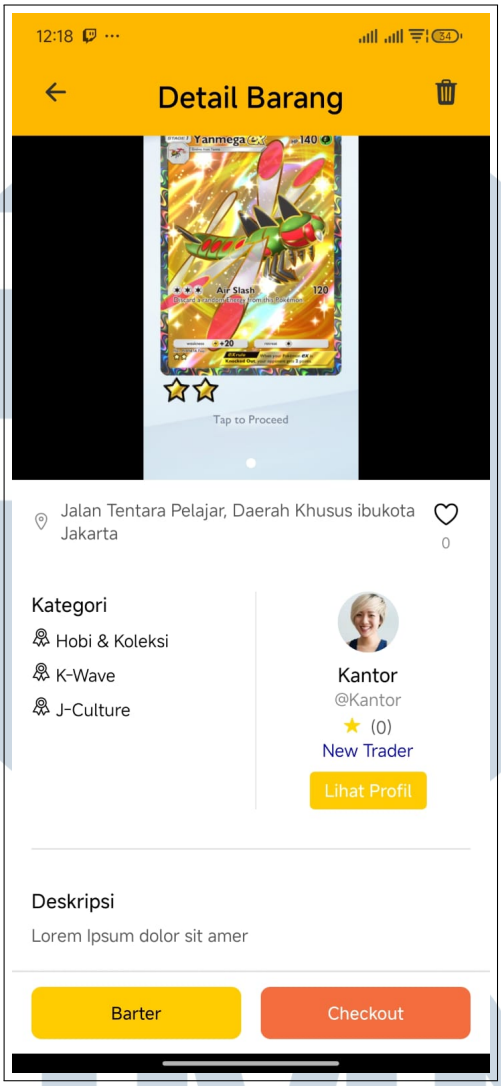
Sementara itu pada *tab* lelang, yang dapat dilihat pada Gambar 3.11, akan memisahkan barang berdasarkan statusnya, yakni berlangsung dan selesai, agar pengguna dapat dengan mudah untuk melakukan pemantauan terhadap proses barang-barang yang sedang dilelang. Pemisahan ini dilakukan untuk merapikan halaman profil agar lebih ter-organisir dan juga memudahkan pengguna untuk mengelola aktivitas mereka pada aplikasi ini.

Secara teknis, halaman ini dibangun dengan menggunakan *switch-tab* untuk melakukan pemindahan *tab* yang ada, berdasarkan *state* yang sedang aktif. Seluruh data pengguna, termasuk barang dan histori lelang, diambil melalui pemanggilan API terhadap *backend* server saat halaman dimuat. Namun dikarenakan lelang

untuk saat ini masih belum terbuat maka contoh dari barang-barang yang sedang di-lelang masih belum terlihat pada bagian *frontend* aplikasi.

B.3 Pembuatan Halaman Detail Barang

Halaman detail barang berfungsi untuk menampilkan informasi lengkap terkait barang yang diunggah oleh pengguna, baik barang milik pengguna lain maupun milik pengguna sendiri. Informasi yang disajikan meliputi gambar atau media barang yang ditampilkan secara visual di bagian atas halaman, lokasi barang, kategori produk, data pemilik barang, deskripsi produk secara lengkap, serta dua tombol utama yang terletak di bagian bawah halaman untuk pemilihan metode transaksi dan proses *checkout*. Tampilan halaman detail barang dapat dilihat pada Gambar 3.12.



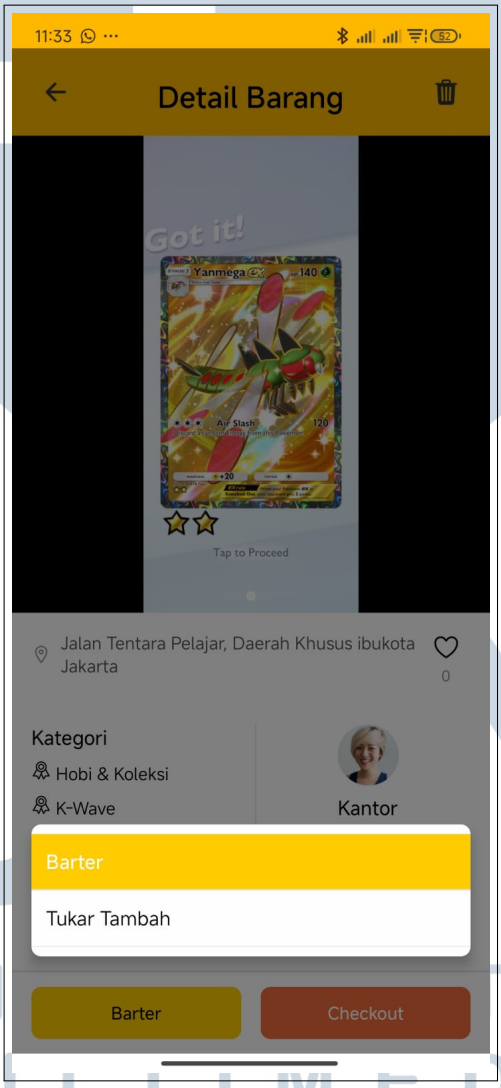
Gambar 3.12. Tampilan halaman detail barang

Pada bagian atas halaman ini, apabila terdapat beberapa gambar atau media terkait barang, pengguna dapat melakukan *swipe* untuk melihat media lainnya secara bergantian. Lokasi barang ditampilkan jelas di bawah media tersebut, disertai tombol untuk menyukai barang atau menambahkannya ke dalam *wishlist* yang terletak di sisi kanan layar. Berikutnya terdapat informasi pemilik barang secara rinci, yang mencakup nama pengguna, *username*, *rating*, serta tombol khusus untuk melihat profil pemilik barang.

Kategori produk juga disajikan dalam tampilan vertikal yang jelas, membantu pengguna untuk memahami jenis barang secara cepat dan akurat. Bagian terakhir adalah kolom deskripsi, yang memberikan detail mendalam mengenai kondisi barang, spesifikasi teknis, atau informasi lain yang relevan bagi

calon pembeli atau pengguna yang tertarik melakukan barter. Semua data yang ditampilkan pada halaman ini didapatkan dari API *backend*, berdasarkan ID barang yang dipilih dari halaman sebelumnya.

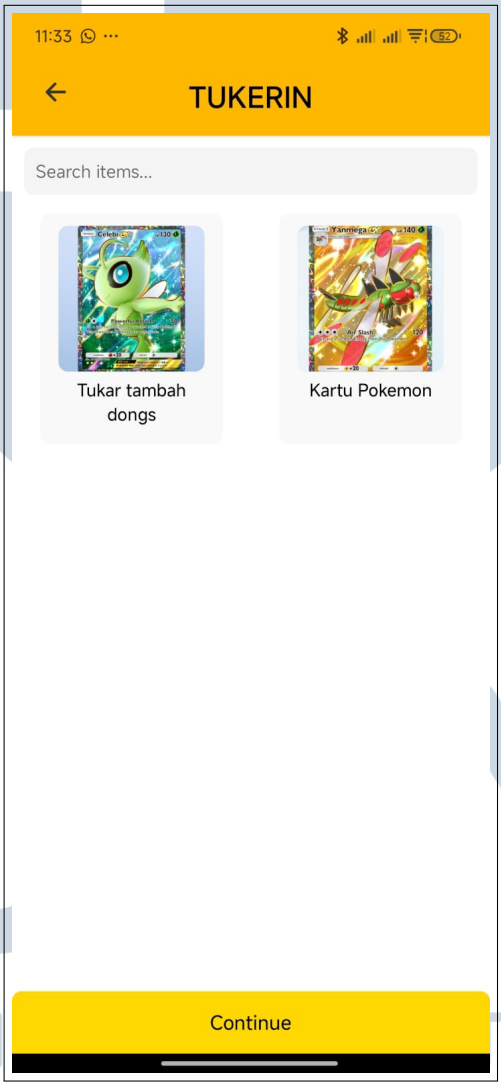
Jika barang tersebut memiliki beberapa opsi transaksi, pengguna dapat memilih tombol transaksi yang berada di kiri bawah. Saat tombol ini ditekan, aplikasi akan menampilkan sebuah *popup* untuk memilih metode transaksi yang diinginkan pengguna, seperti tampak pada Gambar 3.13.



Gambar 3.13. *Popup* pemilihan metode transaksi setelah menekan tombol Barter

Setelah pengguna memilih metode transaksi dan menekan tombol *checkout*, aplikasi akan mengarahkan pengguna ke halaman berikutnya, yang disesuaikan dengan jenis transaksi yang dipilih. Untuk transaksi barter atau tukar tambah, pengguna akan diarahkan langsung ke halaman pemilihan barang milik pengguna

sendiri, yang sebelumnya telah diunggah untuk ditawarkan dalam transaksi barter tersebut. Halaman pemilihan barang ini menampilkan daftar produk pengguna dalam bentuk kartu-kartu kecil yang dapat diklik untuk memilih barang yang ingin ditawarkan. Pengguna dapat memilih salah satu barang, lalu menekan tombol *continue* untuk melanjutkan proses transaksi. Tampilan halaman pemilihan barang ini diperlihatkan pada Gambar 3.14.



Gambar 3.14. Halaman pemilihan barang milik pengguna

Sebagai tambahan untuk memudahkan pencarian barang, pengguna juga dapat menggunakan fitur pencarian yang tersedia di bagian atas halaman ini. Pengguna cukup mengetikkan nama barang yang ingin dicari dalam kolom pencarian, dan halaman akan menampilkan hasil pencarian yang sesuai secara real-time.

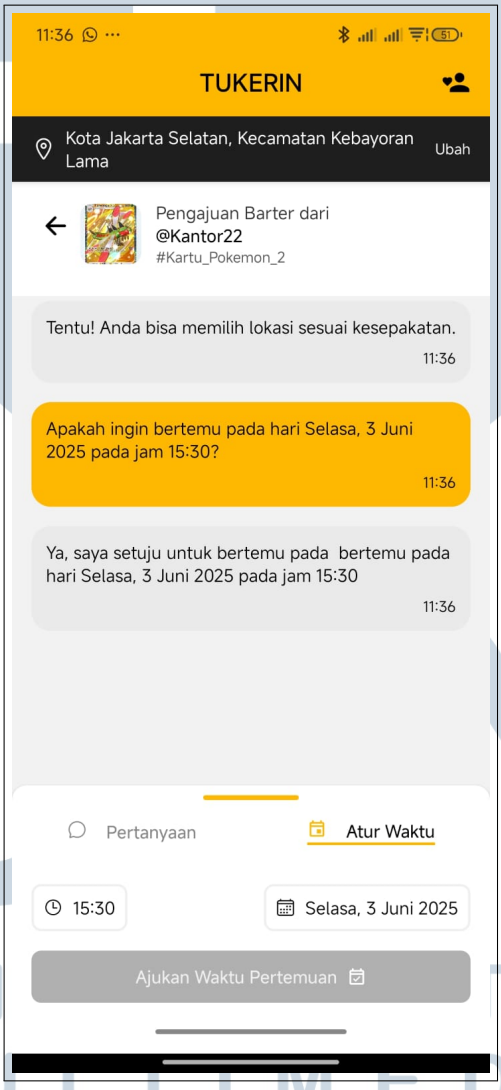
B.4 Pembuatan Prototype Chat

Halaman *chat* dalam aplikasi Tukerin digunakan sebagai sarana komunikasi antara kedua pengguna yang sedang terlibat dalam transaksi. Halaman ini tersedia di bagian bawah detail transaksi setelah pengguna memilih barang dalam proses *checkout* atau dapat diakses langsung dari notifikasi aplikasi. Melalui halaman ini, pengguna dapat melakukan komunikasi dan konfirmasi mengenai detail transaksi seperti lokasi pertemuan, jadwal, serta berdiskusi mengenai kondisi barang yang ditawarkan. Contoh tampilan halaman *chat* transaksi dapat dilihat pada Gambar 3.15.

Gambar 3.15. Tampilan halaman chat transaksi

Pada bagian atas layar, pengguna dapat dengan jelas melihat informasi

pengguna lain yang terlibat dalam transaksi, barang yang diajukan dalam penawaran, serta lokasi transaksi yang telah secara otomatis ditentukan berdasarkan lokasi barang yang dipilih. Fitur utama halaman ini adalah kolom percakapan, di mana kedua pihak dapat saling mengirim pesan secara real-time. Selain itu, halaman ini juga dilengkapi fitur khusus untuk menentukan waktu dan tanggal pertemuan serta fitur pesan cepat seperti ”Saya OTW” atau ”Kondisi barang bagus?”, yang dapat mempercepat dan memudahkan proses komunikasi pengguna.



Gambar 3.16. Fitur pengaturan waktu dan pesan cepat

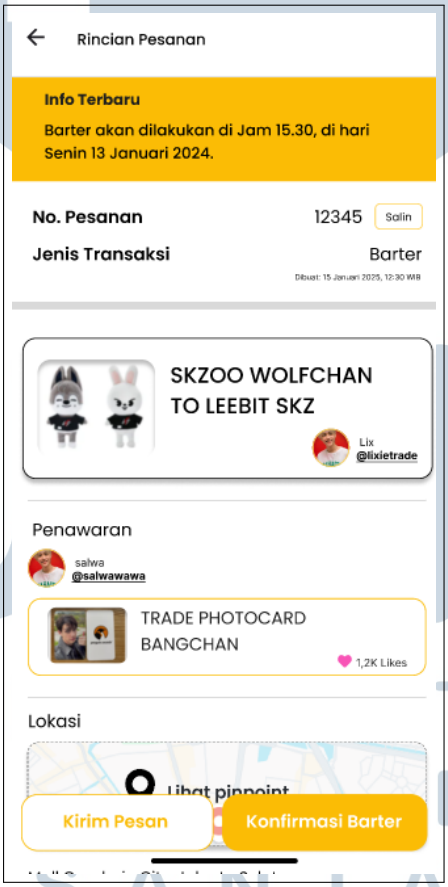
Dengan pertimbangan efisiensi pada tahap prototipe ini, lokasi transaksi telah ditentukan secara otomatis sesuai lokasi barang yang terdaftar, sehingga pengguna hanya perlu menentukan waktu dan tanggal untuk pertemuan. Ketika salah satu pihak mengusulkan waktu dan tanggal transaksi, pihak lain dapat

langsung menyetujui jadwal tersebut melalui pesan yang muncul di halaman chat.

Setelah kesepakatan terhadap waktu dan tanggal tercapai, status transaksi secara otomatis diperbarui menjadi *"Ongoing"*, yang menandakan bahwa transaksi tersebut siap untuk dilaksanakan secara langsung oleh kedua pengguna yang bersangkutan.

B.5 Pembuatan Halaman Konfirmasi Transaksi Barter

Halaman konfirmasi transaksi barter digunakan untuk menampilkan detail lengkap dari transaksi yang telah disepakati antar pengguna. Halaman ini dirancang dengan antarmuka yang *scrollable*, sehingga memungkinkan pengguna untuk melihat keseluruhan informasi transaksi secara lengkap, bahkan untuk transaksi dengan detail yang panjang. Tampilan halaman ini dapat dilihat pada Gambar 3.17.



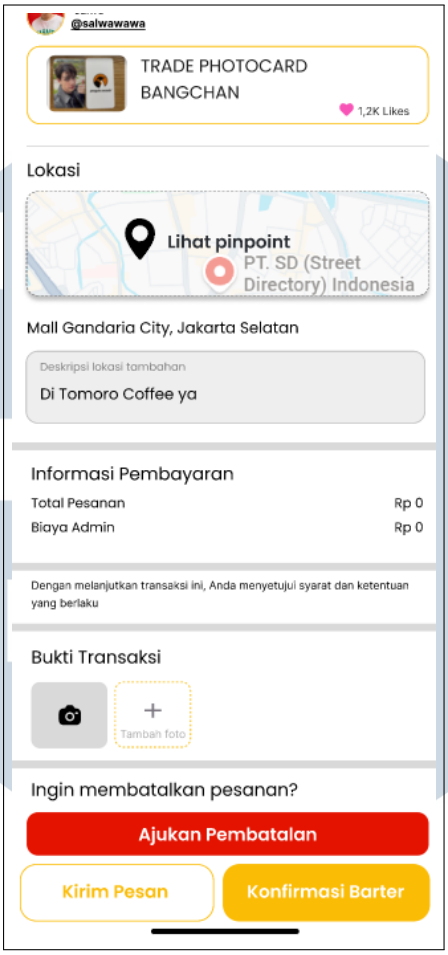
Gambar 3.17. Halaman rincian transaksi barter

Pada bagian atas halaman, pengguna dapat melihat informasi terbaru mengenai waktu pelaksanaan barter yang telah disepakati, termasuk tanggal dan

jam transaksi, serta nomor transaksi yang dapat disalin untuk referensi tambahan. Di bawahnya, barang yang diunggah oleh kedua belah pihak ditampilkan secara modular dalam bentuk kartu, lengkap dengan gambar barang, nama pengguna, serta jumlah *likes* yang menunjukkan popularitas barang tersebut di dalam aplikasi.

