BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pelaksanaan kerja magang dilakukan sebagai posisi *Technical Consultant* dalam divisi tim *Operation*. Untuk saat ini, divisi tim *Operation* beranggotakan 4 orang, Bapak Syahrial Danu sebagai ketua tim teknis, Bapak Rifqi Arief Wicaksana sebagai rekan kerja, dan Bapak Robiul Musthofa sebagai koordinator penugasan dan *Project Manager*. Pekerjaan-pekerjaan yang dilakukan oleh tim *Operation* juga berada di bawah pengawasan Bapak Ahmad Rizki selaku VP *Operation*. Koordinasi untuk melakukan penugasan serta waktu yang diberikan untuk melakukan tugas tersebut dilakukan pada aplikasi web KejarTugas yang disediakan oleh PT GIT. Untuk kebutuhan *Work from Home* (WFH), *meeting* dan diskusi dilakukan pada aplikasi *Gather*.

3.2 Tugas yang Dilakukan

Pelaksanaan tugas kerja magang yang dilakukan merupakan perancangan dan pembangunan aplikasi transaksi barang dan *chatbot*. Tugas ini dilakukan dalam beberapa tahap, yang dijelaskan secara rinci pada poin-poin berikut:

1. ***Explore***

   Pada tahap ini, penulis melakukan eksplorasi terhadap kebutuhan-kebutuhan aplikasi yang ingin dibangun. Dimulai dari teknologi apa yang ingin digunakan yang dapat memenuhi kriteria kebutuhan, anggaran yang dibutuhkan, struktur data serta *flow* aplikasi yang dibuat, serta solusi-solusi masalah yang mungkin akan dihadapi pada tahap pembangunan.

2. ***Development***

   Pada tahap ini, penulis melakukan pembangunan terhadap aplikasi-aplikasi yang sudah dirancang dan eksplorasi. Hal ini mencangkup pembangunan aplikasi dari migrasi *database*, *backend* dan *frontend* dari aplikasi tersebut, serta melakukan integrasi dengan API.

3. ***Testing***

Dalam tahap *testing*, penulis dan pihak-pihak internal GIT melakukan pengujian terhadap aplikasi yang telah dibuat sehingga aplikasi-aplikasi tersebut memenuhi kebutuhan dan juga tidak memiliki *bug* kritis.

3.3 Uraian Pelaksanaan Magang

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

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Perkenalan lingkungan kerja dan mempelajari struktur serta alur aplikasi Kejar Tugas dari sisi *frontend* dan *backend*.
2	Melanjutkan pembelajaran *backend* dan melakukan *sharing session* serta migrasi server awal.
3	Melakukan *bug fixing* terhadap migrasi server dan *setup* awal environement untuk proyek Tukerin.
4	Mengembangkan *frontend* Tukerin seperti *login/register*, profil, dan komponen menu utama.
5	Menginisasi diagram ER, tabel relasional *backend*, serta register *user* di Tukerin.
6	Membangun fitur *login*, CRUD *item*, dan *add* item pada *frontend* dan *backend* Tukerin.
7	Mengerjakan *fetch item*, *filtering kategori*, dan inisialisasi fitur *chat* Tukerin.
8	Mengembangkan dan menyempurnakan sistem notifikasi *real-time* untuk aplikasi Tukerin
9	Melanjutkan *caching* fitur *chat* dan menyelesaikan *build APK* serta *bug fixing* Tukerin.
10	Menyelesaikan fitur *chat* dan memperbaiki alur transaksi barter Tukerin.
11	Finalisasi fitur MVP tukar tambah.
12	Memulai proyek *chatbot* dengan eksplorasi topologi, riset AI, dan kebutuhan teknis.
13	Menyusun *flowchart* dan dokumentasi *chatbot* serta memperbaiki alurnya.
14	Melakukan *testing chatbot* WhatsApp dan *backend*, serta revisi dokumentasi dan *flowchart*.
15	Mempersiapkan *environment* dan mengatur *backend* chatbot di *cloud server*.
16	Melakukan diskusi, pembuatan *chatbot* internal GIT, serta optimisasi berdasarkan data.
17	Menambahkan integrasi Milvus, memperbaiki *frontend* dan menyempurnakan *chatbot*.

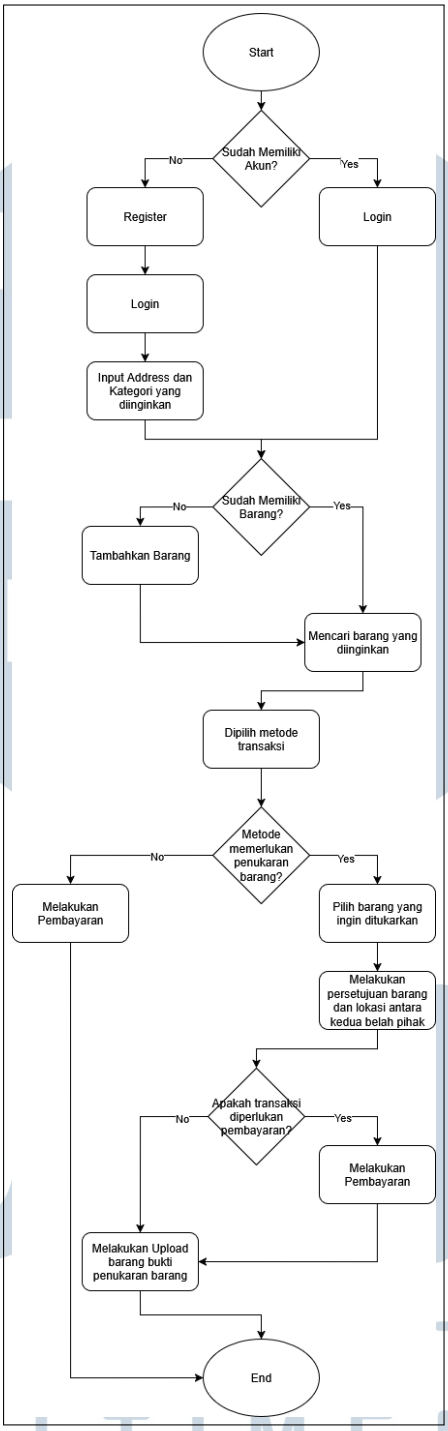
3.3.1 Aplikasi Transaksi Tukerin

Gambar 3.1. Logo Sementara Tukerin

Pada bab ini, penulis akan menjelaskan apa saja yang telah dilakukan oleh penulis saat melakukan proses pekerjaan magang. Pada saat ini, pengembangan aplikasi Tukerin masih berada pada tahap *prototype* dan belum dirilis secara resmi. Hingga akhir periode magang, fitur yang berhasil dikembangkan merupakan fitur barter dan tukar tambah, sedangkan untuk fitur lainnya seperti jual, lelang, dan *giveaway* belum dapat dikembangkan dikarenakan pemberhentian proyek sementara oleh pihak perusahaan.

Secara umum, alur dari pemakaian aplikasi dimulai dari pembuatan akun oleh pengguna. Namun, jika pengguna hanya ingin melakukan pencarian, pengguna dapat melakukan hal tersebut tanpa memiliki akun. Secara keseluruhan, alur yang digunakan untuk mengembangkan aplikasi ini dapat dilihat pada Gambar 3.2

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.2. Alur Pemakaian Aplikasi untuk Melakukan Transaksi

Sebelum masuk ke dalam detail teknis pengembangan, berikut merupakan tahapan perancangan yang telah dilakukan untuk membangun fondasi aplikasi Tukerin.

A Langkah-Langkah Perancangan Aplikasi Tukerin

Pengembangan aplikasi Tukerin mengikuti tahapan umum dalam proses rancang bangun perangkat lunak. Meskipun proyek ini masih berada dalam tahap *prototype*, beberapa langkah penting tetap dilakukan untuk memastikan kesesuaian fitur dengan kebutuhan pengguna. Tahapan-tahapan tersebut meliputi:

1. *Research:* Dilakukan studi terhadap aplikasi sejenis seperti OLX, Facebook Marketplace, dan Carousell untuk memahami keunggulan serta kekurangan sistem transaksi yang ada, terutama terkait fleksibilitas metode tukar-menukar.

2. *Requirement Analysis:* Tim melakukan diskusi dan pengumpulan kebutuhan fungsional berdasarkan ide awal perusahaan, termasuk fitur utama seperti barter, tukar tambah, jual, dan lelang. Fokus utama dalam tahap ini adalah fitur barter dan tukar tambah.

3. *Design:* Penulis membuat alur interaksi pengguna dan skema data dasar. Perancangan tidak mencakup desain visual antarmuka, namun lebih pada struktur API dan alur proses bisnis transaksi barang.

4. *Implementation:* Dilakukan pengembangan backend menggunakan bahasa Golang, React Native untuk frontend, dan PostgreSQL sebagai basis data. Penulis berkontribusi dalam pembuatan dan pengujian beberapa API inti dan fitur-fitur frontend yang dikembangkan.

5. *Testing:* Fitur-fitur yang telah dikembangkan diuji secara fungsional dan digunakan dalam uji coba pengguna internal.

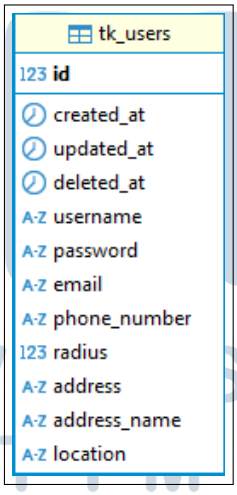
B Infrastruktur Backend Aplikasi Tukerin

Pengembangan *backend* pada aplikasi ini menggunakan bahasa pemrograman Golang dengan *framework* Gin. Pemilihan bahasa dan *framework* ini dikarenakan kebutuhan aplikasi yang cepat dengan performa yang tinggi [3]. Untuk sistem basis data, digunakan PostgreSQL dikarenakan kemampuannya untuk mengelola data relasional yang kuat, serta mampu menangani *query* yang kompleks dengan efisien[4]. DBMS (Database Management System) yang digunakan untuk mengelola data adalah DBeaver.

Proses interaksi antara *frontend* dengan *backend* dimulai dari pengiriman *request* atau permintaan dari *frontend* terhadap *backend*. Permintaan ini biasa berisi tipe *request* apa yang diinginkan, *route* apa yang dituju, serta jika dibutuhkan, data yang dikirimkan dalam bentuk JSON. Permintaan ini kemudian dialihkan ke salah satu *controller* yang dituju berdasarkan *route* dan jenis permintaan yang diterima. *Controller* kemudian akan memproses logika apa yang diperlukan, seperti validasi data, interaksi dengan sistem basis data, ataupun menjalankan pekerjaan lainnya. Pada akhirnya, *controller* akan mengembalikan respons yang sesuai berdasarkan hasil dari proses tersebut dalam bentuk JSON kepada aplikasi.

C Pembuatan API User Register

Dalam aplikasi Tukerin, *user* dapat melihat dan melakukan *search* tanpa memiliki akun. Namun untuk beberapa interaksi, seperti melakukan transaksi ataupun menambahkan barang, *user* perlu mendaftarkan akun Tukerin atau melakukan *login*. Dalam melakukan pendaftaran, *user* harus memasukkan *username*, *email*, *password*, dan nomor telepon. Setelah melakukan pendaftaran, *user* perlu melakukan *setup* awal, yaitu memasukkan lokasi alamat tempat tinggal serta kategori barang apa yang ingin dicari. Tampilan dari struktur tabel *user* dapat dilihat pada gambar 3.3.