Selanjutnya, modul lokasi transaksi menampilkan tempat pertemuan yang telah disepakati. Secara bawaan, lokasi ini ditentukan berdasarkan lokasi barang, tetapi pengguna dapat dengan mudah menambahkan informasi detail tambahan seperti nama tempat atau landmark spesifik. Di bawah modul lokasi, terdapat modul informasi pembayaran yang secara modular dapat diubah atau ditambah di kemudian hari, misalnya jika ada transaksi khusus yang melibatkan biaya administrasi tambahan. Selain itu, modul untuk mengunggah bukti transaksi dalam bentuk foto juga tersedia sebagai dokumentasi tambahan yang dapat disesuaikan dengan kebutuhan.

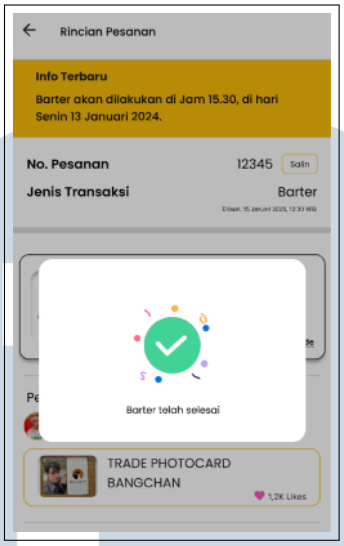
Tampilan lanjutan dari halaman rincian transaksi barter ini ditampilkan pada Gambar 3.18.



Gambar 3.18. Bagian lanjutan rincian transaksi barter

Pada bagian paling bawah halaman, terdapat tombol aksi utama yang juga bersifat modular, seperti ”Kirim Pesan” untuk komunikasi lanjutan melalui fitur chat, ”Konfirmasi Barter” untuk menyelesaikan transaksi, atau ”Ajukan Pembatalan” untuk membatalkan transaksi jika diperlukan. Hal ini memungkinkan penambahan tombol-tombol aksi lain jika diperlukan untuk fitur transaksi tambahan di masa mendatang.

Ketika pengguna menekan tombol ”Konfirmasi Barter”, aplikasi akan menampilkan notifikasi berupa *pop-up* modular sebagai indikasi bahwa transaksi barter tersebut telah selesai secara resmi, seperti yang dapat dilihat pada Gambar 3.19.



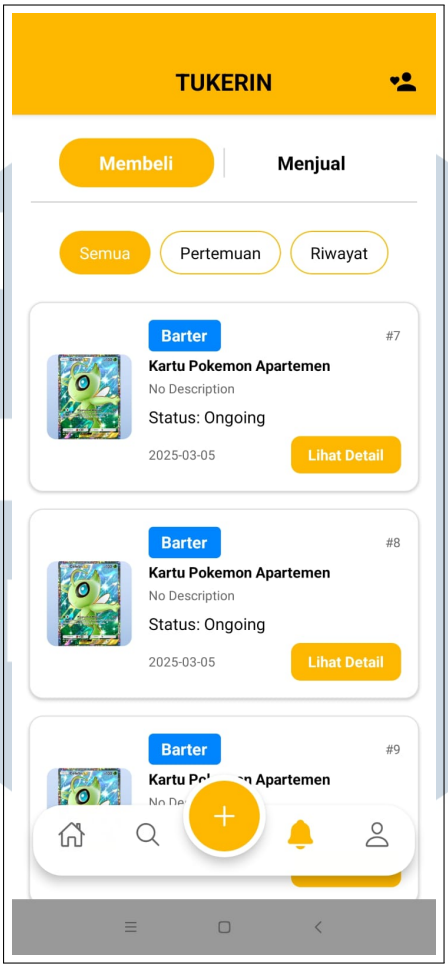
Gambar 3.19. Notifikasi konfirmasi penyelesaian transaksi barter

Dengan desain yang modular dan *scrollable*, halaman ini tidak hanya meningkatkan kenyamanan pengguna tetapi juga memberikan fleksibilitas tinggi untuk pengembangan aplikasi dalam menambahkan berbagai modul atau detail baru terkait transaksi sesuai dengan kebutuhan.

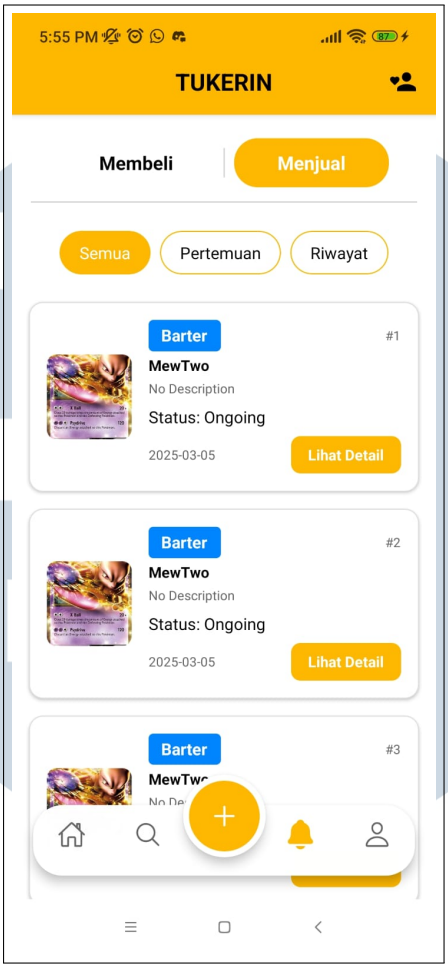
B.6 Pembuatan Halaman Notifikasi Transaksi

Halaman notifikasi transaksi pada aplikasi Tukerin berfungsi sebagai pusat informasi terkait aktivitas transaksi yang dilakukan oleh pengguna, baik dalam posisi pembeli maupun penjual. Halaman ini dirancang untuk memberikan kemudahan kepada pengguna dalam memantau transaksi yang sedang berlangsung (*ongoing*), menjadwalkan pertemuan, serta melihat riwayat transaksi yang telah selesai.

Tampilan halaman notifikasi transaksi terdiri dari dua tab utama, yaitu tab ”Membeli” untuk aktivitas sebagai pembeli, dan tab ”Menjual” untuk aktivitas sebagai penjual. Masing-masing tab tersebut dapat difilter kembali berdasarkan status transaksi seperti ”Semua”, ”Pertemuan”, dan ”Riwayat”, yang memberikan kemudahan pengguna dalam mengelola transaksi secara efektif. Contoh tampilan dari halaman ini ditunjukkan pada Gambar 3.20 dan Gambar 3.21.



Gambar 3.20. Tampilan notifikasi transaksi pada tab ”Membeli”



Gambar 3.21. Tampilan notifikasi transaksi pada tab ”Menjual”

Setiap transaksi pada halaman ini ditampilkan dalam bentuk kartu yang mencakup informasi penting seperti gambar barang, judul barang, nomor transaksi, jenis transaksi (misalnya ”Barter”), status terkini (*ongoing* atau selesai), dan tombol ”Lihat Detail” yang memungkinkan pengguna mengakses informasi lebih mendalam terkait transaksi tersebut. Desain ini juga dibuat modular agar dapat dengan mudah dikembangkan atau ditambahkan jenis informasi lain, seperti status tambahan atau fitur-fitur baru yang relevan dengan transaksi. Dengan adanya halaman notifikasi transaksi ini, pengguna dapat secara efisien memantau dan mengelola berbagai aktivitas transaksi yang dilakukan di dalam aplikasi, sehingga meminimalkan risiko kesalahan atau kelalaian dalam menjalankan transaksi yang sedang berjalan maupun yang telah selesai.

C Testing Aplikasi

Dalam rangka memastikan aplikasi Tukerin memenuhi kebutuhan fungsional dan memberikan pengalaman pengguna yang baik, pengujian dilakukan melalui dua pendekatan utama: *Functional Testing* dan *Usability Testing*.

C.1 Functional Testing

Pengujian fungsional bertujuan memverifikasi bahwa setiap fitur aplikasi berfungsi sesuai spesifikasi. Pengujian ini mencakup berbagai endpoint API dan fitur inti yang dikembangkan oleh penulis dan tim, antara lain:

1. **API Item**: Operasi *Create, Read, Update, Delete* (CRUD) pada item telah berhasil diuji dan berjalan optimal.
2. **API Transaksi**: Fitur transaksi mencakup input data dan pencatatan detail barang yang diperjualbelikan atau ditukar berjalan sesuai harapan.
3. **API Autentikasi**: Mekanisme login dan validasi token telah lolos pengujian skenario yang dirancang.
4. **API Registrasi**: Proses pendaftaran akun baru berjalan lancar dan data tersimpan dengan benar di basis data.

Secara keseluruhan, semua fitur utama dan tambahan dinyatakan berfungsi sesuai kebutuhan aplikasi.

C.2 Usability Testing

Pengujian kenyamanan pengguna berfokus pada kemudahan interaksi dan pemahaman antarmuka aplikasi, diuji secara internal oleh sejumlah pengguna yang menjalankan skenario transaksi seperti barter. Evaluasi menunjukkan antarmuka aplikasi cukup informatif dan mudah dipelajari, dengan tata letak yang memudahkan pengguna mengenali fungsi utama. Namun, beberapa pengguna mengalami kesulitan memahami alur transaksi barter, khususnya pada bagian konfirmasi dan penyelesaian transaksi.

Temuan ini mengindikasikan bahwa selain aspek visual, perlu diperbaiki panduan alur transaksi dan kejelasan interaksi antar pengguna.

Perlu dicatat bahwa pengujian ini baru dilakukan pada perangkat Android, sehingga hasil evaluasi pada perangkat iOS belum tersedia dan memerlukan pengujian lanjutan di masa mendatang.

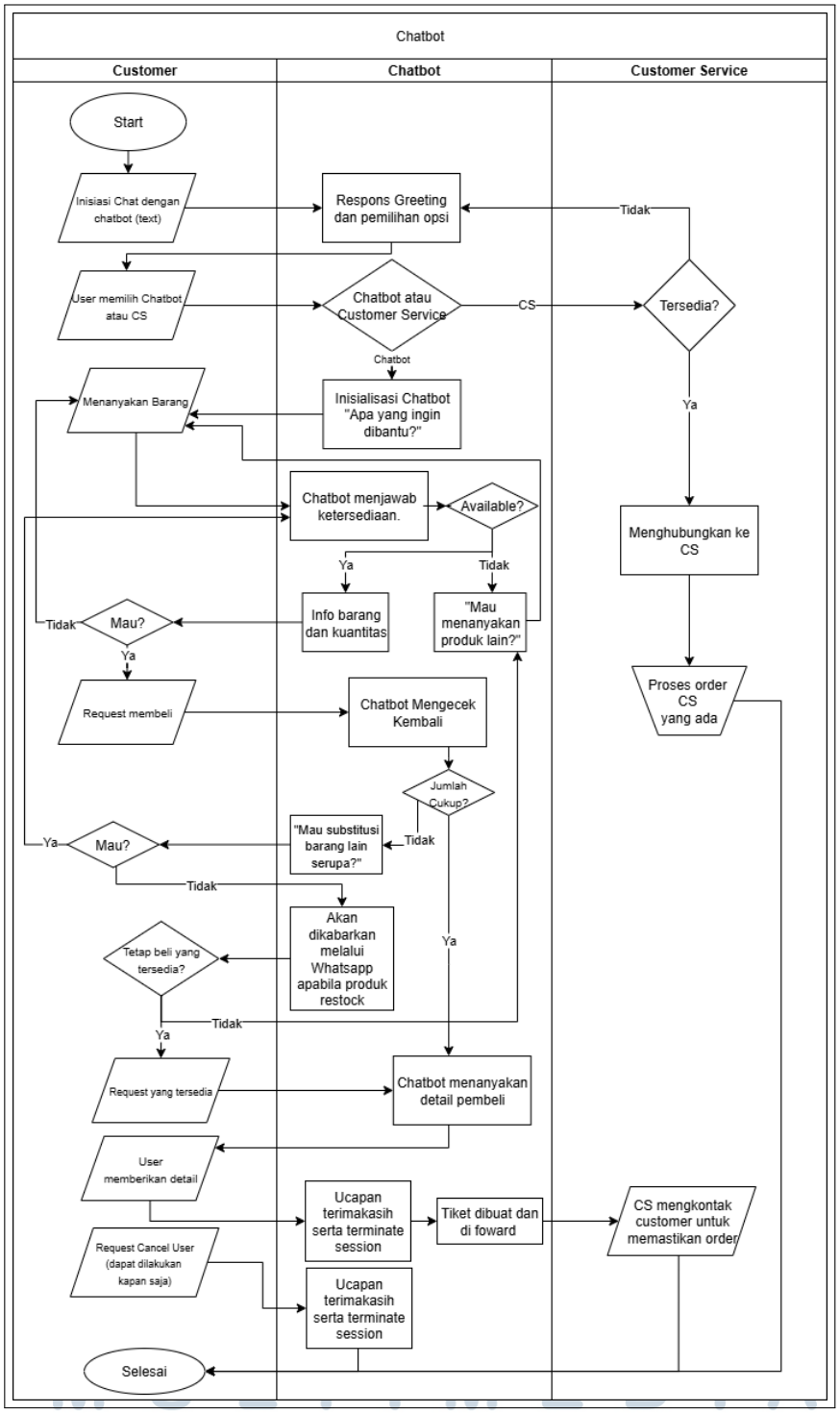
3.3.2 Chatbot

Pengembangan *chatbot* via WhatsApp dan Telegram pada proyek ini difokuskan untuk menggunakan platform WhatsApp terlebih dahulu yang memiliki tujuan akhir untuk menangani transaksi atau pertanyaan terkait satu kategori produk, dalam hal ini adalah barang barang *sparepart* kendaraan. Disini telah diputuskan untuk mengecilkan kategorinya terlebih dahulu, yaitu berupa oli motor, tujuan dari pembatasan kategori ini adalah untuk mempermudah perancangan alur percakapan, sekaligus memastikan bahwa interaksi chatbot dapat berjalan lancar pada skenario yang lebih spesifik sebelum diperluas ke kategori produk lainnya.

Implementasi chatbot ini dibangun sepenuhnya di atas ekosistem *Python*. Lapisan antarmuka dengan WhatsApp memanfaatkan *WhatsApp Business Cloud API* Diakses melalui penyedia gateway *Twilio*., sedangkan Telegram menggunakan `python-telegram-bot`. Seluruh logika layanan dijalankan di *FastAPI* yang dikemas dalam *Docker* dan diatur orkestrasinya melalui *Docker Compose*. Untuk penyimpanan embedding dipilih *FAISS* (Facebook AI Similarity Search) yang diisi vektor dari `sentence-transformers` (*all-MiniLM-L6*). Generasi respons bahasa alami didelegasikan ke *OpenAI GPT-4o* melalui pustaka `langchain`. Integrasi sistem penyimpanan dan pembuatan *invoice* menggunakan *Odoo XML-RPC API* dengan cache transaksi singkat di *Redis*. Seluruh komponen dideploy pada server Ubuntu 22.04 di jaringan internal, dengan *Nginx* bertindak sebagai *reverse proxy* dan terminasi TLS.

A Alur Chatbot dan Infrasturktur

Penulis sendiri tidak secara intensif terlibat dalam keseluruhan proses implementasi teknis chatbot ini, melainkan lebih banyak berkontribusi pada tahap desain alur (*flow design*) percakapan chatbot, serta tahap persiapan *setup* FAISS sebagai salah satu komponen teknisnya. Alur desain yang telah disusun mencakup proses awal percakapan, penanganan pertanyaan oleh chatbot, pengecekan stok atau ketersediaan produk, hingga pembuatan tiket atau *invoice*, seperti yang terlihat pada Gambar 3.24.

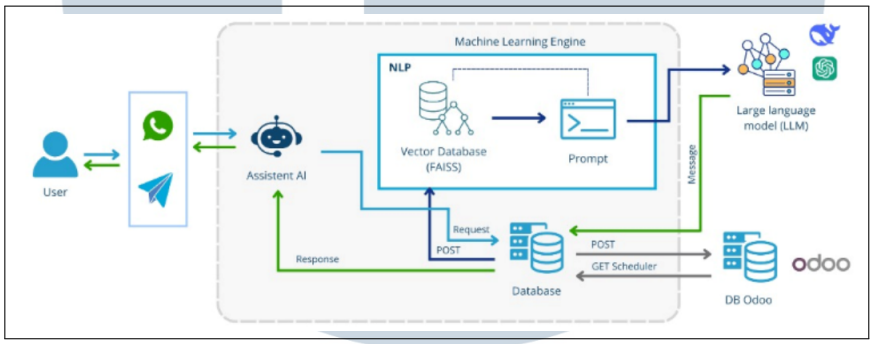


Gambar 3.22. Swimlane diagram chatbot pada WhatsApp untuk melakukan pemesanan

Hingga saat ini, chatbot sudah mampu melakukan percakapan hingga tahap pembuatan *invoice*. Namun, integrasi langsung untuk pengalihan percakapan kepada *Customer Service* (CS) melalui WhatsApp masih belum

sepenuhnya diselesaikan karena fitur pengalihan manual tersebut masih dalam tahap pengembangan. Meskipun demikian, desain chatbot ini dibuat secara modular dan fleksibel agar dapat dengan mudah diperluas ke berbagai jenis kategori produk lainnya di masa depan.

Arsitektur chatbot yang digunakan dalam proyek ini dirancang untuk menangani interaksi secara otomatis dengan pengguna melalui kanal seperti WhatsApp dan Telegram. Secara umum, alur chatbot ini dirancang untuk menerima pesan dari pengguna, memproses pertanyaan atau permintaan tersebut melalui sistem berbasis *Machine Learning*, dan memberikan respons yang sesuai. Diagram arsitektur chatbot dapat dilihat pada Gambar 3.23.



Gambar 3.23. Diagram Arsitektur Infrastruktur Chabot yang digunakan [3]

Dari diagram tersebut, proses alur chatbot dapat dijelaskan secara lebih rinci sebagai berikut:

A.1 Interaksi Pengguna dengan Chatbot

Pengguna memulai interaksi dengan chatbot melalui platform komunikasi seperti WhatsApp atau Telegram. Pesan yang dikirimkan pengguna akan langsung diteruskan menuju sistem chatbot yang bertindak sebagai *Assistant AI*.

A.2 Assistant AI

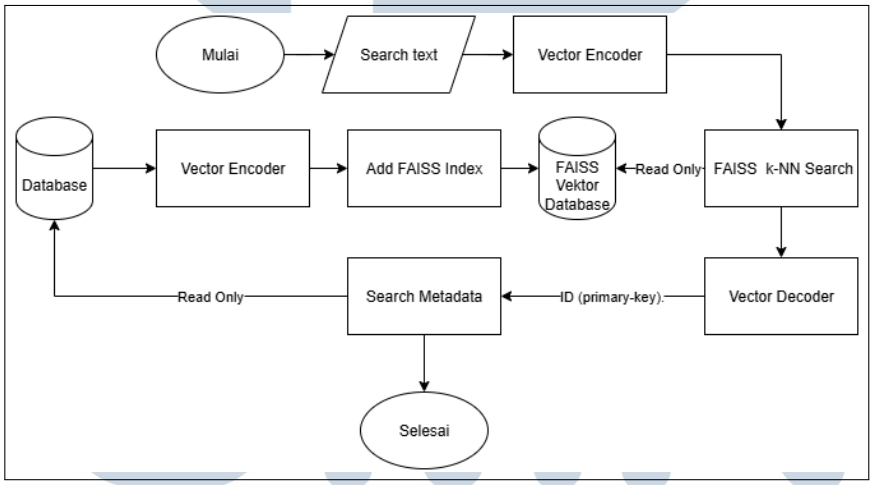
Assistant AI bertindak sebagai penghubung antara pengguna dengan sistem internal chatbot. Pesan yang diterima Assistant AI akan diteruskan menuju mesin pemrosesan utama berbasis *Machine Learning Engine* untuk diolah lebih lanjut.

A.3 Machine Learning Engine (MLE)

Komponen inti dari chatbot ini adalah *Machine Learning Engine*, yang bertanggung jawab untuk memahami maksud pengguna dan menghasilkan jawaban yang relevan. Komponen ini terdiri dari dua bagian utama, yaitu:

A.3.1 Vector Database (FAISS)

Sebagai bagian dari tahap persiapan teknis chatbot, penulis juga turut berkontribusi dalam konfigurasi FAISS (*Facebook AI Similarity Search*), sebuah pustaka yang digunakan untuk menyimpan dan mencari vektor embedding secara efisien. FAISS dipilih karena kemampannya dalam melakukan pencarian vektor yang cepat dan scalable, yang sangat membantu dalam mempercepat proses pencarian informasi atau produk yang relevan dengan pertanyaan pengguna dalam chatbot. Dengan menggunakan FAISS, chatbot dapat memberikan respon yang lebih cepat dan akurat berdasarkan data yang tersedia.[7]



Gambar 3.24. Alur kerja FAISS

A.3.2 Prompt Generation

Setelah menemukan data yang relevan dari FAISS, data tersebut akan digunakan sebagai prompt atau konteks untuk dikirimkan ke model bahasa besar (*Large Language Model/LLM*) seperti GPT, yang akan menghasilkan respons teks secara natural kepada pengguna.

A.3.3 Integrasi dengan Large Language Model (LLM)

Setelah mendapatkan data relevan dari FAISS, informasi tersebut disusun dalam bentuk prompt yang dikirimkan ke LLM. Model bahasa besar ini bertugas untuk menghasilkan respons teks alami yang kemudian dikirim kembali ke Assistant AI. Integrasi dengan LLM memungkinkan chatbot memberikan respons yang informatif, tepat, serta ramah dalam berkomunikasi dengan pengguna.

A.4 Interaksi dengan Sistem Database Internal

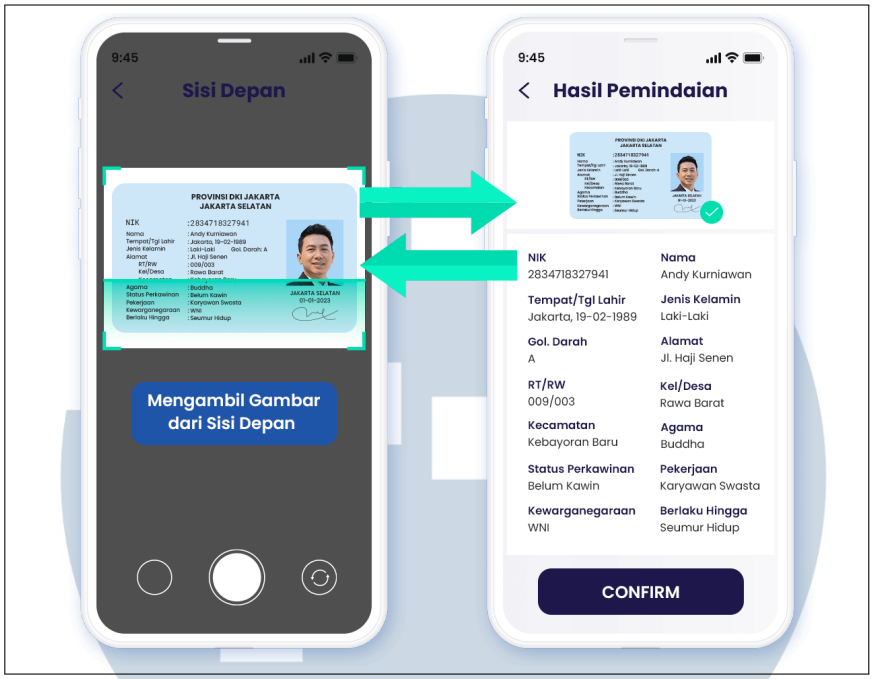
Chatbot juga diintegrasikan dengan database internal, seperti database Odoo. Ketika chatbot membutuhkan informasi tambahan atau harus menyimpan data transaksi (misalnya membuat invoice atau mencatat permintaan pengguna), chatbot akan mengirimkan permintaan (*request*) berupa metode POST atau GET ke database internal. Database kemudian merespon dengan data yang diperlukan atau mencatat transaksi yang dilakukan oleh pengguna.

A.5 Respons kepada Pengguna

Setelah melewati proses pemrosesan di atas, Assistant AI akan mengirimkan respons akhir kepada pengguna melalui kanal komunikasi yang digunakan sebelumnya (WhatsApp atau Telegram). Dengan demikian, pengguna mendapatkan informasi yang akurat dan tepat waktu.

3.3.3 Proyek OCR Finance

Optical Character Recognition (OCR) merupakan teknologi yang memungkinkan komputer untuk mengenali dan mengekstrak teks dari gambar atau dokumen fisik, kemudian mengubahnya menjadi data digital yang dapat diolah lebih lanjut oleh komputer. OCR menggunakan berbagai metode *image processing* dan *Artificial Intelligence/AI* untuk mendeteksi pola karakter dalam gambar secara otomatis, termasuk huruf, angka, maupun simbol. Proses ini mencakup beberapa tahapan utama seperti pra-pemrosesan gambar, segmentasi teks, ekstraksi karakter, dan pengenalan teks melalui algoritma atau model *machine learning*. [8]



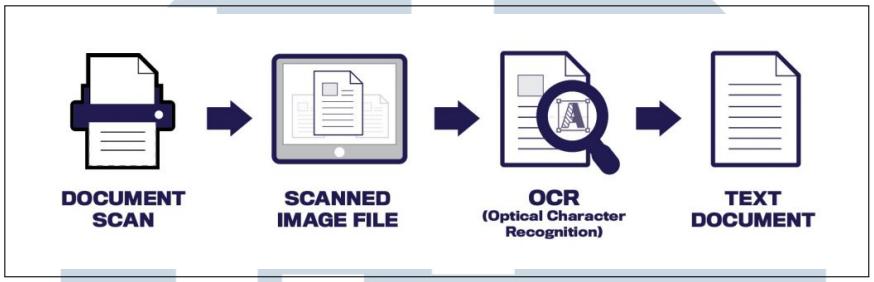
Gambar 3.25. Salah satu pengunaan OCR yang sudah ada di indonesia [9]

Teknologi OCR secara khusus sangat bermanfaat dalam bidang keuangan dan akuntansi untuk mempercepat proses digitalisasi berbagai dokumen seperti invoice, struk pembayaran, bukti transaksi, atau laporan keuangan lainnya yang biasanya dikelola secara manual. Dengan menggunakan OCR, data dari dokumen-dokumen fisik ini dapat dengan mudah dan cepat dikonversi menjadi format digital sehingga dapat diintegrasikan langsung dengan sistem manajemen dokumen maupun sistem keuangan perusahaan. Hal ini sangat mengurangi waktu dan tenaga yang biasanya diperlukan dalam proses input manual serta meminimalisir risiko kesalahan manusia dalam pencatatan data.

Seluruh *pipeline* OCR Finance dibangun di atas ekosistem *Python*. Lapisan layanan memakai *FastAPI* dengan *Uvicorn* sebagai ASGI server, dikemas di dalam *Python virtual environment*. Inferensi dijalankan oleh model *DONUT* yang telah *fine-tuned* menggunakan pustaka `transformers` (*Hugging Face*) dan *PyTorch Lightning*. Karena server development hanya berbasis CPU, optimalisasi dilakukan melalui `torch.compile()` serta *mixed-precision*. Pra-pemrosesan gambar ditangani dengan *OpenCV* dan *imgaug*, sedangkan tahap *post-processing*—validasi entitas—memanfaatkan *regex*, *rapidfuzz*, dan *pydantic* untuk normalisasi skema. Metadata dokumen disimpan di *PostgreSQL*.

A Document OCR

Untuk menggambarkan bagaimana OCR bekerja secara umum, ilustrasi proses OCR dapat dilihat pada Gambar 3.27.



Gambar 3.26. Ilustrasi proses umum *Optical Character Recognition* (OCR) [8]

Implementasi OCR membutuhkan rangkaian komponen teknis mulai dari proses *image-acquisition* (pengambilan gambar menggunakan kamera atau scanner) hingga integrasi hasil ekstraksi ke sistem keuangan. Pertama, pengambilan gambar harus konsisten: kamera ponsel dengan sensor 12 MP sudah memadai selama pencahayaan stabil, tetapi untuk digitalisasi massal disarankan flatbed scanner 300 dpi agar noise dan distorsi perspektif minimal [10]. Setelah dokumen dipindai atau difoto, file gambar memasuki tahap *pre-processing*—biasanya menggunakan pustaka OpenCV (library C/C++ lintas-platform untuk pengolahan gambar)—mencakup normalisasi ukuran, peningkatan kontras, binarisasi adaptif, koreksi kemiringan, *denoise*, hingga *deskew*. Kualitas hasil pra-pemrosesan sangat menentukan akurasi pengenalan karakter berikutnya [11].

Inti pipeline berada pada *engine OCR*. pada umumnya kita akan menggunakan, engine konvensional seperti *Tesseract* (OCR open-source dari Google), *PaddleOCR* (framework OCR berbasis PaddlePaddle), atau *EasyOCR* (wrapper PyTorch untuk lebih dari 80 bahasa) menjalankan dua tahap inti: *text-detection* dan *text-recognition*. Pada tahap deteksi, algoritma modern seperti CRAFT (*Character Region Awareness for Text Detection*) [12] atau EAST *(Efficient and Accurate Scene Text Detector*) [13] memindai area teks dalam gambar. Selanjutnya, tahap pengenalan menggunakan model CRNN (*Convolutional Recurrent Neural Network*) atau TensorFlow-Seq2Seq untuk mengubah potongan gambar huruf menjadi karakter ASCII/Unicode. GPU (seperti NVIDIA RTX dengan *CUDA*) dapat membuat proses ini lebih cepat, tetapi pada volume kecil, CPU multicore sudah cukup [14].

Gambar 3.27. beberapa OCR Tools yang dapat digunakan [15]

Hasil mentah engine diproses ulang dalam *post-processing*. Tahap post-processing bertujuan membersihkan hasil mentah dari engine OCR. Di sini, regular expressions digunakan untuk menyeleksi entitas tertentu seperti pola "INV-" pada nomor invoice, sementara fuzzy matching, misalnya dengan algoritma Levenshtein distance atau struktur data seperti BK-tree, digunakan untuk memperbaiki kesalahan ejaan akibat kegagalan deteksi OCR (Klippa, 2023; Encord, 2023). Kemudian, language-model verification menggunakan teknik n-gram atau model seperti BERT memastikan bahwa frasa yang dihasilkan logis dalam konteks dokumen. Setelah data tervalidasi, hasilnya dikemas ke dalam format JSON dan dikirimkan melalui REST atau GraphQL API ke backend[10].

Di sisi backend, layanan semisal FastAPI (framework Python), *Node.js* (runtime JavaScript server-side), atau *Java Spring* (framework enterprise Java) menyimpan metadata di PostgreSQL/MySQL (relasional DBMS open-source). File gambar disimpan di MinIO / Amazon S3 (layanan objek-storage) ataupun secara lokal apabila skala dan kebijakan keamanan dapat terpenuhi (contohnya data sensitif), sedangkan indeks pencarian cepat—misalnya untuk mendeteksi duplikasi dokumen—bisa menggunakan FAISS (*Facebook AI Similarity Search*).

B Model DONUT (Document Understanding Transformer)

DONUT (*Document Understanding Transformer*) adalah model *end-to-end* berbasis arsitektur transformer yang diciptakan oleh NAVER AI Lab untuk menafsirkan dokumen gambar—invoice, kuitansi, formulir, laporan—tanpa melewati tahapan OCR biasa (deteksi → pengenalan → ekstraksi) secara terpisah. seperti mendeteksi baris, mengenali karakter, kemudian mengekstrak entitas dalam modul terpisah, DONUT langsung memetakan piksel dokumen ke keluaran teks

terstruktur (JSON) dalam satu kali inferensi. Dengan kata lain, ia "melewati" lapisan OCR tradisional dan bekerja layaknya mesin penerjemah yang mengonversi dokumen visual menjadi representasi semantik [16].