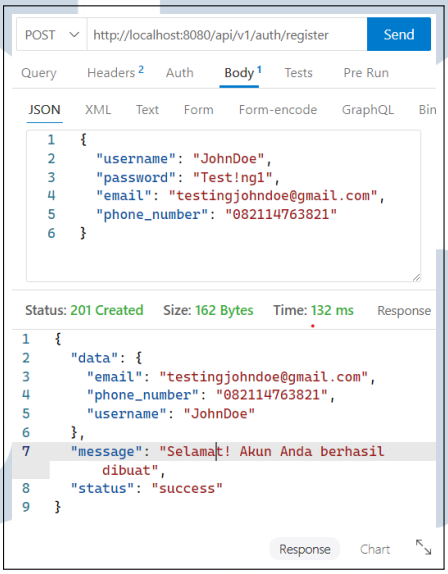
Gambar 3.3. Struktur Tabel *Users*

Dengan dilakukannya pembuatan akun, *user* dapat menggunakan fitur-fitur yang terdapat pada aplikasi, misalnya melakukan transaksi atau menambahkan barang pada akun sendiri. Ini dikarenakan proses-proses tersebut memerlukan

ID dari *user* untuk dapat mengidentifikasi hal-hal yang berasosiasi dengan *user* tersebut, seperti transaksi apa saja yang melibatkan *user* tersebut, ataupun kepemilikan item-item yang diperlihatkan oleh aplikasi.

C.1 Create User

Dalam pembentukan akun *user* baru, diperlukan informasi mengenai *username*, *email*, *password*, dan nomor telepon. Setelah melakukan input tersebut, *controller* pada sisi *backend* akan melakukan validasi pada informasi yang diterima. Setelah melakukan validasi terhadap input *user*, setiap input tersebut akan dilakukan verifikasi keberadaan data untuk menghindari duplikasi informasi dalam *database*. Sehabis semua verifikasi telah dilewati dengan sukses dan semua input telah dimasukkan dalam *database*, *controller* akan mengirimkan respons *success* jika berhasil atau *error* jika terjadi kesalahan dari permintaan pengguna. Contoh dari respons *success* dalam penggunaan API *Create User* dapat dilihat pada Gambar 3.4.

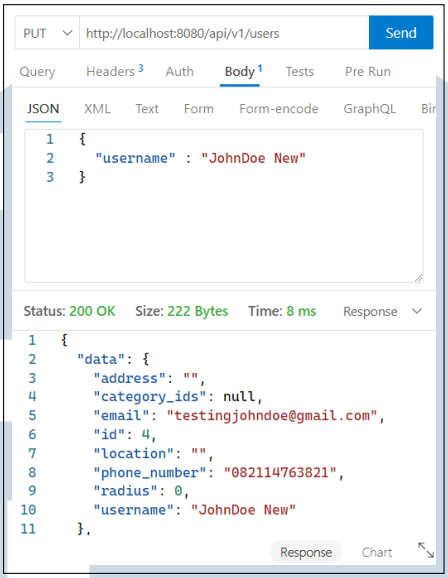


Gambar 3.4. Tampilan Percobaan API *Create User*

C.2 Update User

Setelah melakukan pembuatan akun, *user* juga dapat melakukan *update* atau pembaruan terhadap informasi-informasi pada akun tersebut. Informasi-informasi yang dapat diubah untuk saat ini merupakan *username*, *email*, *password*, dan nomor telepon akun. Contoh dari respons *success* dalam penggunaan API *update user*

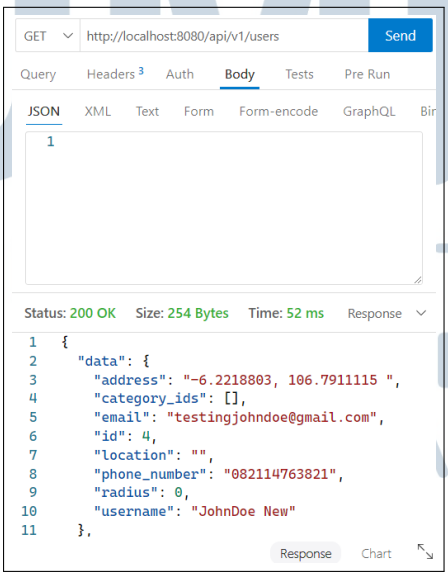
dapat dilihat pada Gambar 3.5.



Gambar 3.5. Tampilan Percobaan API *Update User*

C.3 Get User dengan ID

Untuk mendapatkan informasi-informasi mengenai salah satu akun yang terdaftar, maka hal yang dibutuhkan hanyalah ID yang berasosiasi dengan akun tersebut. ID yang didapatkan dalam bentuk token yang didapatkan saat dilakukannya *login*. Respons dari API ini dapat dilihat pada Gambar 3.6.



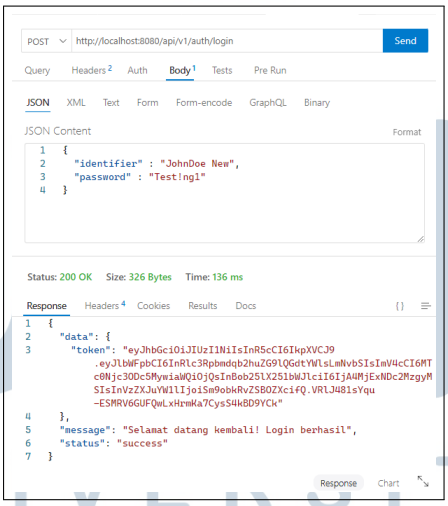
Gambar 3.6. Tampilan Percobaan API *Get User* dengan ID

D Pembuatan API Autentikasi User

Setelah *user* terdaftar dan informasi yang telah dimasukkan berada di *database*, *user* sekarang dapat melakukan *login* untuk mengakses fitur-fitur utama dalam aplikasi transaksi barang.

D.1 API Login

Saat *user* ingin melakukan *login*, user harus memasukkan informasi-informasi yang telah dimasukkan ke dalam *database*. Informasi yang dapat digunakan untuk melakukan *login* dapat berupa *username, email*, ataupun nomor telepon yang terdaftar. Setelah melakukan *login*, *controller* pada *backend* akan mengirimkan status *success* serta sebuah token yang digunakan untuk melakukan autentikasi dan yang memperbolehkan *user* untuk menggunakan semua fitur utama aplikasi. Token yang digunakan merupakan *JSON Web Token*, atau biasa disebut dengan JWT Token, dimana merupakan berbagai macam informasi yang digabungkan dan di-*encode* agar semua token merupakan token yang unik. Contoh dari respons *success* API *login* dapat dilihat pada Gambar 3.7.

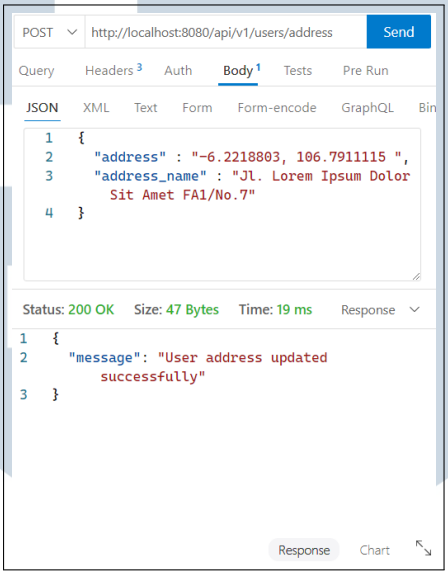


Gambar 3.7. Tampilan Percobaan API *Login User*

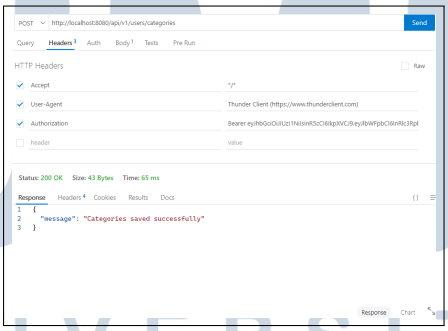
D.2 Update Alamat Rumah dan Kategori Favorit

Bagi *user* yang baru, *user* diperlukan untuk mengisi informasi tambahan mengenai alamat tempat tinggal serta kategori-kategori barang apa saja yang ingin dicari. Keberadaan informasi tersebut didapatkan dengan menggunakan ID yang

didapatkan dari token saat melakukan *login*. Informasi alamat rumah dan kategori favorit dibutuhkan untuk mengetahui keberadaan barang-barang yang dimiliki oleh *user* yang ingin dilakukan penukaran serta barang-barang apa saja yang ingin dicari oleh *user* tersebut. Untuk memasukkan informasi ini juga akan menggunakan API, namun berbeda dari API registrasi sebelumnya. Contoh respons *success* API ini dapat dilihat pada Gambar 3.8 dan Gambar 3.9



Gambar 3.8. Tampilan Percobaan API Input *User Address*

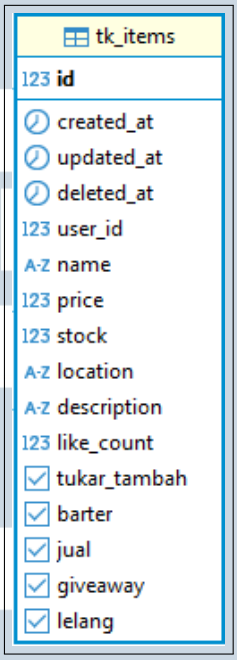


Gambar 3.9. Tampilan Percobaan API *Save User Categories*

E Pembuatan API Item

Setelah *user* telah memiliki akun, *user* dapat menambahkan barang-barang yang ingin ditransaksikan. Dengan memasukkan barang-barang ke dalam akun, *user* akan diperbolehkan untuk melakukan transaksi dengan *user* lainnya. Data-data yang diperlukan untuk menambahkan *item* adalah nama barang, gambar, deskripsi

barang, harga (opsional), jumlah stok, kategori dari barang tersebut, dan transaksi apa yang ingin dilakukan dengan barang tersebut. Gambaran struktur untuk tabel *items* atau barang-barang dapat dilihat pada Gambar 3.10

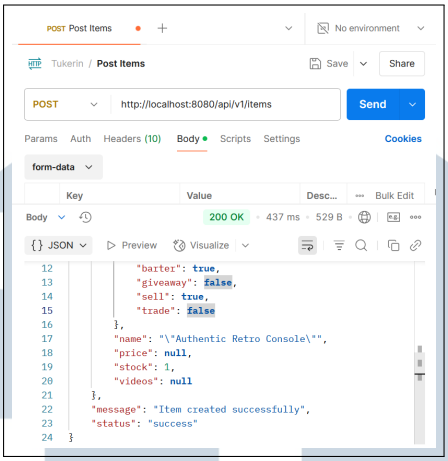


Gambar 3.10. Struktur Tabel *Items*

E.1 Create Item

Untuk menambahkan barang ke dalam akun, informasi yang dibutuhkan adalah *user ID*, yang didapatkan dari ID *user* yang memasukkan barang tersebut, nama dan harga barang, deskripsi, harga (opsional), jumlah stok, kategori, dan jenis transaksi, yang didapatkan dari input *user*, dan lokasi barang, yang didapatkan dari alamat *user* tersebut. Alasan mengapa input harga termasuk opsional adalah karena tidak semua jenis transaksi memerlukan harga untuk melakukan transaksi tersebut. Contoh dari respons *"success"* dapat dilihat pada Gambar 3.11

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.11. Tampilan Percobaan API *Create Items*