Dari sudut pandang penerapan, pipeline DONUT tetap memerlukan langkah *pre-processing* dasar—misalnya koreksi kemiringan atau peningkatan kontras sebagaimana pada OCR—namun setelah gambar dinormalisasi, seluruh proses deteksi, pengenalan, dan penataan informasi diurus oleh satu model. Tidak ada fase *text-detection* (CRAFT, EAST) atau *text-recognition* (Tesseract, CRNN) yang berjalan terpisah; sehingga rantai kesalahan antarmodul dapat dihindari. Selain itu, karena outputnya sudah berbentuk JSON, lapisan *post-processing* rule-based jauh berkurang—cukup validasi ringan (format tanggal, mata uang) sebelum data disimpan di dalam database.

B.1 Arsitektur Inti

DONUT terdiri atas *Vision Transformer* (ViT) *encoder* yang memecah gambar ke dalam patch, lalu mengonversinya menjadi embedding visual, dan *autoregressive transformer decoder* yang menuliskan token JSON satu per satu. Selama pelatihan, tiap dokumen diberi label berupa file JSON yang memuat labeling (`"invoice_number"`, `"date"`, `"amount"`, dan sebagainya) yang lengkap dengan nilai aslinya. Proses *fine-tuning* mengajarkan decoder memadankan pola visual khusus—misalnya kolom total rupiah—dengan token JSON yang tepat. Mekanisme ini membuat model *layout-aware* dan tetap *robust* ketika menghadapi variasi format [17].

B.1.1 Rincian Encoder–Decoder DONUT

Encoder

gambar ($H \times W$ piksel) diubah ke resolusi tetap—misalnya $1\,024 \times 1\,024$—lalu dipotong menjadi patch berukuran $P \times P$ (umumnya $P = 16$). Setiap patch di-flatten, diproyeksikan secara linear, dan dijumlah dengan positional-encoding sehingga model mengetahui posisi spasialnya. Seluruh urutan patch, ditambah token `[CLS]`, diproses oleh beberapa blok *Vision Transformer* (multi-head self-attention + feed-forward). Hasil akhirnya adalah matriks fitur $N \times D$ yang merepresentasikan konteks visual dan tata letak setiap bagian dokumen [16][18]. Untuk rumus Encoder yang digunakan adalah sebagai berikut:

$$N = \left(\frac{H}{P}\right)\left(\frac{W}{P}\right) \tag{3.1}$$

$$\mathbf{z}_i = \mathbf{x}_i\mathbf{W}_e + \mathbf{E}_{\text{pos},i}, \qquad i = 1,\ldots,N \tag{3.2}$$

Decoder

Decoder bersifat autoregresif dengan dua mekanisme atensi: *masked self-attention* (token hanya melihat token sebelumnya) dan *cross-attention* (mengakses embedding hasil encoder). Dimulai oleh token `[BOS]`, pada tiap langkah t decoder menghitung distribusi token berikut $p(y_t \mid y_{<t}, \mathbf{H})$ dan memilih token probabilitas tertinggi (*greedy*) atau *multiple candidate* (*beam search*). Karena target keluaran berbentuk JSON[16][19], model dilatih memakai *teacher-forcing* pada string anotasi seperti:

```
{"invoice_number":"INV123",
 "date":"2025-06-30",
 "amount":"Rp1500000"}
```

Untuk rumus Dekoder yang digunakan DONUT adalah sebagai berikut:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_k}}\right) \tag{3.3}$$

$$\mathbf{H} = \mathbf{A}\mathbf{V} \tag{3.4}$$

$$p(y_t \mid y_{<t}, \mathbf{H}) = \text{softmax}\big(\mathbf{W}_o\mathbf{h}_t + \mathbf{b}_o\big) \tag{3.5}$$

$$y_t = \arg\max_k p_k, \qquad t = 1,2,\ldots,T \tag{3.6}$$

$$\mathscr{L} = -\sum_{t=1}^{T} \log p\big(y_t^* \mid y_{<t}^*, \mathbf{H}\big) \tag{3.7}$$

Dengan desain ini, DONUT memahami konteks spasial melalui patch-embedding + self-attention, membaca isi tabel atau stempel, dan langsung menghasilkan JSON terstruktur tanpa *post-processing* berbasis aturan.

```
<bos> { "invoice_number": "INV123",
 "date": "2025-06-30",
 "amount": "Rp1500000" } <eos>
```

Selama inferensi, strategi *greedy decoding* atau *beam search* (*beam width* 3–5) digunakan untuk menjaga konsistensi tanda kurung dan kunci JSON. Arsitektur ini membuat DONUT mampu:

1. Memahami konteks spasial (melalui patch embedding + self-attention).
2. "Membaca" isi tabel, cap, atau logo sekaligus mengasosiasikannya dengan label semantik JSON.
3. Menghasilkan keluaran terstruktur tanpa pasca-proses berbasis aturan.

B.2 Keunggulan Praktis

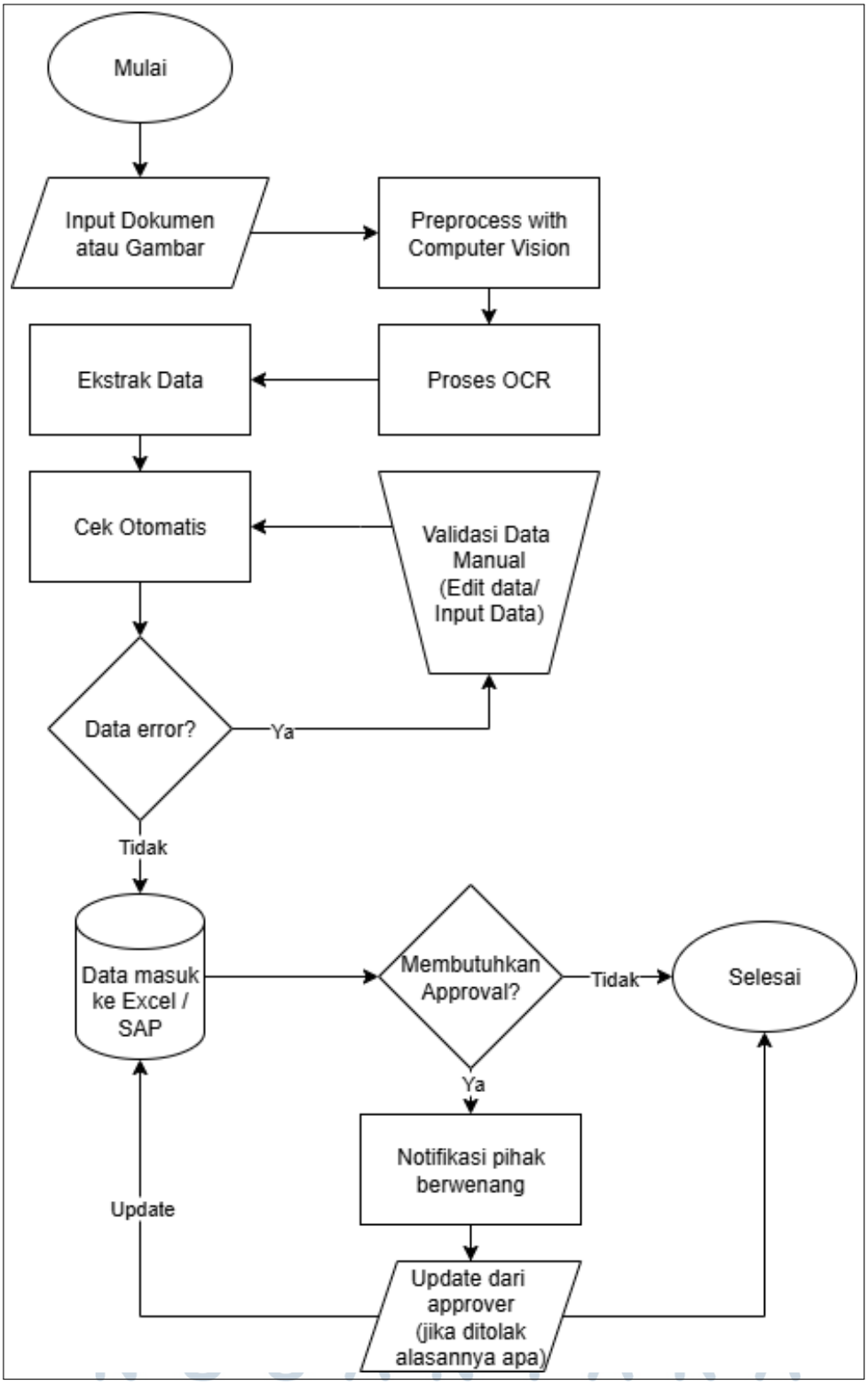
DONUT mengurangi latensi karena hanya sekali inferensi GPU (kurang dari 1 detik apabila menggunakan CUDA RTX). Akurasi tetap tinggi pada dokumen multi-kolom, tabel, atau stempel karena konteks spasial dipelajari oleh encoder. Output terstruktur menghemat waktu integrasi ke PostgreSQL, SAP, atau spreadsheet tanpa parsing regex rumit. Selain itu, ketika kualitas file gambar menurun (noise, blur ringan), konteks global yang dipelajari decoder menolong model tetap menebak nilai dengan tepat, sedangkan OCR tradisional sering gagal merangkai karakter.

Keluaran tersebut siap dimasukkan ke endpoint REST atau langsung di-`INSERT` ke tabel transaksi. Melalui pendekatan *OCR-free*, DONUT memberikan jalur yang lebih ringkas, presisi tinggi, dan minim *glue-code* bagi digitalisasi dokumen finansial dalam proyek ini.

C Rencana implementasi OCR

C.1 Workflow OCR

Alur kerja aplikasi OCR ini dirancang secara rinci untuk memastikan bahwa setiap dokumen dapat diproses dengan efisien dan menghasilkan data yang akurat. Workflow lengkapnya ditampilkan pada Gambar 3.28.



Gambar 3.28. Flowchart proses OCR dalam pengelolaan dokumen finansial

Penjelasan rinci dari masing-masing tahapan adalah sebagai berikut:

C.1.1 Input Dokumen atau Gambar

Proses dimulai ketika pengguna memasukkan dokumen dalam bentuk digital, seperti foto atau hasil scan dari dokumen fisik. Jenis dokumen yang dapat diproses meliputi struk pembayaran, invoice, faktur, dan dokumen finansial lainnya.

C.1.2 Pra-Pemrosesan dengan Computer Vision

Tahap ini melibatkan berbagai teknik pengolahan gambar untuk meningkatkan kualitas input sebelum diproses oleh OCR. Tahapannya termasuk:

Normalisasi Resolusi: Gambar disesuaikan dengan resolusi optimal. **Noise Removal**: Menghilangkan noise atau distorsi gambar. **Peningkatan Kontras**: Membantu memperjelas teks dalam gambar. **Koreksi Orientasi**: Menyesuaikan posisi teks agar lurus dan mudah dibaca OCR.

C.1.3 Proses OCR

Pada tahap ini, teknologi OCR akan bekerja untuk mengekstrak teks dari gambar yang telah diproses sebelumnya. Sistem OCR akan mengenali pola-pola karakter dalam dokumen menggunakan metode AI berbasis Deep Learning, seperti model Donut, untuk menghasilkan teks digital dengan tingkat akurasi yang tinggi.

C.1.4 Ekstraksi Data

Setelah proses OCR selesai, hasil ekstraksi berupa teks digital akan diolah lebih lanjut menggunakan algoritma ekstraksi teks untuk mengidentifikasi data penting seperti tanggal, nominal transaksi, nama barang, nomor invoice, atau data penting lainnya sesuai dengan jenis dokumen yang diproses.

C.1.5 Cek Otomatis dan Validasi Data

Setelah data diekstrak, sistem akan secara otomatis memvalidasi hasil ekstraksi dengan cara mencocokkannya terhadap pola-pola tertentu atau format standar yang telah ditentukan. Jika ditemukan ketidaksesuaian atau kesalahan pada data yang diekstrak, sistem akan menandai data tersebut untuk dilakukan pengecekan atau input manual lebih lanjut agar diperbaiki.

C.1.6 Approval dan Integrasi ke Sistem Manajemen Dokumen

Data yang telah lolos validasi akan ditampilkan dalam sistem approval untuk diverifikasi oleh pengguna yang berwenang. Setelah disetujui, data secara otomatis akan diintegrasikan ke dalam sistem manajemen dokumen internal seperti Excel atau SAP untuk keperluan pelaporan atau proses keuangan berikutnya.

C.2 Pembuatan Infrastruktur OCR

Pembuatan proyek OCR ini terbagi ke dalam beberapa tahap paralel, meliputi:

C.2.1 Pengembangan Frontend

Menggunakan framework Next.js untuk membuat antarmuka pengguna yang interaktif dan responsif. Halaman-halaman seperti login, halaman utama untuk upload dokumen, preview dokumen, halaman proses OCR, hingga tampilan untuk approval data dikembangkan secara bertahap dan modular.

C.2.2 Pengembangan Backend

Backend dikembangkan untuk menangani koneksi antara frontend, OCR, dan database. Ini mencakup API untuk CRUD dokumen, manajemen metadata dokumen, integrasi OCR, audit API, serta pengelolaan workflow approval.

C.2.3 Pengelolaan Database

Basis data dirancang dengan ERD yang terstruktur jelas namun dinamis, hal ini dilakukan agar dapat mencakup tabel-tabel dokumen yang bervariatif, hasil ekstraksi OCR, status approval, dan audit log aktivitas pengguna.

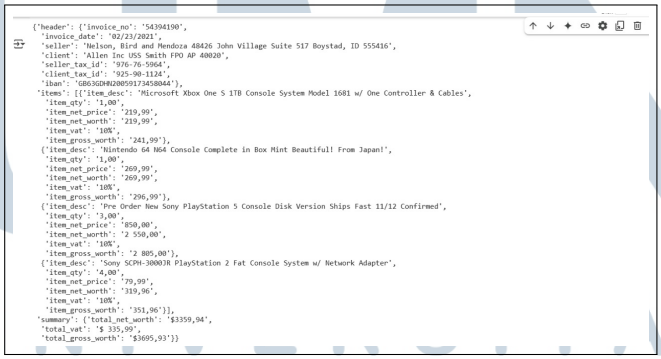
C.2.4 Pengembangan Model AI dan OCR

Pengembangan model OCR berbasis AI menggunakan teknologi Donut yang akan difine-tune dengan dataset yang telah dikumpulkan dan dilabel secara khusus. Proses ini memastikan tingkat akurasi hasil OCR yang tinggi serta performa yang konsisten di berbagai jenis dokumen finansial.

C.3 Pembuatan Backend OCR

Lapisan backend dalam proyek OCR Finance bertindak sebagai "ruang mesin" yang menerima permintaan (request) dari aplikasi antarmuka pengguna, memprosesnya, lalu mengirimkan balasan (response) dengan data atau status yang dibutuhkan. Untuk membangun layanan ini dipilih FastAPI, sebuah kerangka kerja (framework) Python modern yang sudah menggunakan gaya pemrograman *asinkron*. *Asinkron*—secara sederhana—berarti server dapat mengerjakan banyak permintaan secara bergantian tanpa menunggu satu tugas selesai sepenuhnya. Ibarat dapur restoran: alih-alih menunggu satu masakan matang sebelum mengerjakan pesanan berikutnya, koki dapat menyiapkan beberapa menu secara paralel; ketika satu panci perlu dididihkan, ia beralih memotong bahan lain. Pola non-blokir ini membuat aplikasi tetap responsif walau ada permintaan yang memerlukan waktu relatif lama, seperti menjalankan model OCR atau mengunggah file besar.

Seluruh data hasil ekstraksi teks, status dokumen, dan informasi pengguna disimpan di PostgreSQL, sebuah sistem basis data relasional yang terkenal stabil dan mendukung transaksi aman. Basis data dapat diinstal secara lokal di komputer server (on-premise) maupun dijalankan di layanan cloud terkelola—keduanya didukung penuh oleh desain backend. Agar koneksi ke PostgreSQL tidak dibuat-hapus berulang kali (boros sumber daya), backend membentuk *connection pool*: sekumpulan sambungan aktif yang dipakai bergiliran oleh setiap permintaan.

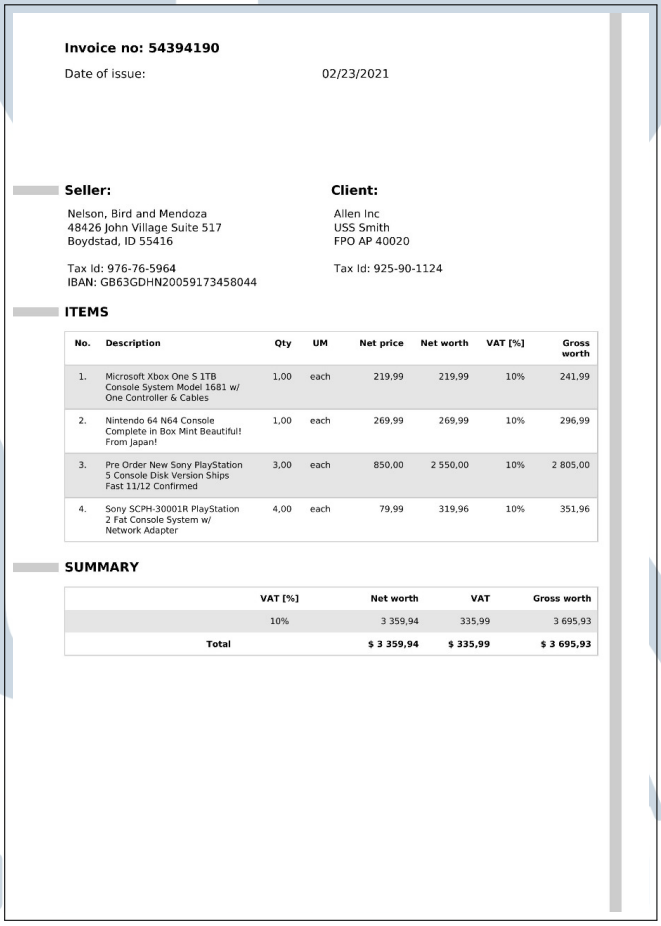


Gambar 3.29. Salah satu hasil data yang bagus dan juga akurat

Untuk keamanan, setiap pengguna yang berhasil masuk (*login*) memperoleh JSON Web Token (JWT). Token ini ibarat kartu identitas digital berisi informasi terenkripsi—siapa pengguna dan kapan token kedaluwarsa. Saat aplikasi klien memanggil API, token dikirim di bagian header sehingga server dapat segera memverifikasi identitas tanpa menyimpan sesi di memori; cara ini ringan dan cocok

untuk arsitektur terdistribusi.

Saat laporan ini disusun, modul *approval* belum diaktifkan penuh di lingkungan produksi. Perhatian dipusatkan pada penyempurnaan hasil OCR dan penyesuaian skema backend dengan model DONUT yang terus di-*fine-tune*. Setiap kali parameter model, nama field, atau struktur keluaran JSON berubah, tabel `documents` dan service pengekstrak harus ikut diperbarui agar kolom hasil tetap sinkron. Untuk meminimalkan kompleksitas, iterasi awal difokuskan pada *single template*—yakni format invoice standar—sehingga kualitas ekstraksi dapat divalidasi secara ketat sebelum membuka dukungan ke variasi layout lain. Setelah akurasi pada template pertama dianggap stabil, mekanisme approval otomatis dan multi-template akan diaktifkan kembali.



Invoice no: 54394190

Date of issue: 02/23/2021

Seller:

Nelson, Bird and Mendoza
48426 John Village Suite 517
Boydstad, ID 55416

Tax Id: 976-76-5964
IBAN: GB63GDHN20059173458044

Client:

Allen Inc
USS Smith
FPO AP 40020

Tax Id: 925-90-1124

ITEMS

No.	Description	Qty	UM	Net price	Net worth	VAT [%]	Gross worth
1.	Microsoft Xbox One S 1TB Console System Model 1681 w/ One Controller & Cables	1,00	each	219,99	219,99	10%	241,99
2.	Nintendo 64 N64 Console Complete in Box Mint Beautiful! From Japan!	1,00	each	269,99	269,99	10%	296,99
3.	Pre Order New Sony PlayStation 5 Console Disk Version Ships Fast 11/12 Confirmed	3,00	each	850,00	2 550,00	10%	2 805,00
4.	Sony SCPH-30001R PlayStation 2 Fat Console System w/ Network Adapter	4,00	each	79,99	319,96	10%	351,96

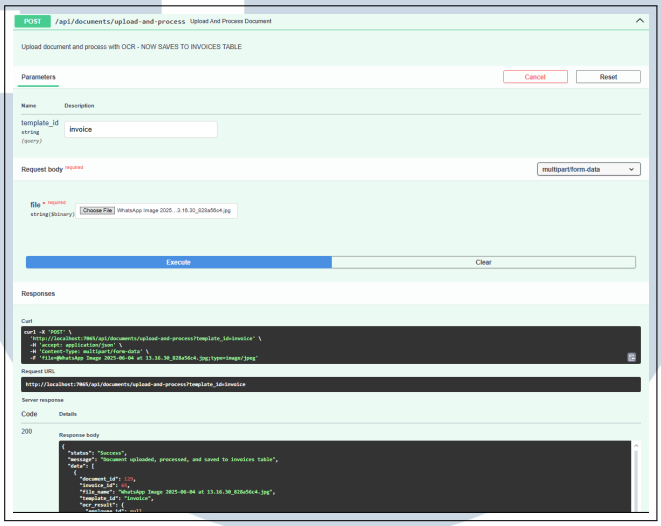
SUMMARY

VAT [%]	Net worth	VAT	Gross worth
10%	3 359,94	335,99	3 695,93
Total	**$ 3 359,94**	**$ 335,99**	**$ 3 695,93**

Gambar 3.30. Contoh Template yang digunakan

Alur pada umumnya, pengguna mengunggah gambar invoice melalui endpoint `/documents/upload`; backend menyimpan file, menjalankan model OCR untuk mengekstrak teks, lalu menaruh hasilnya dalam bentuk JSON

di tabel `documents`. Jika dokumen memerlukan verifikasi, sistem otomatis membuat entri "approval" yang kemudian dapat disetujui (approve) atau ditolak (reject) oleh petugas melalui dashboard. Semua proses—pengunggahan, ekstraksi OCR, penilaian approval, hingga penyimpanan—ditangani oleh beragam *router* (penentuan alamat endpoint) dan *service* (logika bisnis) yang saling terpisah agar mudah dirawat dan dikembangkan.



Gambar 3.31. Tampilan percobaan API upload-and-process melalui Swagger UI

C.3.1 Approvals Router (approvals.py)

File `approvals.py` mendefinisikan seluruh endpoint yang berkaitan dengan siklus persetujuan (approval) dokumen. Router ini dipasang dengan prefiks `/approvals` dan tag `"approvals"`. Semua fungsi menerima koneksi database dari `get_db()` (connection-pool `asyncpg`) dan untuk saat ini tidak memerlukan autentikasi—hal ini memudahkan pengujian awal tanpa token. Endpoint `POST /` membuat permintaan approval baru dengan menetapkan *system approver* (`user_id=1`) sebagai pemeriksa default. Endpoint `GET /` menampilkan seluruh approval, sedangkan `GET /pending` memfilter yang masih berstatus `pending`. Detail satu approval dapat diambil via `GET /{approval_id}`, sementara `PUT /{approval_id}` memperbarui status menjadi `approved` atau `rejected`. Terdapat pula `GET /pending-approval` yang menggabungkan tabel `documents` dan `approvals` untuk mencari file OCR berstatus `completed` namun belum disetujui, dan `GET /document/{doc_id}` untuk melihat seluruh histori approval milik dokumen tertentu. Setiap SELECT menggunakan klausa `OFFSET–LIMIT` agar paginasi efisien dan respons tetap ringan pada data berjumlah besar.

```
from fastapi import APIRouter, Depends, HTTPException, status
from typing import List
```

```python
import json
import asyncpg
from ...database.connection import get_db
from ...schemas.approval import ApprovalCreate, ApprovalResponse, ApprovalUpdate
from ...services.approval_service import approval_service

router = APIRouter(prefix="/approvals", tags=["approvals"])

@router.post("/", response_model=ApprovalResponse)
async def create_approval(
    approval_data: ApprovalCreate,
    db: asyncpg.Connection = Depends(get_db)
):
    """Create a new approval request (no auth required for testing)"""
    system_approver_id = 1
    approval = await approval_service.create_approval(
        db, system_approver_id, approval_data
    )
    return approval

@router.get("/", response_model=List[ApprovalResponse])
async def get_all_approvals(
    skip: int = 0,
    limit: int = 100,
    db: asyncpg.Connection = Depends(get_db)
):
    """Get all approvals (no auth required for testing)"""
    rows = await db.fetch(
        """
        SELECT * FROM approvals
        ORDER BY created_at DESC
        OFFSET $1 LIMIT $2
        """,
        skip, limit
    )
    return [ApprovalResponse(**dict(row)) for row in rows]

@router.get("/pending", response_model=List[ApprovalResponse])
async def get_pending_approvals(
    skip: int = 0,
    limit: int = 100,
```

```
    db: asyncpg.Connection = Depends(get_db)
):
    """Get all pending approvals (no auth required for testing)"""
    rows = await db.fetch(
        """
        SELECT * FROM approvals
        WHERE status = 'pending'
        ORDER BY created_at ASC
        OFFSET $1 LIMIT $2
        """,
        skip, limit
    )
    return [ApprovalResponse(**dict(row)) for row in rows]

@router.get("/{approval_id}", response_model=ApprovalResponse)
async def get_approval(
    approval_id: int,
    db: asyncpg.Connection = Depends(get_db)
):
    """Get specific approval (no auth required for testing)"""
    approval = await approval_service.get_approval(db, approval_id
    )
    if not approval:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Approval not found"
        )
    return approval

@router.put("/{approval_id}", response_model=ApprovalResponse)
async def update_approval(
    approval_id: int,
    update_data: ApprovalUpdate,
    db: asyncpg.Connection = Depends(get_db)
):
    """Update approval status (no auth required for testing)"""
    existing_approval = await approval_service.get_approval(db,
    approval_id)
    if not existing_approval:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Approval not found"
        )
```

```
    updated_approval = await approval_service.update_approval(
        db, approval_id, update_data
    )
    return updated_approval

@router.get("/pending-approval", response_model=List[dict])
async def get_documents_pending_approval(
    skip: int = 0,
    limit: int = 100,
    db: asyncpg.Connection = Depends(get_db)
):
    """Get documents that need approval"""
    rows = await db.fetch(
        """
        SELECT d.*, a.id AS approval_id, a.status AS approval_status,
               a.created_at AS approval_created_at
        FROM documents d
        LEFT JOIN approvals a ON d.id = a.document_id
        WHERE d.ocr_status = 'completed'
          AND (a.status = 'pending' OR a.status IS NULL)
        ORDER BY d.created_at DESC
        OFFSET $1 LIMIT $2
        """,
        skip, limit
    )
    result = []
    for row in rows:
        doc = dict(row)
        if doc.get("parsed_json") and isinstance(doc["parsed_json"], str):
            try:
                doc["parsed_json"] = json.loads(doc["parsed_json"])
            except json.JSONDecodeError:
                doc["parsed_json"] = {}
        result.append({
            "id": doc["id"],
            "file_name": doc["file_name"],
            "template_id": doc["template_id"],
            "parsed_json": doc["parsed_json"],
            "confidence_score": doc["confidence_score"],
            "created_at": doc["created_at"],
```

```
            "approval_id": doc["approval_id"],
            "approval_status": doc["approval_status"]
        })
    return result

@router.get("/document/{document_id}", response_model=List[
    ApprovalResponse])
async def get_document_approvals(
    document_id: int,
    db: asyncpg.Connection = Depends(get_db)
):
    """Get all approvals for a document (no auth required for
    testing)"""
    approvals = await approval_service.get_approvals_by_document(
    db, document_id)
    return approvals
```

Kode 3.1: Snippet Kode approvals.py

C.3.2 Authentication Router (auth.py)

File `auth.py` berisi rute untuk registrasi dan login pengguna. Router dipasang dengan prefiks `/auth` dan tag `"authentication"`. Dua fungsi utilitas, `hash_password()` dan `verify_password()`, memanfaatkan bcrypt untuk mengubah kata sandi menjadi hash aman dan memverifikasi kecocokannya saat login. Fungsi `create_access_token()` membuat JWT, data klaim diperluas dengan kunci `"exp"` yang menandai waktu kedaluwarsa token (default 24 jam) sebelum dienkode memakai `SECRET_KEY` dan algoritma `HS256`.

Endpoint `POST /auth/register` menerima `username`, `email`, `password`, dan `role`. Sistem terlebih dulu memeriksa keberadaan `username` atau `email` di tabel `users`. Jika tidak ada duplikasi, kata sandi di-hash, data disisipkan, dan baris lengkap—termasuk kolom `role` serta `is_active=true`—dikembalikan sebagai objek `UserResponse`.

Endpoint `POST /auth/login` memverifikasi kredensial: backend mengambil baris pengguna aktif berdasarkan `username` lalu menegaskan kecocokan hash melalui `verify_password()`. Apabila valid, server membuat JWT dengan klaim `{"sub": user_id}`, kemudian mengembalikan objek `Token` berisi `access_token`, `token_type="bearer"`, dan profil pengguna. Jika kredensial gagal diverifikasi server merespons HTTP 401 Unauthorized

```
from fastapi import APIRouter, Depends, HTTPException, status
```

```
from jose import jwt  # Change this line
import bcrypt
from datetime import datetime, timedelta
import asyncpg
from ...database.connection import get_db
from ...schemas.user import UserCreate, UserLogin, UserResponse, Token
from ...api.dependencies import SECRET_KEY, ALGORITHM

router = APIRouter(prefix="/auth", tags=["authentication"])

def hash_password(password: str) -> str:
    """Hash password with bcrypt"""
    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verify password"""
    return bcrypt.checkpw(plain_password.encode('utf-8'), hashed_password.encode('utf-8'))

def create_access_token(data: dict, expires_delta: timedelta = None):
    """Create JWT token"""
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(hours=24)

    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

@router.post("/register", response_model=UserResponse)
async def register(
    user_data: UserCreate,
    db: asyncpg.Connection = Depends(get_db)
):
    """Register a new user"""
    # Check if user already exists
```

```
    existing_user = await db.fetchrow(
        "SELECT id FROM users WHERE username = $1 OR email = $2",
        user_data.username, user_data.email
    )

    if existing_user:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Username or email already registered"
        )

    # Hash password and create user
    hashed_password = hash_password(user_data.password)

    query = """
        INSERT INTO users (username, email, password_hash, role)
        VALUES ($1, $2, $3, $4)
        RETURNING *
    """

    row = await db.fetchrow(
        query,
        user_data.username,
        user_data.email,
        hashed_password,
        user_data.role.value
    )

    return UserResponse(**dict(row))

@router.post("/login", response_model=Token)
async def login(
    user_credentials: UserLogin,
    db: asyncpg.Connection = Depends(get_db)
):
    """Login user and return JWT token"""
    print("Login attempt for: '{user_credentials.username}'")

    # Get user from database
    query = "SELECT * FROM users WHERE username = $1 AND is_active
    = true"
    row = await db.fetchrow(query, user_credentials.username)

```

```
    if not row:
        print(" User not found: '{user_credentials.username}'")
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password"
        )

    print(" User found: {row['username']}")
    print(" Stored hash: {row['password_hash'][:20]}...")
    print(" Input password: '{user_credentials.password}'")

    # Test password verification
    password_valid = verify_password(user_credentials.password,
row['password_hash'])
    print(" Password verification result: {password_valid}")

    if not password_valid:
        print(" Password verification failed")
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password"
        )

    print(" Login successful!")

    # Create access token
    access_token = create_access_token(data={"sub": row['id']})
    user = UserResponse(**dict(row))

    return Token(
        access_token=access_token,
        token_type="bearer",
        user=user
    )
```

Kode 3.2: Snippet Kode auth.py

C.3.3 Documents Router (documents.py)

File `documents.py` adalah inti manajemen dokumen pada backend OCR Finance. Router ini dipasang dengan prefiks `/documents` dan tag `"documents"`. Fungsinya mencakup enam kelompok utama: 1) pembuatan dokumen, 2) unggah file, 3) proses OCR, 4) pengambilan data (list, detail, pending approval, template),

5) penyimpanan hasil akhir ke tabel `invoices`, serta 6) operasi pemutakhiran dan penghapusan. Endpoint `POST /` membuat entri dokumen kosong berbasis skema `DocumentCreate` dan—untuk kemudahan uji coba—menggunakan `user_id=1`. Endpoint `GET /` menampilkan daftar dokumen terurut mundur; ia memakai `OFFSET–LIMIT` agar paginasi efisien.