E.2 Get Item

API *Get Item* merupakan API yang digunakan untuk mendapatkan barang-barang yang dapat ditransaksikan dengan *user*. Kriteria barang yang didapatkan oleh *user* berdasarkan pemilihan kategori yang diinginkan, lokasi *user* saat ini, serta juga radius pencarian yang diinginkan. Untuk mendapatkan barang-barang yang berada di dalam radius, digunakan rumus haversine, yang merupakan rumus untuk menghitung jarak lingkaran besar antara dua titik berdasarkan lintang dan bujurnya, dengan memperhitungkan lengkungan Bumi. Bentuk dari formula tersebut adalah sebagai berikut:

$$\Delta\varphi = \varphi_2 - \varphi_1 \tag{3.1}$$

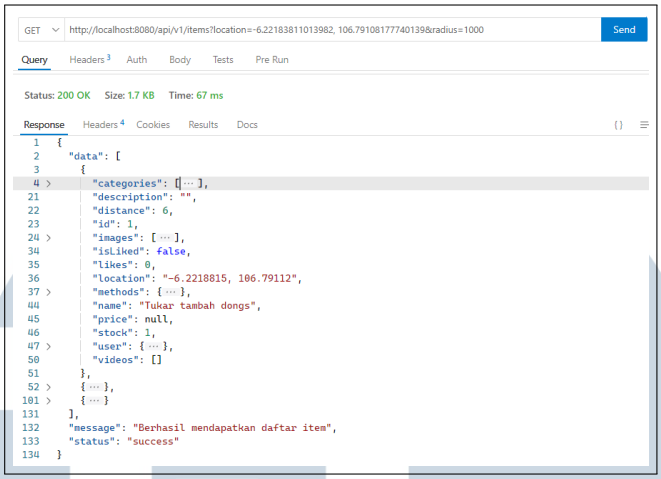
$$\Delta\lambda = \lambda_2 - \lambda_1 \tag{3.2}$$

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cdot\cos(\varphi_2)\cdot\sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{3.3}$$

$$c = 2\cdot\arctan 2\left(\sqrt{a}, \sqrt{1-a}\right) \tag{3.4}$$

$$d = R\cdot c \tag{3.5}$$

Contoh dari respons *success* yang didapatkan dari API *get items* dapat dilihat pada Gambar 3.12

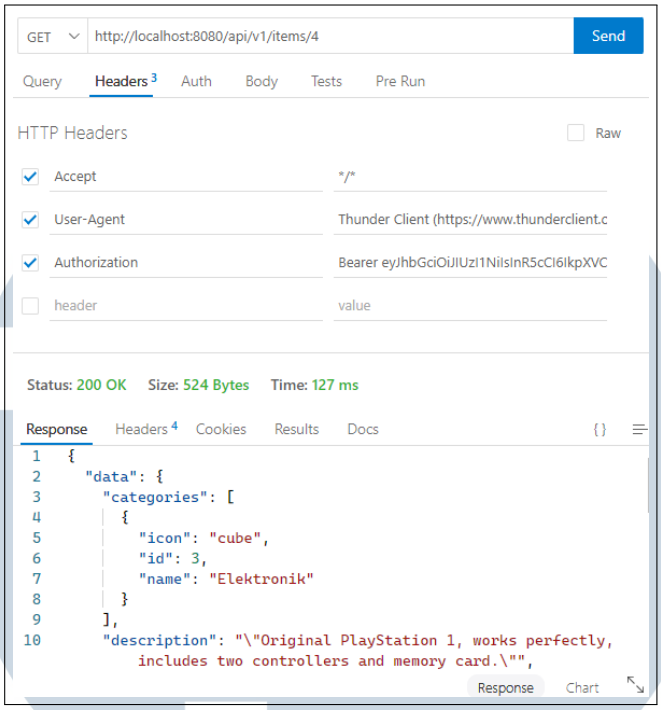


Gambar 3.12. Tampilan Percobaan API *Get Items*

E.3 Get Item By ID

Setelah mendapatkan barang yang ingin dicari, untuk mendapatkan informasi lebih detil mengenai barang tersebut, digunakan API *get item by id*, yang digunakan pada halaman detil produk. Informasi-informasi yang didapatkan dari barang tersebut merupakan semua informasi yang berkaitan. Informasi seperti deskripsi, foto-foto, video, pemilihan metode transaksi, dan lain-lain akan dikirimkan kepada *frontend* untuk diproses. Contoh dari respons *success* dapat dilihat pada Gambar 3.13

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

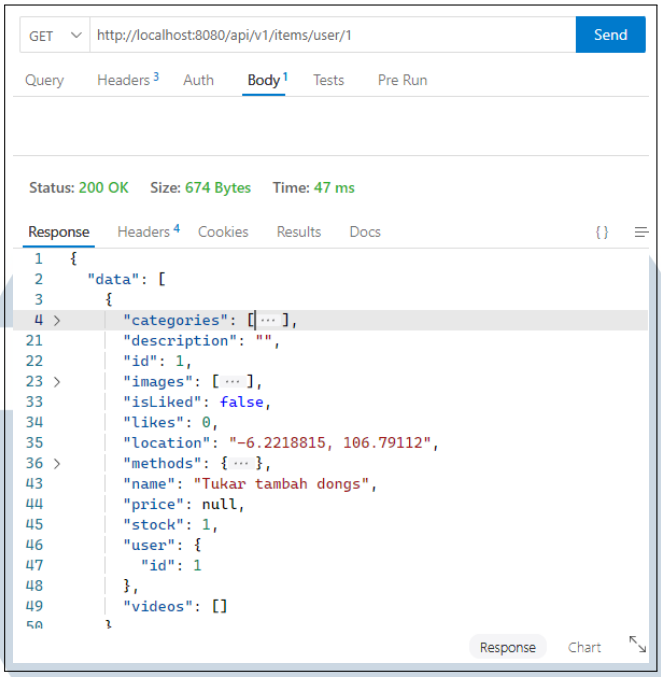


Gambar 3.13. Tampilan Percobaan API *Get Items by ID*

E.4 Get Item By User ID

API *Get Item By User ID* merupakan API yang bekerja untuk mendapatkan barang-barang apa saja yang dimiliki oleh pengguna tersebut. API ini dikirimkan dan digunakan pada halaman *profile* pengguna, dimana pengguna-pengguna lain dapat melihat apa yang dapat ditransaksikan oleh pengguna tersebut. Hasil dari panggilan API ini dapat dilihat pada Gambar 3.14.

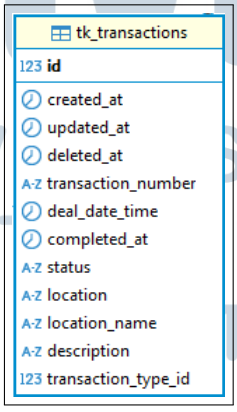
UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.14. Tampilan Percobaan API *Get Items by User ID*

F Pembuatan API Transaction

Setelah pengguna menemukan barang yang ingin ditukarkan dengan barang miliknya, informasi-informasi dan data akan diproses melalui API *transaction*. Tergantung pada pemilihan tipe transaksi, secara umum, API *transaction* ini meliputi transaksi yang berkaitan dengan pertukaran barang, transaksi menggunakan uang, serta juga gabungan dari kedua tipe transaksi tersebut. Bentuk dari struktur tabel transaksi dapat dilihat pada Gambar 3.15.



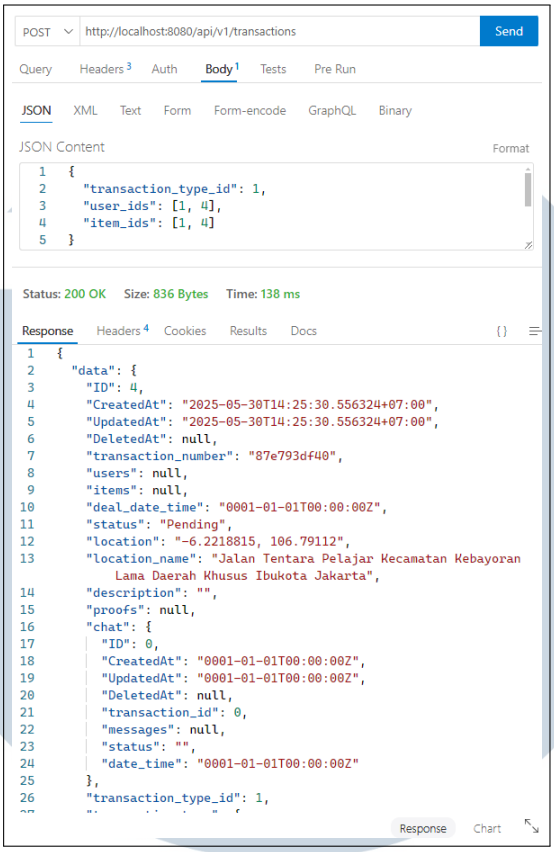
Gambar 3.15. Gambaran Struktur Tabel Transaksi

Dilihat dari tabel tersebut, kolom *"transaction_number"* digunakan untuk melakukan identifikasi terhadap nomor transaksi jika terjadi sebuah permasalahan. Kolom *"deal_date_time"* merupakan tempat yang akan digunakan untuk menyimpan tanggal dilakukannya transaksi jika sudah disetujui oleh kedua belah pihak. Kolom "status" digunakan untuk melihat proses transaksi yang sedang dijalankan oleh pengguna. Untuk sekarang ada 4 tipe status, dimana status *pending* memiliki arti bahwa transaksi sedang berada pada fase negosiasi, status *ongoing* adalah fase transaksi dimana para pengguna sudah melakukan perjanjian untuk tanggal dan tempat, status *rejected* adalah untuk transaksi yang ditolak oleh penggunanya, dan status *completed* dimana transaksi sudah berhasil dilakukan. Kolom *location* dan *location_name* digunakan untuk menyimpan lokasi tempat dilakukannya transaksi barang. Akhirnya, kolom *"description"* digunakan untuk menyimpan informasi tambahan untuk transaksi tersebut.

F.1 API Create Transaction

Dalam inisialisasi pembuatan transaksi, proses ini dimulai dari saat pengguna telah memilih barang apa yang ingin ditukarkan. Informasi ini kemudian dikirimkan ke *backend*, dimana semua hal yang diperlukan untuk melakukan transaksi diinisialisasi-kan. Hal ini termasuk pembuatan *chat room* untuk melakukan persetujuan transaksi, notifikasi transaksi terhadap pengguna untuk mengonfirmasi atau menolak transaksi barang tersebut, serta juga memproses barang-barang apa saja yang ingin ditransaksikan. Contoh dari respons yang akan dihasilkan jika sebuah transaksi telah dibuat dapat dilihat pada Gambar 3.16

UNIVERSITAS MULTIMEDIA NUSANTARA

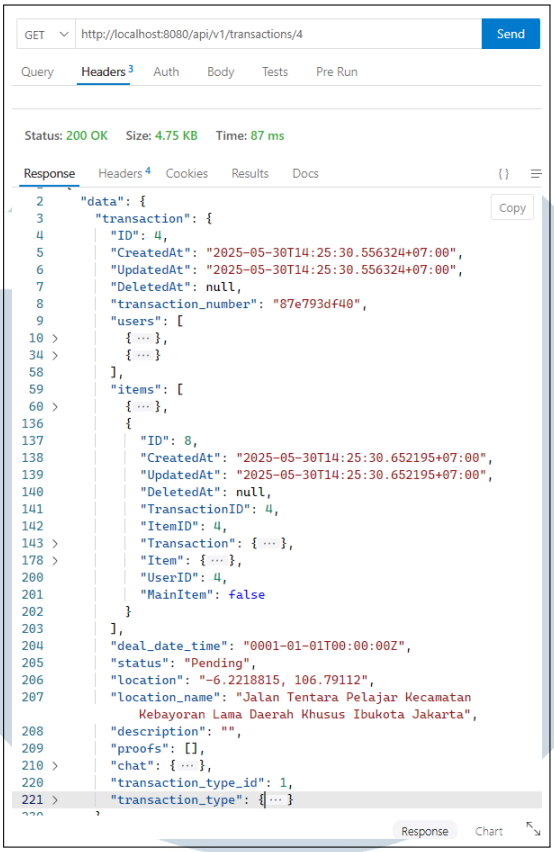


Gambar 3.16. Hasil *Success* API *Create Transaction*

F.2 API Get Transaction by User

API *Get Transaction by User* digunakan untuk mendapatkan detil transaksi dimana *user* tersebut berada sebagai pembeli atau *requester* untuk sebuah barang yang tidak dimiliki oleh *user* tersebut. API ini digunakan jika *frontend* membutuhkan informasi untuk semua transaksi yang *user* inisiasikan. Contoh dari respons *Success* serta data yang dikembalikan dapat dilihat pada Gambar 3.17