Fungsi unggah inti terletak pada `POST /upload`. Endpoint ini memvalidasi tipe (PDF / JPEG / PNG) dan ukuran (maks. 10 MB), menyimpan file ke direktori `uploads/` dengan nama acak UUID, lalu membuat entri database berstatus OCR "completed" sambil menuliskan *mock result* (vendor, nomor faktur, total) ke kolom `parsed_json`. Setelah itu router otomatis menambah baris di tabel `approvals` dengan status `pending`. Bagi frontend verifikator, `GET /pending-approval` menggabungkan tabel `documents` dan `approvals` untuk menampilkan dokumen yang OCR-nya selesai tetapi belum disetujui. Detail penuh—termasuk histori approval dengan nama email approver—dapat diakses melalui `GET /{document_id}/full`.

Endpoint paling penting untuk integrasi Donut adalah `POST /{template_id}/process`. Ia menerima JSON hasil OCR (schema `OCRProcessRequest`), membuat entri dokumen berstatus "processing", lalu menjalankan `ocr_service.process_document_ocr()`. Jika berhasil, router memicu pembuatan approval otomatis. Versi kompaknya, `POST /upload-and-process`, langsung mengunggah file, memanggil Donut untuk inferensi, dan—bila template "invoice"—mengekstrak data terstruktur dan menyimpan ke tabel `invoices`. Hasil percobaan dapat dilihat pada gambar 3.31

Untuk memfasilitasi penyuntingan manual, `PUT /{document_id}/save-final-data` menyimpan revisi `InvoiceDataUpdate` ke tabel `invoices`, sedangkan `GET /{document_id}/structured-data` mengembalikan data terstruktur yang sudah disimpan (atau jatuh ke `parsed_json` sebagai cadangan). Rute tambahan `GET /templates` dan `GET /templates/{template_id}/documents` memberikan statistik penggunaan template dan daftar dokumen per template. setiap operasi menggunakan koneksi database melalui dependency `get_db()`; logika bisnis berat didelegasikan ke `document_service`, `ocr_service`, dan `approval_service`.

```python
@router.post("/upload", response_model=dict)  # This should match frontend
async def upload_document(
    file: UploadFile = File(...),
    template_id: str = "invoice",
    db: asyncpg.Connection = Depends(get_db)
```

```
):
    """Upload a document file for OCR processing"""

    print("Uploading file: {file.filename}")

    # Validate file type
    allowed_types = ['application/pdf', 'image/jpeg', 'image/png',
    'image/jpg']
    if file.content_type not in allowed_types:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"File type {file.content_type} not supported"
        )

    # Read file content
    content = await file.read()
    if len(content) > 10 * 1024 * 1024:  # 10MB limit
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="File too large. Maximum size is 10MB"
        )

    try:
        # Create uploads directory
        upload_dir = Path("uploads")
        upload_dir.mkdir(exist_ok=True)

        # Save file
        file_extension = Path(file.filename).suffix
        unique_filename = f"{uuid.uuid4()}{file_extension}"
        file_path = upload_dir / unique_filename

        with open(file_path, "wb") as buffer:
            buffer.write(content)

        # Create document in database
        system_user_id = 1
        query = """
            INSERT INTO documents (
                user_id, file_name, file_path, file_size,
   mime_type,
                template_id, ocr_status, created_at
            ) VALUES ($1, $2, $3, $4, $5, $6, $7,
```

```
    CURRENT_TIMESTAMP)
47              RETURNING id
48          """
49
50          document = await db.fetchrow(
51              query, system_user_id, file.filename, str(file_path),
52              len(content), file.content_type, template_id, '
    completed'
53          )
54
55          # Mock OCR result
56          mock_result = {
57              "vendor_name": "Sample Company Ltd",
58              "invoice_number": f"INV-{uuid.uuid4().hex[:8].upper()}
    ",
59              "total": f"{random.randint(100, 9999)}.{random.randint
    (10, 99)}"
60          }
61
62          # Update with OCR data
63          await db.execute(
64              "UPDATE documents SET parsed_json = $1,
    confidence_score = $2 WHERE id = $3",
65              json.dumps(mock_result), random.uniform(0.85, 0.98),
    document['id']
66          )
67
68          # Create approval
69          await db.execute(
70              "INSERT INTO approvals (document_id, status) VALUES (
    $1, 'pending')",
71              document['id']
72          )
73
74          return {
75              "success": True,
76              "document_id": document['id'],
77              "file_name": file.filename,
78              "status": "completed"
79          }
80
81      except Exception as e:
82          raise HTTPException(status_code=500, detail=f"Upload
```

```
    failed: {str(e)}")
```

Kode 3.3: Snippet Kode documents.py

C.3.4 Dependencies (`dependencies.py`)

File `dependencies.py` menyediakan lapisan keamanan dan helper yang disuntikkan (*dependency injection*) ke setiap endpoint FastAPI. Ia memuat objek `HTTPBearer` untuk mengekstrak header `Authorization: Bearer <token>` dan menggunakan `python-jose` guna mendekode JSON Web Token (JWT) dengan `SECRET_KEY` dan algoritma `HS256`. Fungsi `get_current_user()` mengambil token, memverifikasi tanda tangan, lalu mencari baris pengguna aktif di tabel `users`. Jika token kadaluwarsa atau pengguna tidak ditemukan, fungsi melempar `HTTP_401_UNAUTHORIZED`. Dua guard tambahan—`get_admin_user()` dan `get_approver_user()`—memeriksa kolom `role`; pertama hanya mengizinkan `UserRole.ADMIN`, sedangkan kedua menerima `ADMIN` maupun `APPROVER`. Berkat pendekatan ini, endpoint sensitif cukup menambahkan `Depends(get_admin_user)` atau `Depends(get_approver_user)` untuk membatasi akses tanpa menulis logika otorisasi berulang

```
import os
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer,
    HTTPAuthorizationCredentials
from jose import jwt  # Change this line
import asyncpg
from typing import Optional
from ..database.connection import get_db
from ..database.models import UserRole
from ..schemas.user import UserResponse

# Security - Load from environment
security = HTTPBearer()
SECRET_KEY = os.getenv("SECRET_KEY", "your-fallback-secret-key")
ALGORITHM = "HS256"

async def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(security),
    db: asyncpg.Connection = Depends(get_db)
) -> UserResponse:
    """Get current authenticated user"""
    try:
        token = credentials.credentials
```

```
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        user_id: int = payload.get("sub")
        if user_id is None:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Could not validate credentials"
            )
    except jwt.PyJWTError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Could not validate credentials"
        )

    # Get user from database
    row = await db.fetchrow(
        "SELECT * FROM users WHERE id = $1 AND is_active = true",
        user_id
    )

    if row is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="User not found or inactive"
        )

    return UserResponse(**dict(row))

async def get_admin_user(
    current_user: UserResponse = Depends(get_current_user)
) -> UserResponse:
    """Require admin privileges"""
    if current_user.role != UserRole.ADMIN:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Admin privileges required"
        )
    return current_user

async def get_approver_user(
    current_user: UserResponse = Depends(get_current_user)
) -> UserResponse:
    """Require approver or admin privileges"""
```

```
    if current_user.role not in [UserRole.ADMIN, UserRole.APPROVER
    ]:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail="Approver privileges required"
        )
    return current_user
```

Kode 3.4: Snippet Kode depenencies.py

C.3.5 Database Connection Helper (`connection.py`)

File `connection.py` merangkum logika pembuatan dan pengelolaan *connection-pool* PostgreSQL menggunakan pustaka `asyncpg`. Kelas `DatabaseConnection` mem-*encapsulate* sebuah atribut `pool` bertipe `Optional[asyncpg.Pool]` serta membaca `DATABASE_URL` dari variabel lingkungan. Metode `create_pool()` memanggil `asyncpg.create_pool()` dengan konfigurasi `min_size=10`, `max_size=20`, `command_timeout=120 s`, dan `timeout=30 s` untuk waktu tunggu akuisisi koneksi. Metode `close_pool()` menutup pool secara elegan ketika aplikasi berhenti. Sebuah *context-manager* asinkron, `get_connection()`, menyediakan koneksi siap pakai yang dilepas kembali ke pool setelah blok `async with` selesai. Objek `db = DatabaseConnection()` diekspos secara global, sedangkan fungsi `get_db()` dipakai FastAPI sebagai dependency sehingga setiap request memperoleh koneksi basis data tanpa membuat sambungan baru berulang kali .

```
import os
import asyncpg
import asyncio
from typing import Optional
from contextlib import asynccontextmanager

class DatabaseConnection:
    def __init__(self):
        self.pool: Optional[asyncpg.Pool] = None
        # Use environment variable; never commit real credentials
        self.database_url = os.getenv("DATABASE_URL", "postgresql
    ://user:password@localhost:5432/your_db")

    async def create_pool(self):
        """Create connection pool"""
        if not self.pool:
            self.pool = await asyncpg.create_pool(
                self.database_url,
```

```
                min_size=10,
                max_size=20,
                command_timeout=120,  # seconds
                timeout=30            # acquisition timeout
            )

    async def close_pool(self):
        """Close connection pool"""
        if self.pool:
            await self.pool.close()
            self.pool = None

    @asynccontextmanager
    async def get_connection(self):
        """Yield a database connection from the pool"""
        if not self.pool:
            await self.create_pool()
        async with self.pool.acquire() as connection:
            yield connection

# Global database instance
db = DatabaseConnection()

async def get_db():
    """FastAPI dependency providing a pooled connection"""
    async with db.get_connection() as connection:
        yield connection
```

Kode 3.5: Snippet Kode connection.py

C.3.6 Pydantic Data Models (`models.py`)

File `models.py` mendefinisikan seluruh skema data yang digunakan pada lapisan service dan router. Pertama, tiga `Enum`—`UserRole`, `OCRStatus`, dan `ApprovalStatus`—menetapkan nilai legal masing-masing kolom, misalnya `OCRStatus` hanya dapat berisi `pending`, `processing`, `completed`, atau `failed`. Selanjutnya, tiap entitas utama—`User`, `Document`, dan `Approval`—turun dari `BaseModel` Pydantic dengan `ConfigDict(from_attributes=True)`; opsi ini mempermudah konversi hasil query asyncpg (berupa dict) ke objek Pydantic. Untuk setiap entitas disediakan varian *Create* dan *Update*: `UserCreate` memerlukan `username`, `email`, dan `password`; `UserUpdate` memakai field opsional agar PATCH dapat dilakukan sebagian. Hal serupa berlaku pada `DocumentCreate/DocumentUpdate` dan `ApprovalCreate/ApprovalUpdate`.

Field waktu `created_at` serta `updated_at` bertipe `datetime` untuk audit.

```python
from pydantic import BaseModel, ConfigDict
from typing import Optional, Dict, Any
from datetime import datetime
from enum import Enum

class UserRole(str, Enum):
    ADMIN = "admin"
    USER = "user"
    APPROVER = "approver"

class OCRStatus(str, Enum):
    PENDING = "pending"
    PROCESSING = "processing"
    COMPLETED = "completed"
    FAILED = "failed"

class ApprovalStatus(str, Enum):
    PENDING = "pending"
    APPROVED = "approved"
    REJECTED = "rejected"

class User(BaseModel):
    model_config = ConfigDict(from_attributes=True)

    id: int
    username: str
    email: str
    is_active: bool
    role: UserRole
    created_at: datetime
    updated_at: datetime

class UserCreate(BaseModel):
    username: str
    email: str
    password: str
    role: UserRole = UserRole.USER

class UserUpdate(BaseModel):
    username: Optional[str] = None
    email: Optional[str] = None
    is_active: Optional[bool] = None
```

```
    role: Optional[UserRole] = None

class Document(BaseModel):
    model_config = ConfigDict(from_attributes=True)

    id: int
    user_id: int
    file_name: str
    file_path: Optional[str] = None
    file_size: Optional[int] = None
    mime_type: Optional[str] = None
    template_id: Optional[str] = None
    ocr_status: OCRStatus
    confidence_score: Optional[float] = None
    processing_time: Optional[int] = None
    created_at: datetime
    updated_at: datetime

class DocumentCreate(BaseModel):
    file_name: str
    file_path: Optional[str] = None
    file_size: Optional[int] = None
    mime_type: Optional[str] = None
    template_id: Optional[str] = None

class DocumentUpdate(BaseModel):
    file_name: Optional[str] = None
    ocr_status: Optional[OCRStatus] = None
    confidence_score: Optional[float] = None
    processing_time: Optional[int] = None

class Approval(BaseModel):
    model_config = ConfigDict(from_attributes=True)

    id: int
    document_id: int
    approver_id: Optional[int] = None
    status: ApprovalStatus
    comments: Optional[str] = None
    created_at: datetime
    updated_at: datetime

class ApprovalCreate(BaseModel):
```

```
86      document_id: int
87      status: ApprovalStatus = ApprovalStatus.PENDING
88      comments: Optional[str] = None
89
90  class ApprovalUpdate(BaseModel):
91      status: Optional[ApprovalStatus] = None
92      comments: Optional[str] = None
```

Kode 3.6: Snippet Kode models.py

C.3.7 Approval Schemas (`approval.py`)

File `approval.py` memuat skema Pydantic khusus entitas persetujuan (approval). Ia mengimpor `ApprovalStatus` dari modul `database.models` untuk menjamin keseragaman enum status di seluruh aplikasi. Kelas dasar `ApprovalBase` hanya memuat satu atribut kunci, `document_id`, yang menautkan baris approval ke dokumen terkait. Kelas `ApprovalCreate` menurunkan `ApprovalBase` dan menambahkan kolom `status`—secara default diisi `PENDING`—serta `comments` opsional untuk catatan pengajuan.

Kelas `ApprovalResponse` mewakili bentuk data yang dikirimkan kembali ke klien setelah operasi baca. Ia menambahkan kolom `id` (primary key), `approver_id` (opsional, pengesah dokumen), `created_at`, dan `updated_at`. Properti konfigurasi `Config.from_attributes = True` memungkinkan konversi instan dari hasil query asyncpg (dict) ke objek Pydantic tanpa casting manual. Terakhir, `ApprovalUpdate` menyediakan field opsional `status` dan `comments`—cukup untuk operasi `PATCH` sehingga klien dapat memperbarui sebagian data tanpa mengirim keseluruhan objek.

```
1   from pydantic import BaseModel
2   from typing import Optional
3   from datetime import datetime
4   from ..database.models import ApprovalStatus
5
6   class ApprovalBase(BaseModel):
7       document_id: int
8
9   class ApprovalCreate(ApprovalBase):
10      status: ApprovalStatus = ApprovalStatus.PENDING
11      comments: Optional[str] = None
12
13  class ApprovalResponse(ApprovalBase):
14      id: int
15      approver_id: Optional[int] = None
```

```
    status: ApprovalStatus
    comments: Optional[str] = None
    created_at: datetime
    updated_at: datetime

    class Config:
        from_attributes = True

class ApprovalUpdate(BaseModel):
    status: Optional[ApprovalStatus] = None
    comments: Optional[str] = None
```

Kode 3.7: Snippet Kode approval.py

C.3.8 Document and Invoice Schemas (`document.py`)

File `document.py` mendefinisikan skema Pydantic terlengkap di proyek, karena di sinilah seluruh hasil OCR—baik data mentah maupun data terstruktur—dipetakan menjadi objek Python yang tervalidasi. Struktur dimulai dari `InvoiceItem`, skema satu baris barang pada faktur, yang menyimpan kuantitas, harga satuan, PPN, dan nilai total. Ia menggunakan `json_encoders` untuk mengonversi `Decimal` menjadi string demi kompatibilitas JSON. Lapisan berikutnya, `DocumentBase`, hanya menyimpan `file_name`; kelas turunan `DocumentCreate` menambah `file_path`, `mime_type`, dan `template_id` agar endpoint upload dapat membuat entri awal.