UNIVERSITAS MULTIMEDIA NUSANTARA

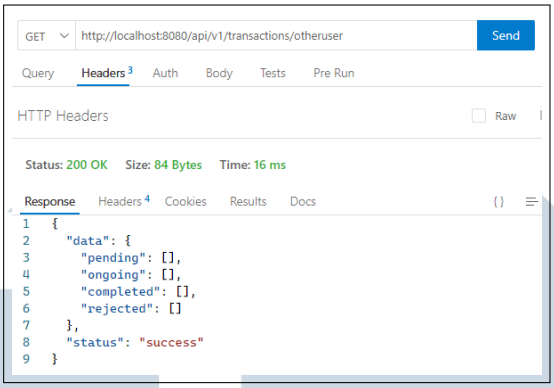


Gambar 3.17. Hasil *Success* API *Get Transaction by User ID*

F.3 API Get Transaction by Other User

API *Get Transaction by Other User* digunakan untuk mendapatkan daftar transaksi dimana *user* merupakan pemilik barang yang ingin ditransaksikan. API ini digunakan agar dapat membedakan fungsi dari API *Get Transaction by User*, dimana kedua halaman ini memiliki halaman yang berbeda untuk memanggil data-data yang berbeda. Contoh dari hasil respons *Success* serta data yang dikembalikan dapat dilihat pada Gambar 3.18.

UNIVERSITAS MULTIMEDIA NUSANTARA

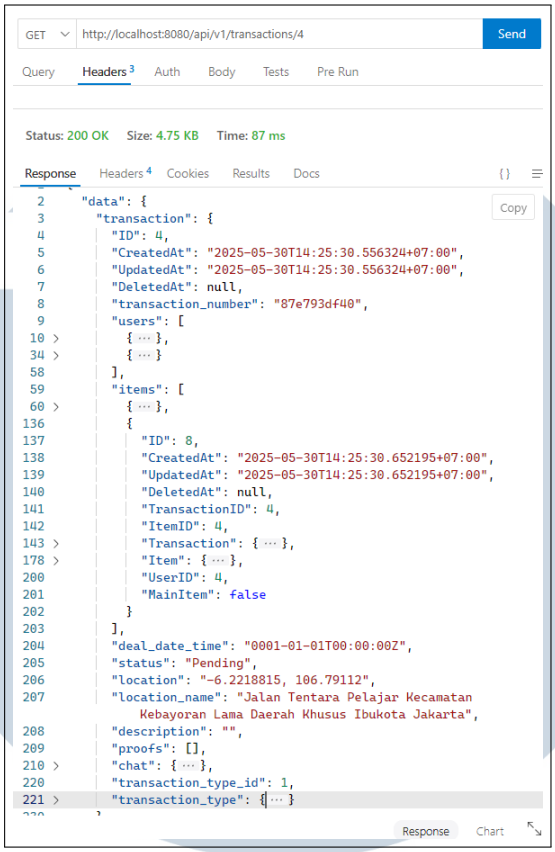


Gambar 3.18. Hasil *Success* API *Get Transaction by Other User*

F.4 API Get Transaction Details

Setelah melakukan inisialisasi pembuatan transaksi, *user* dapat melihat detil ini pada halaman transaksi tersebut. Halaman ini akan memanggil API *Get Transaction Details* untuk mendapatkan informasi-informasi mengenai transaksi tersebut. Informasi-informasi yang termasuk adalah barang-barang yang ingin ditransaksikan, gambar-gambar transaksi tersebut, data mengenai perbincangan *chat* yang telah dilakukan, kesepakatan tanggal dan waktu, serta status dari transaksi tersebut. Respons yang dikembalikan dapat berupa *success* ataupun *error*. Contoh dari respons *success* serta data-data yang dikirimkan dapat dilihat pada Gambar 3.19

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



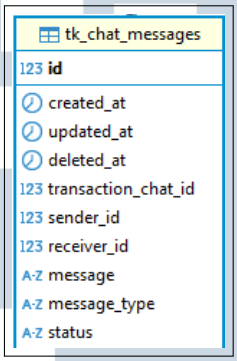
Gambar 3.19. Hasil *Success* API *Get Transaction Details*

F.5 API Respond to Transaction

API *Respond to Transaction* digunakan untuk melakukan persetujuan akhir antara kedua pengguna yang ingin melakukan transaksi. Hal ini dilakukan saat kedua pengguna tersebut telah menyetujui lokasi dan waktu transaksi yang ingin dilakukan. Untuk melakukan persetujuan ini, dilakukan oleh pengguna yang memiliki barang yang ingin ditransaksi, sedangkan pengguna yang menginisiasi transaksi adalah pengguna yang akan melakukan *request* tanggal berdasarkan kemauan kedua belah pihak. Maka dengan ini, pengguna yang dituju memiliki 2 pilihan: melakukan penolakan terhadap transaksi tersebut ataupun melakukan persetujuan. Jika dilakukan penolakan, maka proses transaksi akan berakhir dan status transaksi akan diperbarui menjadi *rejected*. Jika dilakukan persetujuan, maka transaksi tetap akan dilanjutkan ke tahap terakhir, dimana kedua pengguna akan melakukan transaksi pada tempat dan waktu yang telah ditentukan.

G Pembuatan Fungsi Chat

Dalam pembuatan fungsi *chat* dalam aplikasi ini, salah satu rintangan yang dihadapi merupakan masalah jika pengguna tidak menggunakan aplikasi ini untuk berkomunikasi satu sama lain dan berpindah aplikasi untuk melakukan transaksi. Maka dari ini, salah satu solusi merupakan untuk melakukan *chat* menggunakan *template* yang sudah disediakan. Bentuk dari struktur tabel *chat* dapat dilihat pada Gambar 3.20



Gambar 3.20. Gambaran Struktur Tabel Chat

Pada tabel ini, kolom *"sender_id"* dan kolom *"receiver_id"* digunakan untuk mengidentifikasi pengguna pada *chat bubble*-nya masing-masing. Kolom "*message*" digunakan untuk melihat apa yang dikirimkan oleh pengguna tersebut. Dan yang terakhir, kolom *"message_type"* digunakan untuk mengidentifikasi apakah pesan yang dikirimkan berupa sebuah pertanyaan yang memerlukan jawaban, dan jawaban apa yang bisa dikirimkan berdasarkan pertanyaan tersebut. Namun dikarenakan waktu *development* yang kecil, *protoype* chat hanya dilakukan pada *frontend*-nya saja.

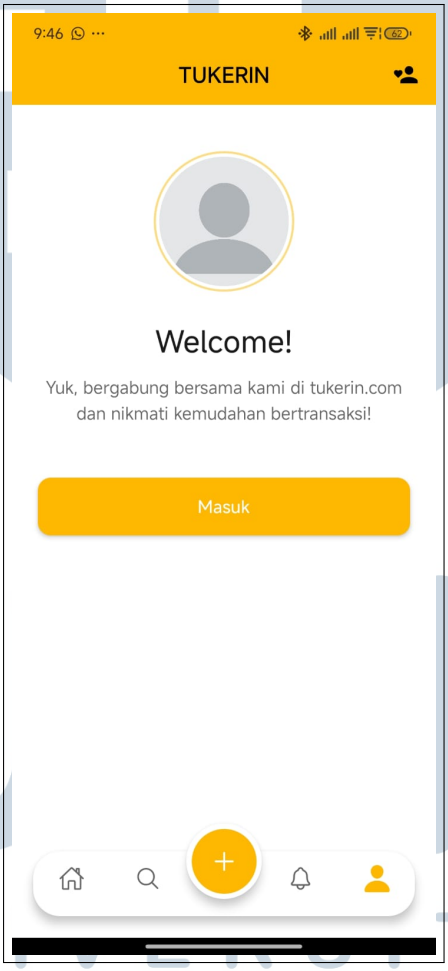
H Pembuatan Frontend Aplikasi Tukerin

Pengembangan *frontend* dari aplikasi Tukerin memiliki tujuan untuk menyediakan tampilan yang intuitif, responsif, dan mudah digunakan oleh pengguna dalam melakukan aktivitas transaksi. Aplikasi ini dikembangkan menggunakan *React Native* yang didukung oleh Expo. Pemilihan *framework* ini dikarenakan dukungan untuk pengembangan lintas platform, Android dan iOS, dengan satu basis kode yang efisien.

Pemilihan React Native didasarkan pada kemampuannya dalam membangun

aplikasi *mobile* secara cepat, modularisasi yang mudah, serta dokumentasi yang matang. Sementara itu, *framework* Expo dipilih untuk menyederhanakan proses *setup, testing, dan deployment* selama tahap pengembangan. Dengan kombinasi ini, pengembang dapat lebih fokus pada logika dan tampilan aplikasi tanpa harus menangani konfigurasi native yang kompleks.

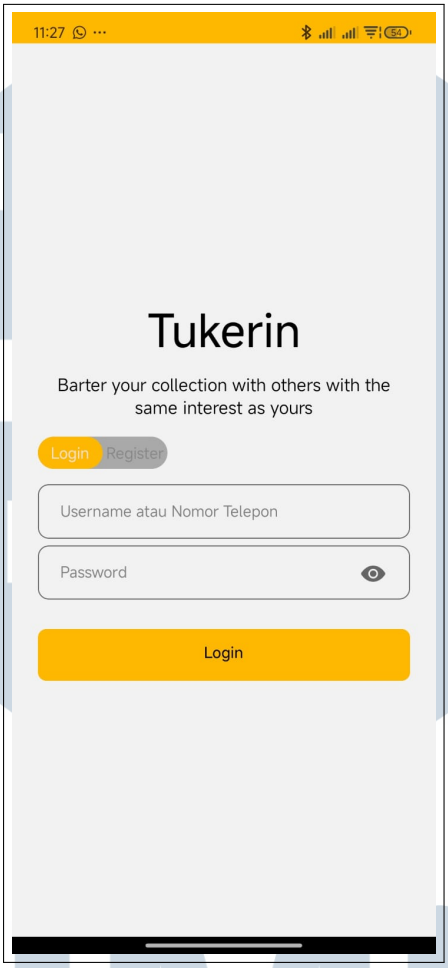
H.1 Pembuatan Halaman Login dan Register



Gambar 3.21. Layar sebelum melakukan *login* atau *register*

Untuk pengguna baru dan yang belum melakukan *login* pada aplikasi akan mendapatkan halaman sambutan untuk bergabung dan melakukan *register* ataupun *login* pada aplikasi Tukerin, seperti yang terlihat pada Gambar 3.21. Layar sambutan ini digunakan untuk meningkatkan *user experience* dalam penggunaan aplikasi. Dari layar ini, pengguna dapat menekan tombol ”masuk” untuk berpindah

ke halaman *login/register*.