Kelas `DocumentResponse` memuat hampir semua kolom relevan: status OCR (`OCRStatus`), teks mentah (`raw_ocr_text`), hasil parsing JSON, skor kepercayaan, serta seluruh isian faktur (nomor invoice, tanggal jatuh tempo, total pajak, dan sebagainya). Koleksi `items` berupa list `InvoiceItem`—disetel default empty list—memungkinkan frontend menampilkan tabel barang di faktur. Untuk operasi `PATCH`, `DocumentUpdate` mendeklarasikan semua field sebagai *optional*, sehingga pengguna dapat memperbarui sebagian nilai tanpa mengirim objek penuh.

Bagian kedua berfokus pada pembuatan dan respons faktur. `InvoiceLineItem` bersifat *dynamic schema*: properti `extra = "allow"` mengizinkan frontend mengirim field apa pun di luar yang dideklarasikan, sangat berguna jika model Donut menemukan kolom unik di template baru. `InvoiceCreate` memuat `line_items` sebagai list `InvoiceLineItem`, sedangkan `InvoiceResponse` menambahkan kolom `created_at` dan `updated_at` agar histori dapat ditampilkan. Terakhir, `InvoiceDataUpdate` mendukung pembaruan

dinamis; semua field bertipe serba `Optional` dan encoding spesial disiapkan agar nilai `Decimal` atau `date` diubah ke ISO-8601 string saat diserialisasi.

Keseluruhan desain—menggunakan konfigurasi `from_attributes = True`—memungkinkan konversi instan dari hasil query *asyncpg* (dict) ke objek Pydantic tanpa casting manual, sekaligus memberi fleksibilitas penuh kepada frontend untuk menambah kolom baru tanpa perlu perubahan kode backend.

```
from pydantic import BaseModel, Field
from typing import Optional, Dict, Any, List, Union, Generic,
    TypeVar
from datetime import datetime, date
from decimal import Decimal
from ..database.models import OCRStatus

T = TypeVar("T")

# ---------- Standard API envelopes ----------

class StandardResponse(BaseModel, Generic[T]):
    status: str = Field(..., description="success or error")
    message: str = Field(..., description="Response message")
    data: Optional[T] = Field(None, description="Response data")

class DocumentListResponse(BaseModel):
    documents: List["DocumentResponse"]
    total: Optional[int] = None
    page: Optional[int] = None
    limit: Optional[int] = None

class DocumentDetailResponse(BaseModel):
    document: "DocumentResponse"
    approvals: Optional[List[Dict[str, Any]]] = None
    related_data: Optional[Dict[str, Any]] = None

# ---------- Core document + invoice item ----------

class InvoiceItem(BaseModel):
    id: Optional[int] = None
    document_id: Optional[int] = None
    number: Optional[int] = None
    description: Optional[str] = None
    quantity: Optional[Decimal] = None
    unit_price: Optional[Decimal] = None
```

```
    extended_value: Optional[Decimal] = None
    vat_percent: Optional[str] = None
    vat_amount: Optional[Decimal] = None

    class Config:
        from_attributes = True
        json_encoders = {Decimal: str}

class DocumentBase(BaseModel):
    file_name: str

class DocumentCreate(DocumentBase):
    file_path: Optional[str] = None
    file_size: Optional[int] = None
    mime_type: Optional[str] = None
    template_id: Optional[str] = None

class DocumentResponse(DocumentBase):
    id: int
    user_id: Optional[int] = None
    file_path: Optional[str] = None
    file_size: Optional[int] = None
    mime_type: Optional[str] = None
    template_id: Optional[str] = None
    ocr_status: OCRStatus
    raw_ocr_text: Optional[str] = None
    parsed_json: Optional[Dict[str, Any]] = None
    confidence_score: Optional[float] = None
    processing_time: Optional[int] = None
    created_at: datetime
    updated_at: datetime

    # Invoice-level fields
    employee_id: Optional[int] = None
    invoice_number: Optional[str] = None
    invoice_date: Optional[date] = None
    due_date: Optional[date] = None
    customer_number: Optional[str] = None
    customer_gfb_id: Optional[str] = None
    credit_term: Optional[str] = None
    attention: Optional[str] = None
    po_number: Optional[str] = None
    quotation_number: Optional[str] = None
```

```
    customer_name: Optional[str] = None
    customer_address: Optional[str] = None
    customer_city: Optional[str] = None
    customer_npwp: Optional[str] = None
    total_excl_vat: Optional[Decimal] = None
    total_tax_based_value: Optional[Decimal] = None
    total_vat_amount: Optional[Decimal] = None
    total_incl_vat: Optional[Decimal] = None
    items: Optional[List[InvoiceItem]] = []

    class Config:
        from_attributes = True

class DocumentUpdate(BaseModel):
    file_name: Optional[str] = None
    ocr_status: Optional[OCRStatus] = None
    raw_ocr_text: Optional[str] = None
    parsed_json: Optional[Dict[str, Any]] = None
    confidence_score: Optional[float] = None
    processing_time: Optional[int] = None
    template_id: Optional[st]()_
```

Kode 3.8: Snippet Kode document.py

C.3.9 User Schemas (`user.py`)

File `user.py` merangkum skema Pydantic untuk seluruh operasi terkait pengguna. Kelas dasar `UserBase` menyimpan `username` dan `email`; tipe `EmailStr` memastikan alamat e-mail tervalidasi formatnya. `UserCreate` mewarisi `UserBase` dan menambah `password` serta kolom `role` yang secara default diisi `UserRole.USER`. Untuk proses login, `UserLogin` hanya membutuhkan `username` dan `password`. Respons standar dari endpoint pengguna diwakili `UserResponse`: field `id`, `is_active`, `role`, serta cap waktu `created_at` dan `updated_at`; konfigurasi `from_attributes=True` memudahkan konversi hasil query basis data menjadi objek Pydantic. Skema `UserUpdate` menjadikan semua properti bersifat opsional guna mendukung operasi `PATCH`. Terakhir, `Token` menyatukan `access_token`, `token_type="bearer"`, dan data `user` dalam satu respons saat login berhasil.

```
from pydantic import BaseModel, EmailStr
from typing import Optional
from datetime import datetime
from ..database.models import UserRole

```

```python
class UserBase(BaseModel):
    username: str
    email: EmailStr

class UserCreate(UserBase):
    password: str
    role: UserRole = UserRole.USER

class UserLogin(BaseModel):
    username: str
    password: str

class UserResponse(UserBase):
    id: int
    is_active: bool
    role: UserRole
    created_at: datetime
    updated_at: datetime

    class Config:
        from_attributes = True

class UserUpdate(BaseModel):
    username: Optional[str] = None
    email: Optional[EmailStr] = None
    is_active: Optional[bool] = None
    role: Optional[UserRole] = None

class Token(BaseModel):
    access_token: str
    token_type: str
    user: UserResponse
```

Kode 3.9: Snippet Kode user.py

C.3.10 Approval Service (`approval_service.py`)

Berkas `approval_service.py` mengumpulkan seluruh logika bisnis terkait objek persetujuan ke dalam satu kelas `ApprovalService`. Setiap metode menerima `asyncpg.Connection` agar dapat dijalankan secara asinkron di dalam `router`. Fungsi `create_approval()` menyisipkan baris baru ke tabel `approvals`—menggunakan parameter `document_id`, `approver_id`, status awal, dan komentar—lalu mengembalikan objek `ApprovalResponse`.

Metode `get_approval()` mengambil satu baris berdasarkan `id`, sedangkan `get_approvals_by_document()` menarik seluruh histori persetujuan milik satu dokumen dengan urutan terbaru di atas. Metode `get_pending_approvals()` memfilter baris berstatus `PENDING` yang ditugaskan kepada seorang approver tertentu, disertai dukungan paginasi `skip–limit`. Terakhir, `update_approval()` menerima objek `ApprovalUpdate`; ia secara dinamis membangun klausa `SET` hanya dari field yang dikirim klien, menambahkan stempel waktu `updated_at = CURRENT_TIMESTAMP`, dan mengembalikan baris terbarui. Objek `approval_service` dibuat di bagian akhir file sehingga dapat di-*import* langsung oleh `approvals.py`.

```
from typing import List, Optional
import asyncpg
from ..schemas.approval import ApprovalCreate, ApprovalUpdate,
    ApprovalResponse
from ..database.models import ApprovalStatus

class ApprovalService:
    def __init__(self):
        pass

    async def create_approval(
        self,
        connection: asyncpg.Connection,
        approver_id: int,
        approval_data: ApprovalCreate
    ) -> ApprovalResponse:
        """Create a new approval request"""
        row = await connection.fetchrow(
            """
            INSERT INTO approvals (document_id, approver_id,
    status, comments)
            VALUES ($1, $2, $3, $4)
            RETURNING *
            """,
            approval_data.document_id,
            approver_id,
            approval_data.status.value,
            approval_data.comments
        )
        return ApprovalResponse(**dict(row))

    async def get_approval(
        self,
```

```
        connection: asyncpg.Connection,
        approval_id: int
    ) -> Optional[ApprovalResponse]:
        """Get approval by ID"""
        row = await connection.fetchrow(
            "SELECT * FROM approvals WHERE id = $1",
            approval_id
        )
        return ApprovalResponse(**dict(row)) if row else None

    async def get_approvals_by_document(
        self,
        connection: asyncpg.Connection,
        document_id: int
    ) -> List[ApprovalResponse]:
        """Get approvals for a document"""
        rows = await connection.fetch(
            """
            SELECT * FROM approvals
            WHERE document_id = $1
            ORDER BY created_at DESC
            """,
            document_id
        )
        return [ApprovalResponse(**dict(row)) for row in rows]

    async def get_pending_approvals(
        self,
        connection: asyncpg.Connection,
        approver_id: int,
        skip: int = 0,
        limit: int = 100
    ) -> List[ApprovalResponse]:
        """Get pending approvals for an approver"""
        rows = await connection.fetch(
            """
            SELECT a.* FROM approvals a
            JOIN documents d ON a.document_id = d.id
            WHERE a.approver_id = $1
              AND a.status = $2
            ORDER BY a.created_at ASC
            OFFSET $3 LIMIT $4
            """,
```

```
            approver_id,
            ApprovalStatus.PENDING.value,
            skip,
            limit
        )
        return [ApprovalResponse(**dict(row)) for row in rows]

    async def update_approval(
        self,
        connection: asyncpg.Connection,
        approval_id: int,
        update_data: ApprovalUpdate
    ) -> Optional[ApprovalResponse]:
        """Update approval status"""
        fields = []
        values = []
        param_count = 1

        for field, value in update_data.model_dump(exclude_unset=
True).items():
            if value is not None:
                actual_val = value.value if field == "status" else
 value
                fields.append(f"{field} = ${param_count}")
                values.append(actual_val)
                param_count += 1

        if not fields:
            return await self.get_approval(connection, approval_id
)

        query = f"""
            UPDATE approvals
            SET {', '.join(fields)}, updated_at =
CURRENT_TIMESTAMP
            WHERE id = ${param_count}
            RETURNING *
        """
        values.append(approval_id)

        row = await connection.fetchrow(query, *values)
        return ApprovalResponse(**dict(row)) if row else None

```

```
114 approval_service = ApprovalService()
```

Kode 3.10: Snippet Kode approval_service

C.3.11 Document Service (`document_service.py`)

Berkas `document_service.py` membungkus logika bisnis dokumen ke dalam kelas `DocumentService`. Metode `create_document()` menyisipkan baris baru ke tabel `documents`—dengan status awal `PENDING`—dan langsung mengembalikan objek `DocumentResponse`. Metode `get_document()` dan `get_documents_by_user()` membaca satu dokumen atau daftar dokumen (dengan paginasi) dari basis data. Metode `update_document_with_ocr_result()` menerima hasil OCR dan menambahkan kolom `ocr_data`, `processing_time`, dan `ocr_status` jika kolom tersebut belum ada—fungsi `ensure_column_exists()` menyusun perintah `ALTER TABLE` secara dinamis.

Untuk penghapusan, `delete_document()` membersihkan catatan di `invoice_line_items` dan `invoices`, lalu menghapus baris dokumen utama. Bagian "`dynamic schema`" menetapkan dua prosedur: `ensure_invoice_columns()` dan `ensure_invoice_line_items_columns()`, yang memindai `information_schema.columns` dan menambahkan kolom baru ke tabel `invoices` maupun `invoice_line_items` jika field dari hasil OCR JSON belum tersedia.

Metode `create_invoice()` menggunakan pemeriksaan kolom untuk menyusun perintah `INSERT` dinamis; kolom dan placeholder dibentuk dari pasangan kunci–nilai non-`null`, lalu baris invoice utama serta baris `line_items` disimpan. `update_invoice()` bekerja dengan cara yang serupa namun menyusun perintah `UPDATE` dinamis dan mengganti semua `line_items` lama. Untuk menyimpan hasil OCR, `process_ocr_result()` menilai struktur dokumen (stub `_analyze_document_structure`), menghitung skor keyakinan, memperbarui status menjadi `COMPLETED`, dan menulis `parsed_json`. Metode utilitas `_get_sql_type_for_value()` memetakan nilai Python ke tipe SQL konservatif; fungsi `_convert_value_for_column_type()` mengonversi tanggal, `dict`, `list`, dan angka agar cocok dengan tipe kolom di PostgreSQL.

Seluruh metode asinkron menerima `asyncpg.Connection` dan mengembalikan skema Pydantic (`DocumentResponse` atau `InvoiceResponse`) setelah parsing ulang menggunakan `_parse_db_row()`. Objek tunggal `document_service` di bagian akhir file disiapkan untuk diimpor langsung oleh router.

```python
class DocumentService:
    def __init__(self):
        pass

    # === BASIC CRUD METHODS ===

    async def create_document(
        self,
        connection: asyncpg.Connection,
        user_id: Optional[int],
        document_data: DocumentCreate
    ) -> DocumentResponse:
        """Create a new document"""
        row = await connection.fetchrow(
            """
            INSERT INTO documents (
                user_id, file_name, file_path, file_size, mime_type,
                template_id, ocr_status, created_at, updated_at
            ) VALUES ($1, $2, $3, $4, $5, $6, $7,
                      CURRENT_TIMESTAMP, CURRENT_TIMESTAMP)
            RETURNING *
            """,
            user_id,
            document_data.file_name,
            document_data.file_path,
            document_data.file_size,
            document_data.mime_type,
            document_data.template_id,
            OCRStatus.PENDING,
        )
        return DocumentResponse(**self._parse_db_row(row))

    async def get_document(
        self,
        connection: asyncpg.Connection,
        document_id: int
    ) -> Optional[DocumentResponse]:
        """Fetch a document by ID"""
        try:
            row = await connection.fetchrow(
                "SELECT * FROM documents WHERE id = $1",
                document_id
```

```
            )
            if not row:
                return None
            return DocumentResponse(**self._parse_db_row(row))
        except Exception as e:
            print(f"Error in get_document: {e}")
            return None

    # === OCR RESULT UPDATES ===

    async def update_document_with_ocr_result(
        self,
        connection: asyncpg.Connection,
        document_id: int,
        ocr_result: Dict[str, Any],
        processing_time: Optional[int] = None
    ) -> bool:
        """
        Update a document with OCR results.
        Dynamically ensures columns exist for `ocr_data`,
        `processing_time`, and `ocr_status`.
        """
        try:
            # Prepare JSON for storage
            ocr_data_json = json.dumps(ocr_result) if ocr_result
    else None

            # Ensure required columns are present
            await self.ensure_column_exists(connection, "documents
    ", "ocr_data", "JSONB")
            await self.ensure_column_exists(connection, "documents
    ", "processing_time", "INTEGER")
            await self.ensure_column_exists(connection, "documents
    ", "ocr_status", "VARCHAR(50)")

            row = await connection.fetchrow(
                """
                UPDATE documents
                SET ocr_data = $1,
                    processing_time = $2,
                    ocr_status = 'completed',
                    updated_at = CURRENT_TIMESTAMP
                WHERE id = $3
```

```
                RETURNING *
                """,
                ocr_data_json,
                processing_time or 0,
                document_id
            )
            return bool(row)
        except Exception as e:
            print(f"Error updating document {document_id}: {e}")
            return False
```

Kode 3.11: Snippet Kode document_service

C.3.12 OCR Service (`ocr_service.py`)

Berkas `ocr_service.py` memusatkan seluruh proses inferensi model Donut ke dalam kelas `OCRService`. Konstruktor kelas secara dinamis mencari direktori hasil kloning model Donut, menambahkannya ke `sys.path`, dan memuat model `fauzansalim/GITInvoice2` menggunakan `DonutModel.from_pretrained()`. Pemilihan perangkat (`cuda` atau `cpu`) dilakukan otomatis; jika GPU tersedia, bobot model dipindahkan ke CUDA dan dikonversi ke presisi `half`. Metode `run_donut_ocr()` digunakan untuk menjalankan inferensi terhadap satu gambar: metode ini menerima objek `PIL.Image`, memanggil `self.model.inference()` dengan prompt `<s_invoices-git2>`, mengukur durasi proses, lalu mengembalikan kamus Python berisi hasil terstruktur maupun `raw_text`.