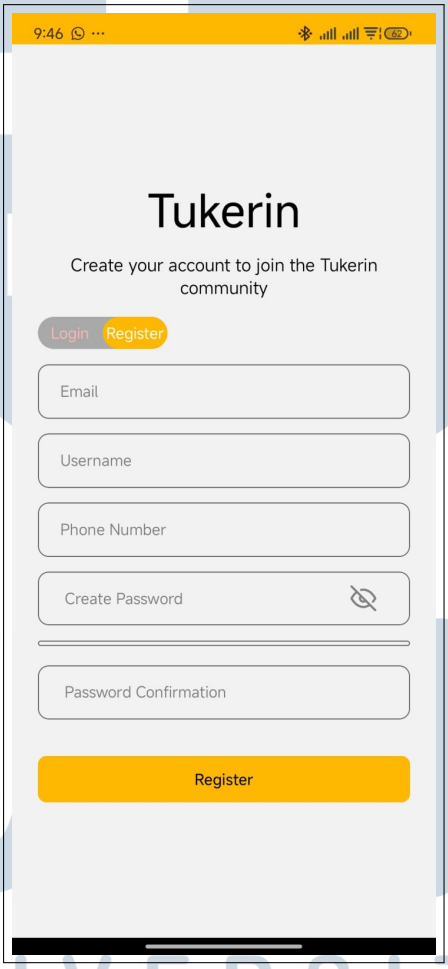


Gambar 3.22. Layar halaman *login*

Setelah menekan tombol ”masuk”, pengguna akan mendapatkan halaman ini. Halaman *login* ini dirancang sebagai pintu masuk utama bagi pengguna untuk mengakses fitur-fitur aplikasi, seperti melakukan transaksi dengan pengguna lain atau melihat *detail* barang pada aplikasi. Desain halaman ini menggunakan pendekatan minimalis, dimana akan fokus pada keterbacaan dan kejelasan interaksi. Seperti yang dapat dilihat pada Gambar 3.22, terdapat elemen input untuk *Username* atau *Nomor Telepon*. Pada *input* ini juga dapat menggunakan *email* untuk melakukan *login*. Terdapat juga elemen input *password*, disertai ikon ”eye” untuk melakukan *toggle* visibilitas *password*.

Secara teknis, penggunaan elemen input menggunakan *”TextInput”* yang merupakan bawaan dari React Native. Setelah menekan tombol *Login*, yang merupakan komponen dari *”Touchable Opacity”*, akan *trigger* fungsi autentikasi

yang memanggil API backend menggunakan Axios. Data yang dikirimkan akan dalam bentuk format JSON, dan jika autentikasi berhasil, pengguna akan diarahkan ke halaman utama aplikasi. Penanganan error atau input kosong juga telah ditambahkan untuk memberikan pengalaman pengguna yang lebih baik. Namun, jika pengguna tidak memiliki akun, pengguna dapat menekan tombol *Login/Register* yang berada di atap kiri untuk berpindah *tab* ke halaman *register*. Bentuk layar *register* dapat terlihat pada Gambar 3.23



Gambar 3.23. Layar halaman *Register*

Halaman registrasi dibuat untuk pengguna baru yang belum memiliki akun untuk melakukan *login*. Tampilan halaman registrasi terdiri dari beberapa kolom *input*, yaitu: *Email, Username,* Nomor Telepon, *Password,* dan Konfirmasi *Password*. Setiap input pada halaman ini memiliki autentikasi awal untuk melakukan pengecekan apakah *input* yang telah dimasukkan memiliki format yang benar, seperti *email*, panjang *password*, kriteria dari *password*, serta kesesuaian

antara *password* dan konfirmasinya.

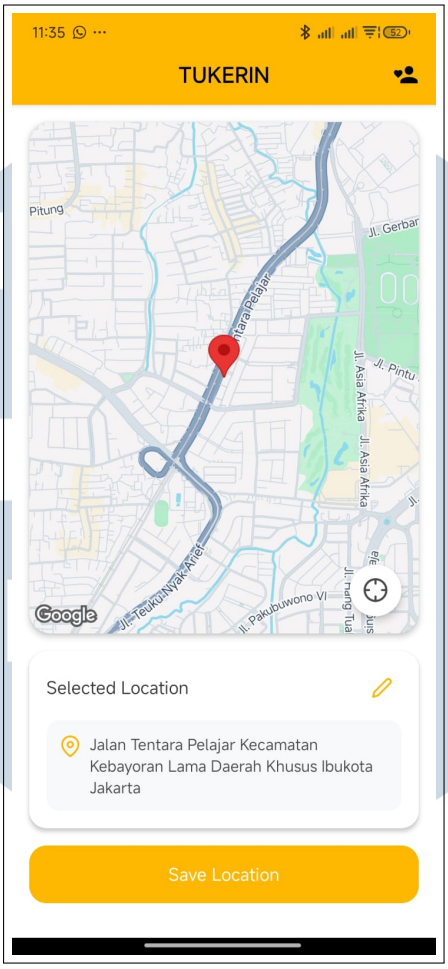
Setiap *input field* menggunakan *Text Input* yang telah dikustom agar dapat memenuhi syarat-syarat untuk masing-masing tipe data. Setelah menekan tombol *"Register"*, sebuah fungsi akan dipanggil untuk mengirim data ke *endpoint backend register*. Validasi akan dilakukan di sisi *frontend* untuk meminimalkan *error* dari sisi server. Jika registrasi berhasil, pengguna akan dialihkan ke halaman *login* untuk melakukan *login* pada aplikasi ini.

Dari sisi UI, navigasi antara *tab "Login"* dan *Registrasi* didesain dalam satu screen menggunakan *tab-switch* berdasarkan *state* yang aktif. Desain *form* mempertahankan gaya visual yang konsisten dengan *brand* Tukerin, menggunakan warna dominan kuning sebagai penanda aksi utama.

H.2 Pembuatan Halaman Pendaftaran Alamat dan Kategori

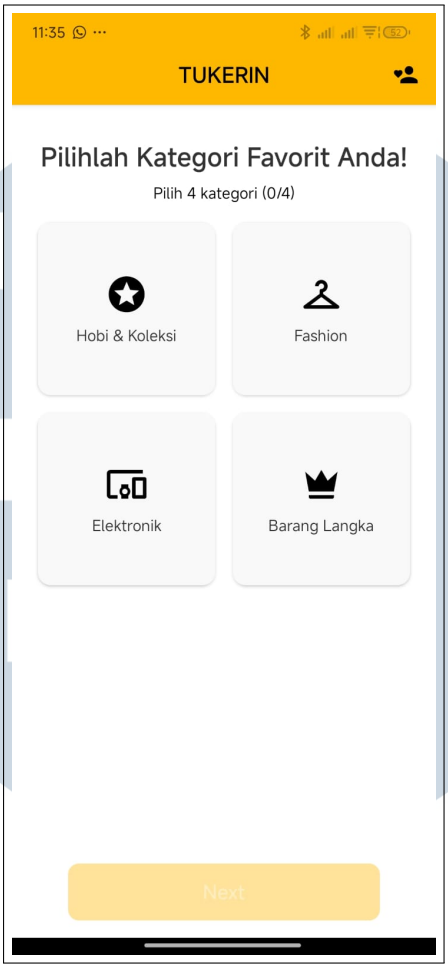
Setelah *user* berhasil melakukan proses registrasi dan login untuk pertama kalinya, halaman akan secara otomatis berpindah, mengarahkan pengguna ke dua tahap tambahan, yaitu melakukan pendaftaran lokasi dan pemilihan kategori yang ingin dicari. Tahap ini dirancang untuk melengkapi profil pengguna dan juga memungkinkan sistem untuk memberikan rekomendasi barang berdasarkan apa yang dipilih oleh pengguna.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.24. Layar halaman pendaftaran lokasi (alamat)

Seperti yang dapat terlihat pada Gambar 3.24, pengguna diminta untuk memilih lokasi rumah pengguna melalui peta tersebut. Dengan bantuan *Global Positioning System* (GPS), aplikasi akan secara otomatis mendapatkan lokasi pengguna saat ini, seperti yang ditandai dengan pin merah pada tampilan peta. Namun jika pengguna sedang berada di luar rumah, pengguna dapat menggunakan peta untuk menentukan titik rumah secara manual. Alamat lengkap yang dipilih melalui peta juga akan ditampilkan di bawah peta. Lokasi ini akan disimpan dalam *database* pengguna melalui API ketika pengguna menekan tombol *Save Location*. Data lokasi ini nantinya akan dipakai sebagai acuan untuk menampilkan produk atau pengguna terdekat agar proses transaksi barang menjadi lebih efisien dan kontekstual.



Gambar 3.25. Layar pemilihan kategori favorit

Setelah melakukan registrasi lokasi, pengguna akan diarahkan ke halaman berikutnya seperti pada Gambar 3.25. Pada halaman ini, pengguna akan diminta untuk memilih beberapa kategori barang favorit dari beberapa opsi yang tersedia. Dikarenakan kebutuhan *development*, hanya 4 kategori yang tersedia, yaitu Hobi dan Koleksi, *Fashion*, Elektronik, dan Barang Langka. Tujuan dari pemilihan kategori ini adalah untuk melakukan pengaturan terhadap preferensi pengguna dalam melakukan pencarian barang atau penawaran barter. *Frontend* kategori dibuat dengan komponen berbentuk kartu yang dapat di-*toggle* jika ingin dilakukannya pemilihan. Kategori yang terpilih kemudian akan dikirimkan kepada *backend* dan disimpan sebagai bagian dari profil *user*.

Dari segi teknis, data dari dua tahap ini akan dilengkapi sebagai bagian dari profil pengguna dan hanya akan diminta satu kali saat pertama kali *login*. Jika pengguna belum melengkapi data ini dan keluar dari aplikasi, sistem akan secara

otomatis menampilkan halaman-halaman ini terlebih dahulu sebelum pengguna dapat mengakses fitur utama aplikasi Tukerin.

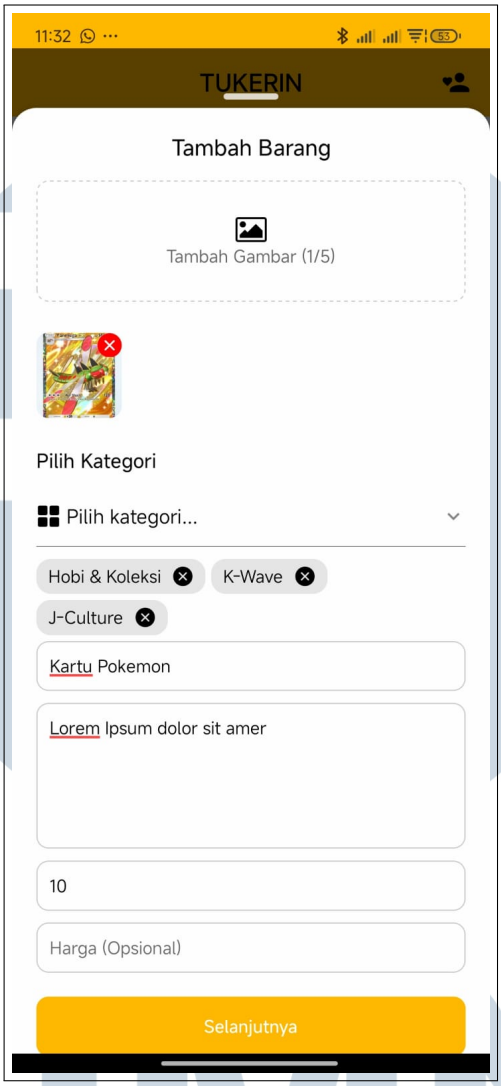
H.3 Pembuatan Fitur Penambahan Barang

Fitur penambahan barang merupakan salah satu fitur inti dari aplikasi Tukerin, yang akan memungkinkan para pengguna untuk melakukan pengunggahan barang milik mereka yang ingin ditawarkan kepada pengguna lain melalui berbagai macam metode transaksi. Akses terhadap fitur ini dilakukan melalui tombol ”+” yang terletak pada bagian tengah *bottom navigation bar*, yang terlihat pada Gambar 3.26. Tombol ”+” ini ditonjolkan dengan ikon berwarna kuning dengan dimensi yang berbeda dari tombol-tombol lainnya, melambangkan tombol tersebut sebagai tombol aksi utama.

Gambar 3.26. Tampilan *bottom navigation bar* dengan tombol tambah

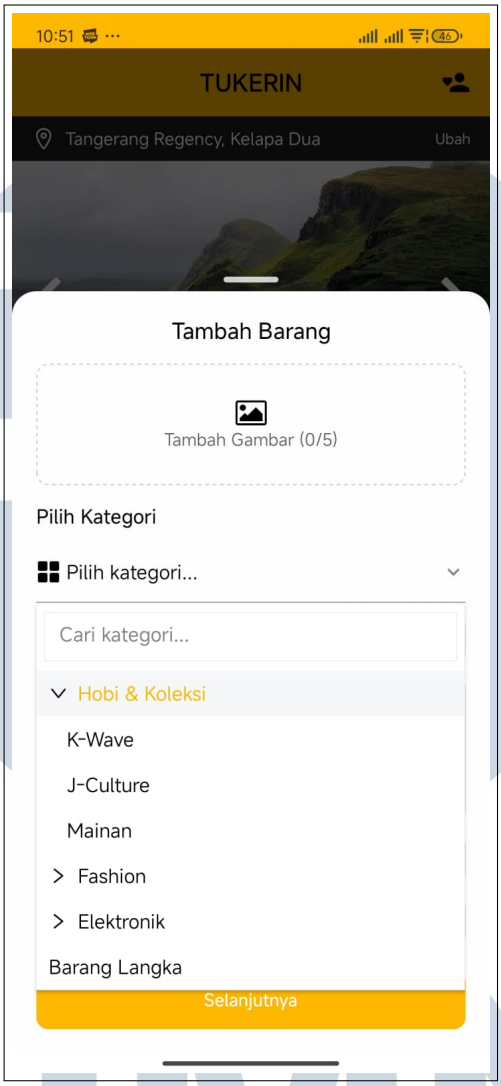
Setelah menekan tombol tersebut, pengguna akan diarahkan ke sebuah *pop-up* untuk melakukan penambahan barang seperti pada Gambar 3.27. Pada tahap pertama ini, pengguna diminta untuk melakukan pengisian terhadap detil produk yang akan ditambahkan, seperti gambar atau video yang dimaksimalkan menjadi 5 (maksimal video yang ditampilkan merupakan 1), nama barang, kategori barang, deskripsi barang, jumlah atau stok barang, serta harga yang bersifat opsional.

UNIVERSITAS MULTIMEDIA NUSANTARA



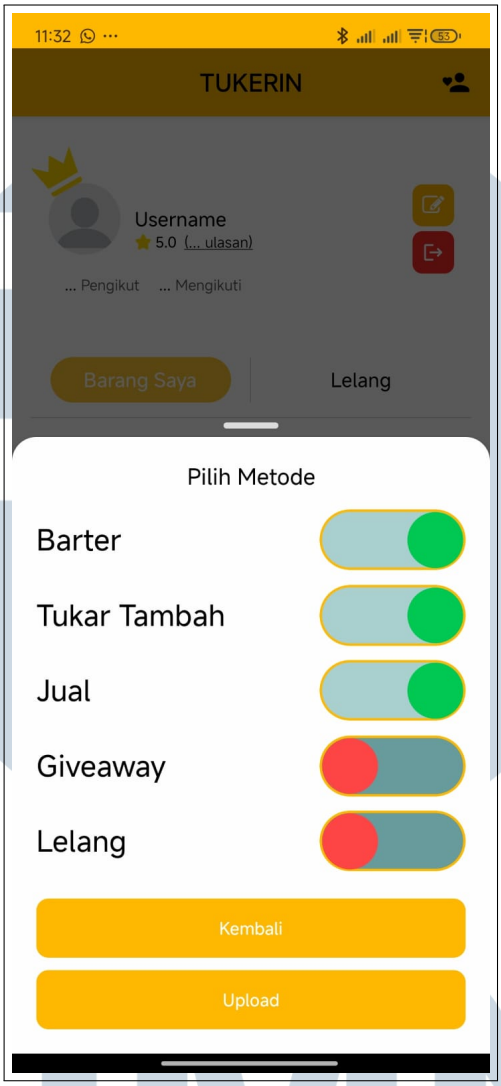
Gambar 3.27. Formulir penambahan barang

Pemilihan kategori dilakukan melalui *dropdown menu* yang mendukung kategori bertingkat atau kategori beranak, seperti yang terlihat pada Gambar 3.28. Sebagai contoh, pengguna dapat memilih kategori utama sebagai Hobi dan koleksi, kemudian memilih kategori *K-Wave* atau *J-Culture*. Dengan ini, pengguna dapat memilih beberapa kategori sekaligus, tergantung pada kesesuaian karakteristik barang. Hal ini juga yang akan menjadi salah satu kriteria untuk memunculkan barang berdasarkan kategori yang sebelumnya sudah dipilih oleh pengguna.



Gambar 3.28. Pemilihan kategori barang bertingkat

Setelah proses tersebut sudah terlewati, pengguna dapat menekan tombol ”selanjutnya”, yang akan melanjutkan proses penambahan barang, yakni melakukan pemilihan metode transaksi, yang dapat terlihat pada Gambar 3.29. Untuk saat ini, aplikasi Tukerin akan direncanakan untuk mendukung lima jenis metode transaksi, yaitu barter, tukar tambah, jual, giveaway, ataupun lelang.



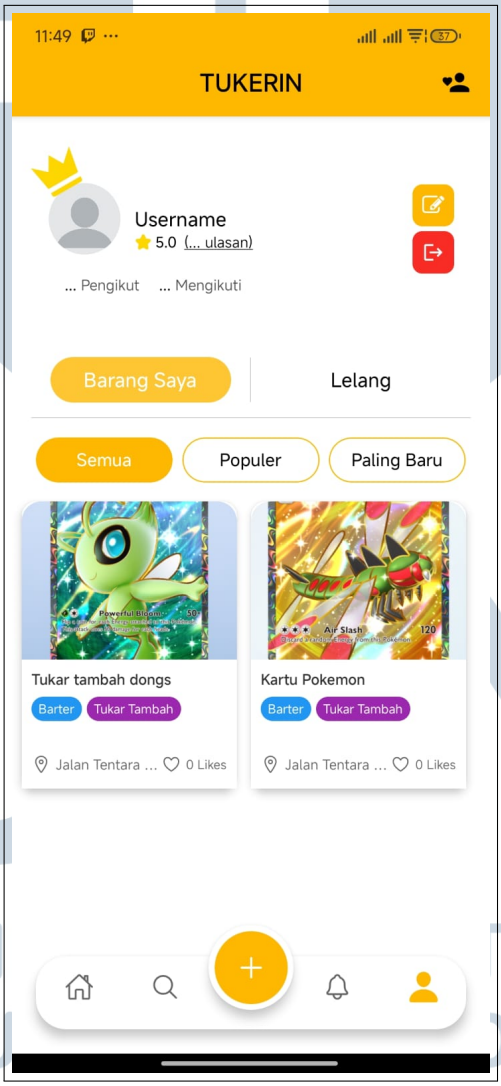
Gambar 3.29. Pemilihan metode transaksi barang

Dalam pemilihan jenis transaksi, pengguna dapat memilih barter, tukar tambah, dan jual secara bersamaan, namun metode *giveaway* dan lelang hanya dapat dipilih secara eksklusif. Ini dikarenakan kedua metode ini tidak bisa digabungkan dengan metode lain. Mekanisme ini dibuat untuk memudahkan penyesuaian jenis transaksi terhadap kondisi barang dan preferensi pengguna.

Setelah proses dalam melakukan pemilihan metode selesai, pengguna dapat menekan tombol *Upload* untuk menyimpan barang ke dalam sistem. Semua data yang telah dimasukkan akan langsung dikirimkan ke *backend* melalui API dalam format JSON. Setelah berhasil, barang pengguna yang sudah terunggah dapat dilihat pada daftar barang pengguna ataupun listing *global*, berdasarkan kategori dan juga lokasi barang.

H.4 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 3.30

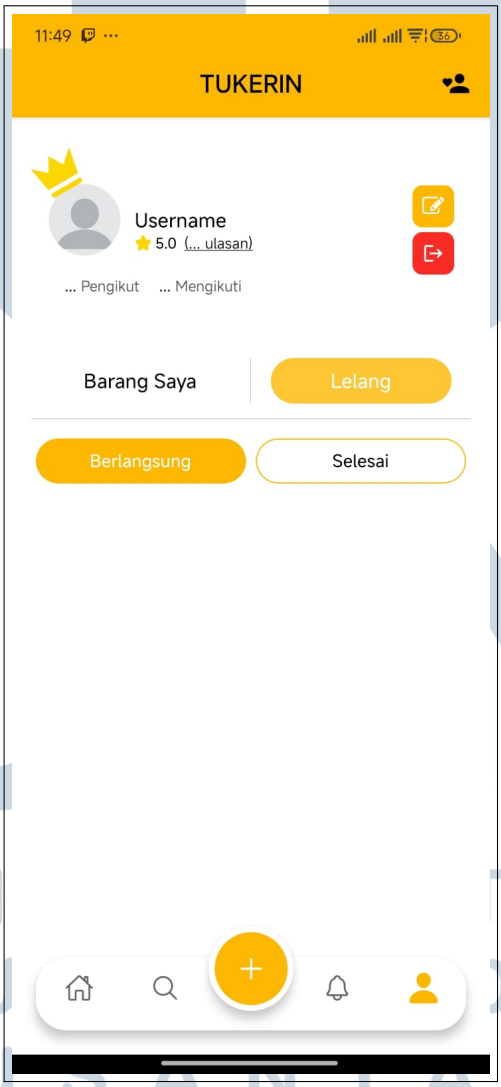


Gambar 3.30. Tampilan awal halaman profil pengguna

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 3.31. Tampilan daftar barang pengguna dan tab lelang

Sementara itu pada *tab* lelang, yang dapat dilihat pada Gambar 3.31, akan memisahkan barang berdasarkan statusnya, yakni berlangsung dan selesai, agar

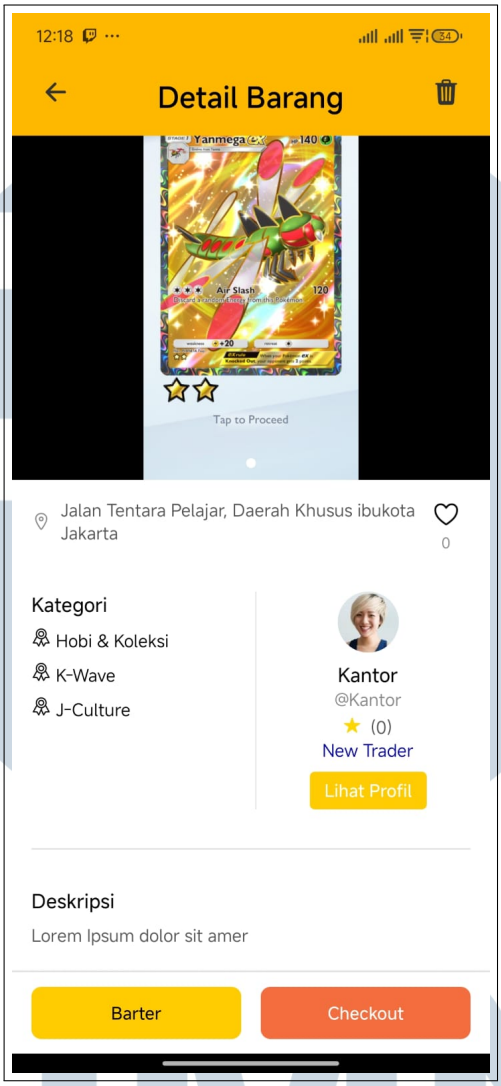
pengguna dapat dengan mudah untuk melakukan pemantauan terhadap proses barang-barang yang sedang dilelang. Pemisahan ini dilakukan untuk merapikan halaman profil agar lebih ter-organisir dan juga memudahkan pengguna untuk mengelola aktivitas mereka pada aplikasi ini.