Sebelum hasil dikirim ke antarmuka pengguna, fungsi `normalize_ocr_result()` digunakan untuk memecah keluaran `structured_data` dari Donut menjadi tiga bagian utama: `invoice_data`, `items`, dan `totals`, serta menghitung nilai `net_total` jika informasi memungkinkan. Untuk pemrosesan file, metode `get_file_info()` akan menebak `mime_type` dan ukuran file; sedangkan `_process_image_async()` menjalankan pembacaan gambar di thread pool, mengeksekusi model Donut, dan mengembalikan hasilnya ke loop asinkron utama.

Metode kunci `process_document_async()` menerima parameter `file_path` dan `document_id`, memilih jalur proses berdasarkan jenis file (saat ini hanya gambar), memanggil Donut, lalu memperbarui kolom `parsed_json` dan `processing_time` pada tabel `documents` menggunakan `document_service.update_document_with_ocr_result()`. Jalur terpadu `process_image_file()` bertanggung jawab untuk membuat entri dokumen baru, menjalankan proses OCR, dan membungkus hasil akhir ke dalam struktur JSON

yang dapat langsung dikonsumsi oleh frontend. Di akhir berkas, objek tunggal `ocr_service` diekspos agar modul router atau service lain dapat menggunakan fungsi-fungsi ini dengan cara `from ... import ocr_service`.

```python
import torch
from PIL import Image
import json
from typing import Dict, Any
import time
import os
import sys
from pathlib import Path

# Dynamically locate the cloned Donut source and add it to PYTHONPATH
current_dir = Path(__file__).resolve().parent
backend_root = current_dir.parent.parent                # <project-root>/Backend_API
candidate_paths = [
    backend_root / "donut",
    backend_root.parent / "donut",
    current_dir.parent.parent.parent / "donut",
    Path("/content/donut"),                             # Colab-style fallback
]

donut_path: Path | None = next(
    (p for p in candidate_paths if (p / "donut" / "__init__.py").exists()),
    None,
)
if donut_path and str(donut_path) not in sys.path:
    sys.path.insert(0, str(donut_path))
    print(f"Added donut path: {donut_path}")

from donut import DonutModel


class OCRService:
    """
    Wrapper around the Donut model that handles device placement, precision,
    and a simplified inference routine.
    """
```

```
36
37      def __init__(self) -> None:
38          try:
39              self.device = torch.device("cuda" if torch.cuda.
    is_available() else "cpu")
40              model_name = "fauzansalim/ALLInvoices"
41
42              print(f"Loading Donut model '{model_name}' on {self.
    device}")
43              self.model = DonutModel.from_pretrained(model_name)
44
45              if self.device.type == "cuda":
46                  self.model.half().to(self.device)
47                  print("Model moved to CUDA (float16)")
48              else:
49                  self.model.float()
50                  print("Model running on CPU (float32)")
51
52              self.model.eval()
53              print("Donut model loaded successfully")
54          except Exception as exc:
55              print("Model loading failed:", exc)
56              raise
57
58      @torch.inference_mode()
59      def run_donut_ocr(self, image: Image.Image) -> Dict[str, Any]:
60          """
61          Perform OCR on a single PIL image and return either
    structured JSON
62          or raw text, together with processing time in milliseconds
    .
63          """
64          task_prompt = "<s_dataset_all>"
65          start = time.perf_counter()
66
67          try:
68              output = self.model.inference(image=image, prompt=
    task_prompt)
69              duration_ms = int((time.perf_counter() - start) * 1
    _000)
70
71              if isinstance(output, dict) and output.get("
    predictions"):
```

```
                prediction = output["predictions"][0]
                return {
                    "result": {"structured_data": prediction},
                    "processing_time": duration_ms,
                }

            return {
                "result": {"raw_text": str(output)},
                "processing_time": duration_ms,
            }
        except Exception as exc:
            duration_ms = int((time.perf_counter() - start) * 1
    _000)
            return {
                "result": {"raw_text": f"Error: {exc}"},
                "processing_time": duration_ms,
            }


ocr_service = OCRService()
```

Kode 3.12: Snippet Kode ocr_service

C.3.13 Utility Functions (`helpers.py`)

File `helpers.py` menyajikan dua fungsi utilitas yang sering dipanggil oleh service layer. Fungsi pertama, `flatten()`, menerima `dict` bertingkat dan mengubahnya menjadi kamus datar satu level. Algoritme rekursifnya mengecek setiap nilai: bila bertipe `dict`, fungsi memanggil dirinya sendiri sambil menambahkan `parent_key`; bila bertipe list, fungsi mengiterasi elemen, memberi indeks pada kunci baru, dan melanjutkan proses yang sama. Separator default adalah garis bawah (_) sehingga struktur seperti `{"header":{"date":"2025-06-30"}}` menjadi `{"header_date":"2025-06-30"}`.

Fungsi kedua, `template_id()`, menghasilkan identitas template berbasis struktur kunci JSON hasil OCR. Ia menerima `set` berisi seluruh kunci ter-*flatten*, mengurutkannya untuk konsistensi, lalu membuat string gabungan dipisah pipa (`|`). Nilai string ini di-*hash* menggunakan MD5 dan dipotong delapan karakter pertama guna memperoleh ID singkat. Fungsi juga menebak kategori template: bila ada kata `item` dan `invoice`, ia mengembalikan pola `invoice_<hash>`; bila kata `receipt` terdeteksi, dikembalikan `receipt_<hash>`; jika tidak ada pola spesifik, keluaran

default adalah `generic_<hash>`. Dengan cara ini, backend dapat mengelompokkan dokumen berbeda ke dalam template unik tanpa konfigurasi manual.

```
from typing import Dict, Any, Set
import hashlib

def flatten(d: Dict[str, Any], parent_key: str = '', sep: str = '_') -> Dict[str, Any]:
    """
    Flatten a nested dictionary into a single level dictionary.
    """
    items = []
    for k, v in d.items():
        new_key = f"{parent_key}{sep}{k}" if parent_key else k
        if isinstance(v, dict):
            items.extend(flatten(v, new_key, sep=sep).items())
        elif isinstance(v, list):
            for i, item in enumerate(v):
                if isinstance(item, dict):
                    items.extend(flatten(item, f"{new_key}{sep}{i}", sep=sep).items())
                else:
                    items.append((f"{new_key}{sep}{i}", item))
        else:
            items.append((new_key, v))
    return dict(items)

def template_id(keys: Set[str]) -> str:
    """
    Generate a template ID based on the flattened key structure.
    """
    sorted_keys = sorted(keys)
    key_string = '|'.join(sorted_keys)
    template_hash = hashlib.md5(key_string.encode()).hexdigest()[:8]

    if any('item' in key.lower() for key in keys):
        if any(term in key.lower() for term in ('invoice', 'bill')):
            return f"invoice_{template_hash}"
        if any('receipt' in key.lower() for key in keys):
            return f"receipt_{template_hash}"
        return f"itemized_{template_hash}"
    if any('header' in key.lower() for key in keys):
```

```
        return f"document_{template_hash}"
    return f"generic_{template_hash}"
```

Kode 3.13: Snippet Kode helpers.py

C.3.14 Application Entrypoint (`main.py`)

File `main.py` bertindak sebagai titik mula (entry-point) aplikasi FastAPI. Pertama, ia memuat variabel lingkungan melalui `python-dotenv` agar `DATABASE_URL`, `SECRET_KEY`, dan konfigurasi lainnya tersedia bagi modul lain. Fungsi `lifespan()` didekorasi `@asynccontextmanager` untuk menangani siklus hidup server: saat startup, ia memanggil `db.create_pool()` guna membuka connection-pool PostgreSQL; saat shutdown, ia memastikan `db.close_pool()` terpanggil agar sambungan tertutup rapi. Objek `app = FastAPI(..., lifespan=lifespan)` dibuat dengan judul dan deskripsi singkat, menandakan bahwa mode "No Auth" sedang aktif (router otentikasi sengaja tidak diikutsertakan).

Middleware CORS ditambahkan dengan kebijakan permisif—semua origin, metode, dan header diizinkan—untuk memudahkan pengujian dari berbagai domain frontend. Selanjutnya, router `documents` dan `approvals` di-*include* pada prefiks `/api`. Dua endpoint diagnostik tersedia: `GET /` mengembalikan pesan status server, sedangkan `GET /health` memulangkan JSON `{"status":"healthy"}` yang dapat dipakai load-balancer untuk health-check.

Dalam blok `if __name__ == "__main__":`, file menjalankan server Uvicorn pada `0.0.0.0:8000` dengan `timeout_keep_alive=300`, `proxy_headers=True`, dan menonaktifkan header bawaan server; konfigurasi ini disiapkan agar aplikasi dapat dipasang langsung melalui Docker atau layanan Platform-as-a-Service.

```
import os
from dotenv import load_dotenv
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
from .database.connection import db
from .api.routes import documents, approvals

# Load environment variables
load_dotenv()

@asynccontextmanager
```

```
async def lifespan(app: FastAPI):
    # Startup
    await db.create_pool()
    print("Database connection pool created")
    yield
    # Shutdown
    await db.close_pool()
    print("Database connection pool closed")

# Create FastAPI app
app = FastAPI(
    title="OCR Backend API",
    description="Backend API for OCR document processing system (No Auth Mode)",
    version="1.0.0",
    lifespan=lifespan
)

# CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Routers
app.include_router(documents.router, prefix="/api")
app.include_router(approvals.router, prefix="/api")

@app.get("/")
async def root():
    return {"message": "OCR Backend API is running (No Auth Mode)"}

@app.get("/health")
async def health_check():
    return {"status": "healthy"}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(
```

```
        app,
        host="0.0.0.0",
        port=8000,
        timeout_keep_alive=300,
        proxy_headers=True,
        forwarded_allow_ips="*",
        server_header=False
    )
```

Kode 3.14: Snippet Kode main.py

D Pengujian Backend

Pengujian difokuskan semata–mata pada lapisan **backend** untuk memastikan layanan OCR Finance tetap andal, aman, dan mudah diganti meskipun model OCR-nya berganti (mis. dari `DONUT` ke `TrOCR`, `LayoutLM`, dan seterusnya).

D.1 Functional API Testing

Tujuan utama pengujian fungsional adalah memverifikasi bahwa setiap endpoint berperilaku sesuai kontrak REST dan tetap valid ketika model OCR diganti. Pengujian meliputi:

1. **Upload Dokumen** (`POST /documents/upload`) — memastikan file tersimpan, metadata tercatat, dan antrian inferensi dibuat.
2. **Inferensi OCR** (`POST /documents/{id}/process`) — memanggil abstraksi `OCRServiceInterface`. Dengan pola *dependency injection*, unit test mengganti implementasi asli dengan *stub* model untuk memverifikasi format keluaran tanpa tergantung model tertentu.
3. **Ambil Hasil Dokumen** (`GET /documents/{id}`) — memeriksa bahwa JSON hasil ekstraksi, status inferensi, dan stempel waktu valid.
4. **CRUD Approval** (`POST` / `PATCH` / `GET /approvals`) — mengecek pembuatan entri approval otomatis, serta transisi status *pending*, *approved*, *rejected*.

5. **Autentikasi & Otorisasi** (`POST /auth/login`, Middleware JWT) — unit test memverifikasi pembuatan token, validasi tanda tangan, dan penolakan permintaan dengan token kedaluwarsa.

6. **Koneksi Database** — integration test memastikan *connection pool* tidak bocor, transaksi `ROLLBACK` pada kegagalan, serta migrasi skema (`alembic`) berjalan mulus setelah perubahan kolom hasil model.

Temuan pengujian ini mengindikasikan bahwa, di luar keberhasilan setiap endpoint memenuhi kontrak REST, masih diperlukan penyempurnaan pada prosedur migrasi skema dan mekanisme monitoring—agar pergantian model OCR (contoh: DONUT $\rightarrow$ TrOCR $\rightarrow$ LayoutLM, dan seterusnya) dapat berlangsung mulus tanpa mengorbankan konsistensi data maupun stabilitas layanan.

Perlu dicatat pula bahwa seluruh pengujian dijalankan pada satu server internal berbasis CPU (tanpa GPU). Evaluasi tambahan di klaster produksi berkapasitas lebih besar atau pada konfigurasi micro-service terdistribusi masih diperlukan untuk memvalidasi kinerja dan daya tahan sistem pada skala beban yang lebih tinggi.

D.2 Hasil Singkat

Berdasarkan pengujian di server internal, seluruh alur—mulai dari unggah berkas, inferensi OCR, hingga penyimpanan hasil ke PostgreSQL—dapat diselesaikan rata-rata ±1 menit per dokumen. Ketika beban dinaikkan menjadi lima dokumen yang diproses tidak terlalu terpengaruhi dengan proses scan lainnya, performa tetap stabil: waktu rata-rata per dokumen tidak bergeser signifikan dan akurasi ekstraksi text beserta kolom tetap terjaga.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala yang Ditemui

Selama masa pelaksanaan magang dan pengembangan berbagai proyek, terdapat sejumlah tantangan teknis maupun non-teknis yang dihadapi. Beberapa di antaranya adalah sebagai berikut:

1. Dalam proyek OCR Finance, tim menghadapi tantangan dalam pemilihan dan penggunaan model. Karena kebutuhan akan akurasi tinggi dan format

keluaran yang spesifik (seperti JSON), proses pengembangan melibatkan perpindahan dari satu model ke model lainnya. Setiap perpindahan model memerlukan perubahan signifikan pada struktur kode, format output, serta strategi training. Hal ini menyebabkan waktu eksplorasi dan pengujian menjadi cukup panjang karena setiap model memiliki arsitektur dan perilaku yang berbeda.

2. Dalam pengerjaan proyek Tukerin, waktu pengembangan yang tersedia sangat terbatas sehingga banyak fitur harus dikejar dalam jangka waktu singkat. Hal ini menimbulkan tekanan dalam penyelesaian dan integrasi antar-komponen.

3. Keterbatasan waktu juga menyebabkan struktur *source code* yang dikembangkan menjadi kurang rapi. Banyak fungsi dan variabel dibuat secara cepat tanpa standar penamaan yang konsisten, dan belum terorganisasi secara modular, sehingga sulit dibaca kembali maupun dikembangkan lebih lanjut oleh anggota tim lain.

3.4.2 Solusi yang Diterapkan

Untuk mengatasi tantangan-tantangan di atas, beberapa strategi dan pendekatan diterapkan secara sistematis:

1. Dalam menghadapi tantangan migrasi model OCR, tim bekerja secara kolaboratif dalam skema tandem. Salah satu anggota fokus melakukan eksplorasi dan eksperimen awal terhadap model baru—dengan membuat *preview* sederhana—sementara anggota lainnya tetap menjaga kestabilan sistem yang ada. Ketika model baru terbukti lebih efektif, tim mulai memigrasikan pipeline lama ke struktur baru yang disesuaikan dengan model tersebut, termasuk melakukan penyesuaian dalam format input/output dan penataan ulang *service layer*.

2. Untuk mempercepat pengembangan prototipe aplikasi Tukerin, beberapa fitur *frontend* seperti *chat* yang dimana trigger chat bubble yang ada sengaja dibuat secara *hardcoded* terlebih dahulu tanpa tergantung pada API *backend*. Strategi ini memungkinkan pengguna atau stakeholder untuk mendapatkan gambaran alur sistem meskipun koneksi backend belum sepenuhnya stabil.

3. Setelah fitur utama berhasil diselesaikan dan terdapat jeda waktu sebelum rilis akhir, dilakukan sesi *code cleaning* untuk merapikan struktur proyek. Fungsi-fungsi dipisah ke dalam modul tersendiri, penamaan variabel diperjelas, dan komentar ditambahkan agar dokumentasi teknis menjadi lebih mudah dipahami dan dipelihara di masa depan.