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.

H.5 Pembuatan Halaman Detil Barang

Halaman detail barang digunakan untuk menampilkan informasi sebuah barang secara lengkap serta visual yang di-*posting* oleh pengguna lain ataupun sendiri. Tampilan halaman utama ini terdiri dari foto barang pada paling atas halaman, lokasi barang tersebut, kategori, informasi mengenai pemilik barang tersebut, deskripsi, dan dua tombol aksi utama dibagian bawah, yaitu pemilihan transaksi yang ingin dipilih serta *checkout*. Contoh dari tampilan barang ini dapat terlihat pada Gambar 3.32

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



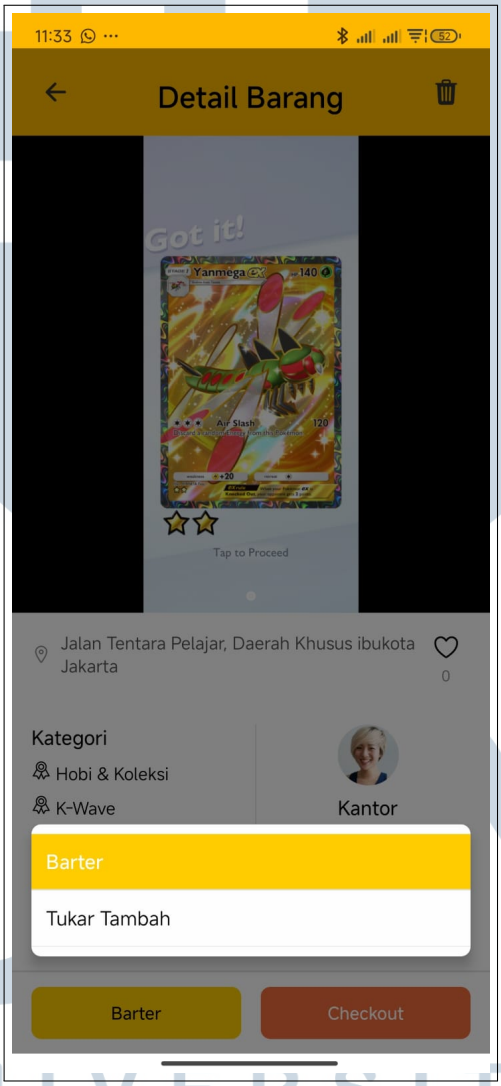
Gambar 3.32. Tampilan halaman detail barang

Pada bagian atas halaman, jika terdapat beberapa media maka dapat di-*swipe* untuk melihat media lainnya. Lokasi dari barang dapat dilihat secara eksplisit di bawah gambar tersebut, dengan tombol like atau melakukan *wishlist* pada bagian kanan. Setelah itu pada bagian bawahnya, terlihat informasi pengguna yang menggunggah barang yang ditampilkan dengan lengkap dengan nama, username, *rating*, dan tombol untuk melihat profil.

Kategori produk ditampilkan dalam daftar vertikal yang dapat memungkinkan pengguna untuk memahami jenis barang apa yang sedang dilihat. Dan yang terakhir adalah bagian deskripsi pada bagian bawah halaman untuk melihat detil dari barang apa yang sedang ditawarkan. Seluruh informasi yang ditampilkan ini ditarik dari *backend* melalui API berdasarkan ID dari barang

yang dipilih dari halaman sebelumnya.

Ketika barang tersebut memiliki beberapa pilihan untuk melakukan transaksi, pengguna dapat menekan tombol pada bagian paling bawah kiri. Tombol ini akan memunculkan *popup* pilihan metode transaksi, seperti yang terlihat pada Gambar 3.33



Gambar 3.33. Popup pilihan metode saat menekan tombol Barter

Setelah metode pemilihan dipilih dan tombol *checkout* telah ditekan, aplikasi akan mengarahkan pengguna ke halaman berikutnya berdasarkan jenis transaksi yang telah dipilih. Pada transaksi dengan menggunakan metode barter atau tukar tambah, pengguna akan langsung diarahkan kepada halaman pemilihan barang milik mereka sendiri untuk ditawarkan sebagai barang tukar. Tampilan dari halaman ini berfungsi untuk menampilkan daftar barang yang sebelumnya

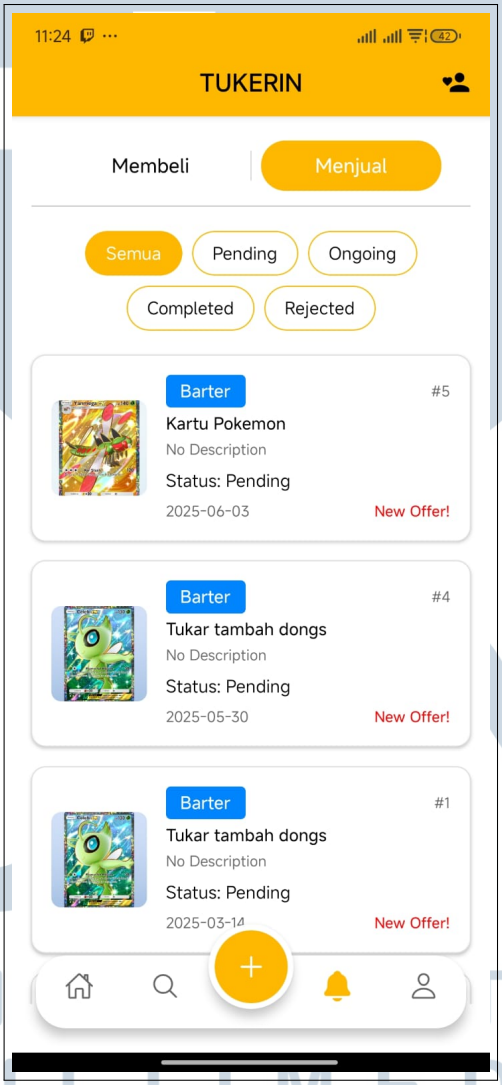
telah diunggah oleh pengguna, yang dapat dipilih sebagai bagian dari penawaran. Pengguna cukup untuk memilih satu barang pada halaman tersebut lalu menekan tombol *continue* untuk melanjutkan proses ke tahap selanjutnya. Contoh pemilihan barang dapat dilihat pada Gambar 3.34.

Gambar 3.34. Halaman pemilihan barang milik pengguna

Seperti yang dapat dilihat pada contoh tersebut, setiap barang akan ditampilkan dalam bentuk kartu yang berisi gambar dan nama barang, yang dapat diklik sebagai pilihan. Untuk memudahkan pengguna untuk mendapatkan barang apa yang ingin dicari, pengguna dapat menekan kolom input pencarian pada atas halaman dan mencari nama barang apa yang ingin dipilih.

H.6 Pembuatan Halaman Notifikasi dan Riwayat Transaksi

Halaman notifikasi pada aplikasi Tukerin dibuat untuk membantu pengguna dalam memantau seluruh aktivitas transaksi yang sedang berlangsung maupun yang telah selesai. Halaman ini akan menampilkan daftar barang apa saja yang sedang dilakukan proses transaksi, baik sebagai pembeli ataupun penjual, dengan status yang berdasarkan dari interaksi kedua belah pihak.



Gambar 3.35. Tampilan halaman notifikasi transaksi

Seperti yang terlihat dari Gambar 3.35, halaman ini terdiri dari dua bab utama, yaitu ”Membeli” dan ”Menjual”, yang menunjukkan peran pengguna dalam sebuah transaksi tertentu. Setiap *tab* memiliki filter tertentu berdasarkan status, seperti semua, *pending* untuk transaksi yang sedang dalam tahap tawar-menawar,

ongoing untuk transaksi yang sudah disepakati kedua belah pihak dan menunggu pelaksanaan transaksi, *completed*, dan *rejected*. Filter ini ditampilkan dalam bentuk tombol *tab rounded* agar dapat terlihat lebih mudah.

Setiap kartu transaksi menampilkan informasi penting seperti jenis metode transaksi, nama barang, id dari transaksi tersebut, status transaksi, tanggal inisiasi transaksi, serta peringatan "*New Offer!*" bagi transaksi yang masih baru.

Data transaksi ini telah dilakukan *fetch* dari *backend* melalui endpoint transaksi sesuai dengan user ID saat login. Dan untuk lebih mempermudah pengguna untuk mencari notifikasi mereka, setiap warna dari *badge* dipakai untuk membedakan jenis-jenis transaksi yang telah ditampilkan.

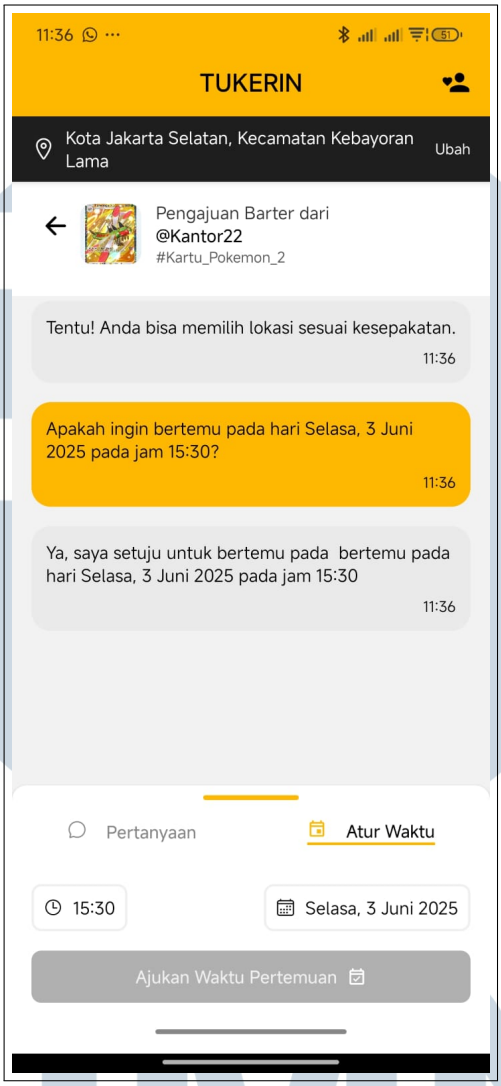
H.7 Pembuatan Prototype Chat

Halaman *chat* ini memiliki fungsi sebagai ruang komunikasi antara kedua pengguna yang sedang melakukan transaksi. Halaman ini dapat dilihat pada bagian paling bawah detil transaksi, yang dapat diakses setelah melakukan pemilihan barang dalam proses *checkout* ataupun melalui notifikasi. Halaman ini digunakan untuk melakukan konfirmasi detil pertemuan , waktu, serta melakukan diskusi ringan terkait barang yang ingin ditawarkan. Contoh dari halaman *chat* ini dapat dilihat pada Gambar 3.36.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

Gambar 3.36. Tampilan halaman chat transaksi

Dapat terlihat pada bagian atas, pengguna dapat melihat nama pengguna pengaju, barang yang ingin diajukan, serta lokasi transaksi yang sudah ditentukan secara *default* berdasarkan lokasi barang. Fitur utama dari halaman ini merupakan kolom percakapan, dimana pengguna dapat melihat riwayat balasan dari kedua belah pihak, tombol waktu dan tanggal pertemuan, dimana salah satu dari pengguna dapat mengajukan jadwal untuk bertemu, serta pertanyaan cepat seperti ”Saya OTW” atau ”Kondisi barang bagus?”.



Gambar 3.37. Fitur pengaturan waktu dan pertanyaan cepat

Karena pertimbangan efisiensi untuk tahap prototipe ini, lokasi pertemuan sudah secara otomatis ditentukan berdasarkan lokasi barang, sehingga pengguna hanya perlu untuk melakukan kesepakatan tanggal dan waktu pertemuan. Setelah salah satu pengguna telah melakukan pengajuan untuk waktu dan tanggal bertemu, pihak lainnya dapat menyetujui langsung dari pesan yang tampil.

Setelah sebuah kesepakatan terhadap tanggal dan waktu telah ditentukan, maka status transaksi akan diperbarui menjadi *"Ongoing"*, menandakan bahwa transaksi telah siap untuk dilakukan secara langsung.

I Testing Aplikasi

Dalam proses pengembangan aplikasi Tukerin, dilakukan beberapa tahap pengujian untuk memastikan bahwa sistem berjalan sesuai dengan kebutuhan fungsional serta memberikan pengalaman yang baik bagi pengguna. Pengujian dilakukan dalam dua jenis utama, yaitu *Functional Testing* dan *Usability Testing*.

I.1 Functional Testing

Functional Testing bertujuan untuk memastikan bahwa setiap fitur dalam aplikasi dapat berjalan dengan benar sesuai dengan kebutuhan dan spesifikasi yang telah ditentukan. Pengujian ini dilakukan terhadap berbagai endpoint API serta fitur yang telah dibangun, baik oleh penulis maupun oleh tim pengembang lainnya.

Hasil pengujian menunjukkan bahwa semua fungsi utama telah berjalan dengan baik. Beberapa komponen utama yang diuji antara lain:

- API Item: Pengujian terhadap fitur *Create, Read, Update, Delete* (CRUD) item berhasil dilakukan tanpa kendala.
- API Transaction: Fitur transaksi berjalan dengan baik, mencakup input data transaksi hingga pencatatan detail barang yang dipertukarkan.
- API Autentikasi User: Mekanisme login dan validasi token telah berhasil diuji sesuai skenario yang dirancang.
- API User Register: Proses registrasi pengguna baru berhasil dan tersimpan dengan benar di basis data.

Selain itu, beberapa fitur tambahan yang dikembangkan oleh rekan tim juga diuji dan dinyatakan berjalan sesuai dengan kebutuhan fungsional sistem.

I.2 Usability Testing

Usability Testing difokuskan pada aspek kenyamanan dan kemudahan pengguna dalam mengakses dan memahami aplikasi. Pengujian dilakukan secara internal dengan meminta beberapa pengguna untuk mencoba alur transaksi seperti barter.

Dari hasil pengujian, ditemukan bahwa secara visual aplikasi cukup informatif dan mudah dipahami. Tata letak dan desain antarmuka membantu

pengguna mengenali fungsi-fungsi utama secara cepat. Namun, terdapat beberapa masukan dari pengguna terkait alur transaksi barter dan transaksi kompleks lainnya. Beberapa pengguna menyatakan kebingungan dalam memahami langkah-langkah melakukan barter antar pengguna, terutama dalam hal konfirmasi dan penyelesaian transaksi.

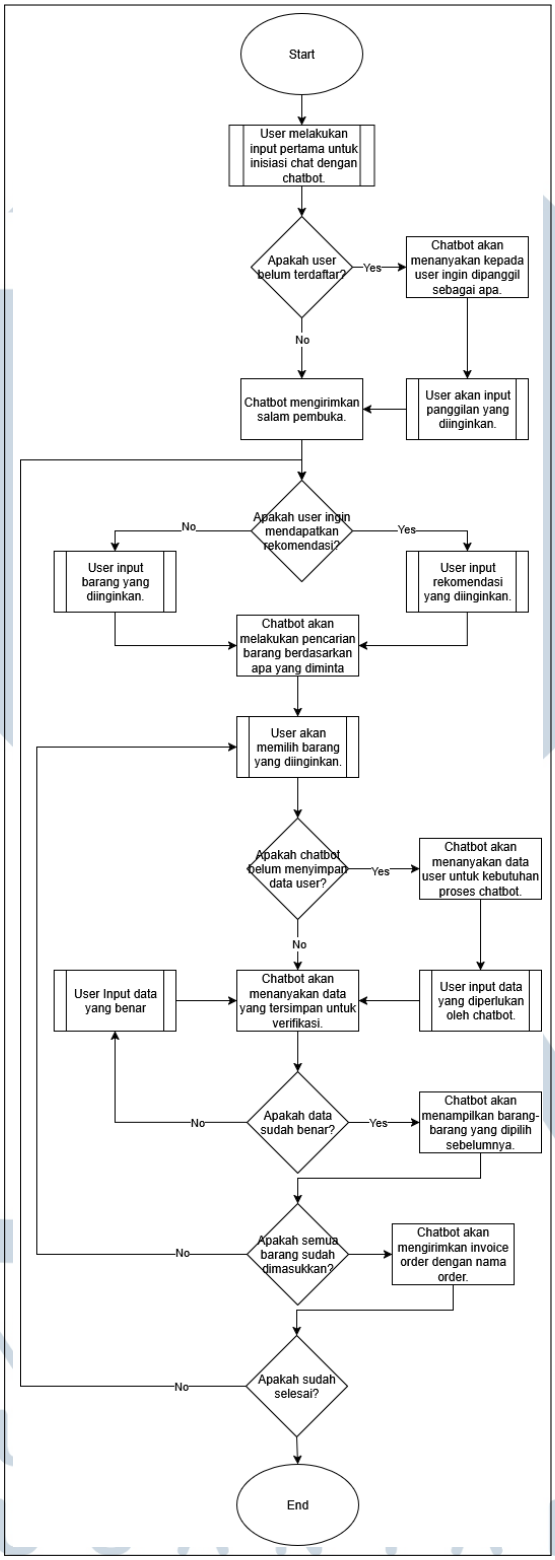
Hal ini menunjukkan bahwa meskipun aplikasi secara visual telah mendukung kenyamanan, diperlukan perbaikan pada sisi panduan alur transaksi dan peningkatan kejelasan sistem interaksi antar pengguna.

Selain itu, perlu dicatat bahwa pengujian aplikasi pada tahap ini hanya dilakukan pada perangkat Android. Pengujian pada perangkat iOS belum dilakukan, sehingga aspek kenyamanan dan tampilan pada sistem operasi tersebut belum dapat dievaluasi secara menyeluruh.

3.3.2 Chatbot Whatsapp

Pengembangan *chatbot* Whatsapp dalam proyek ini difokuskan untuk melakukan penanganan terhadap satu jenis kategori produk terlebih dahulu, sebagai contohnya yakni oli motor, agar alur interaksi pengguna dapat dibangun serta diuji secara terfokus. Alur yang telah dirancang dapat dilihat pada Gambar 3.38, yang mencakupi awal dari inisiasi percakapan, penanganan pertanyaan oleh *chatbot*, pengecekan ketersediaan produk, hingga proses pembuatan tiket atau *invoice*.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.38. Alur chatbot WhatsApp untuk melakukan pemesanan

Sebelum masuk ke tahap implementasi sistem, berikut adalah tahapan-

tahapan perancangan yang telah dilakukan untuk membangun sistem chatbot WhatsApp secara bertahap dan terstruktur.

A Langkah-Langkah Perancangan

Perancangan sistem chatbot WhatsApp dilakukan mengikuti tahapan umum pengembangan perangkat lunak yang terstruktur. Tahapan ini disusun untuk memastikan pengembangan sistem berjalan sesuai kebutuhan bisnis dan teknis. Adapun langkah-langkah yang telah dilalui dalam perancangan chatbot ini mencakup:

1. *Research:* Studi dilakukan terhadap sistem chatbot berbasis WhatsApp yang umum digunakan, serta integrasi WhatsApp Cloud API dengan backend chatbot.
2. *Requirement Analysis:* Ditentukan cakupan fitur awal seperti pencarian oli, pengisian data pelanggan, serta pembuatan invoice berdasarkan kebutuhan klien internal.
3. *Design:* Perancangan dilakukan pada arsitektur backend, rancangan alur percakapan (state flow), serta alur sinkronisasi data dari PostgreSQL ke Milvus.
4. *Implementation:* Implementasi meliputi koneksi chatbot dengan WhatsApp melalui webhook, pembangunan API pengguna, penyusunan logic per state, dan sistem sinkronisasi vektor embedding produk.
5. *Testing:* Pengujian fungsional telah dilakukan secara internal untuk memastikan seluruh transisi antar state dan sinkronisasi data berjalan dengan baik.

B Pembuatan Infrastruktur Backend Chatbot

Infrastruktur *backend* untuk sistem chatbot ini dibangun menggunakan bahasa Go (Golang) menggunakan *framework* Gin sebagai HTTP *web framework* utama. Sistem ini dirancang menggunakan pendekatan modular yang mencakupi konfigurasi aplikasi, koneksi ke basis data, penyusunan API *routing*, serta proses-proses asinkron seperti sinkronisasi data.

Backend ini terhubung dengan *database* PostgreSQL untuk menyimpan informasi pengguna serta detail produk. Koneksi kepada *database* dikelola menggunakan *file* konfigurasi .env. Untuk *vector database*, digunakan Milvus untuk melakukan pencarian semantik produk. Dan yang terakhir, *backend* juga menggunakan Redis sebagai *cache* dan penyimpanan *state* ringan untuk proses validasi atau penyimpanan *fingerprint* data.

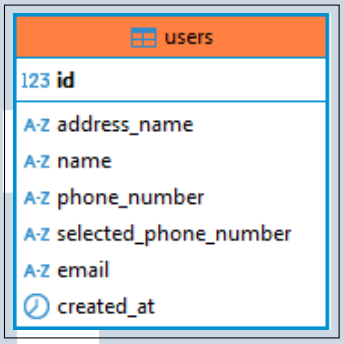
Pemilihan bahasa pemograman sebagai Golang didasari oleh keunggulannya dalam segi performa, efisiensi memori, dan kemudahan dalam membangun layanan web yang ringan dan stabil[3]. *Gin Framework* dipilih dikarenakan sifatnya yang minimalis namun tetap dapat mendukung fitur-fitur modern seperti *routing* yang efisien dan *middleware* yang fleksibel. Untuk penyimpanan data relasional, PostgreSQL dipilih dikarenakan skalabilitasnya yang baik serta dukungan untuk relasi data yang lebih kompleks. Milvus digunakan sebagai *vector database* dikarenakan kemampuannya untuk menangani pencarian semantik yang berbasis *embedding* dengan performa yang tinggi. Terakhir, Redis digunakan karena kemampuannya untuk melakukan sinkronisasi data yang efisien dan cepat.

C Pembuatan API User

Sebagai bagian dari sistem *chatbot*, dibutuhkannya penyimpanan informasi mengenai pengguna-pengguna yang telah berinteraksi dengan *chatbot* ini melalui Whatsapp. Untuk itu, dibuat sebuah tabel bernama *"users"* dalam basis data PostgreSQL yang bertugas untuk menyimpan data-data identitas. Kolom-kolom dalam tabel *"users"* dibagi menjadi kolom-kolom berikut:

- **id** – Sebagai *primary key*, merupakan angka unik yang di-*generate* otomatis untuk setiap pengguna.
- *name* – Nama lengkap pengguna yang dapat diisi melalui alur obrolan.
- *phone_number* – Nomor telepon pengguna yang menjadi identitas utama dan bersifat *unique*.
- *selected_phone_number* – Digunakan apabila pengguna ingin mengatur nomor lain sebagai kontak utama dalam konteks tertentu.
- *email* – Alamat email pengguna, yang dapat dikumpulkan sebagai data tambahan.

- *created_at* – Waktu saat data pengguna pertama kali disimpan, terisi otomatis menggunakan *"current_timestamp"*

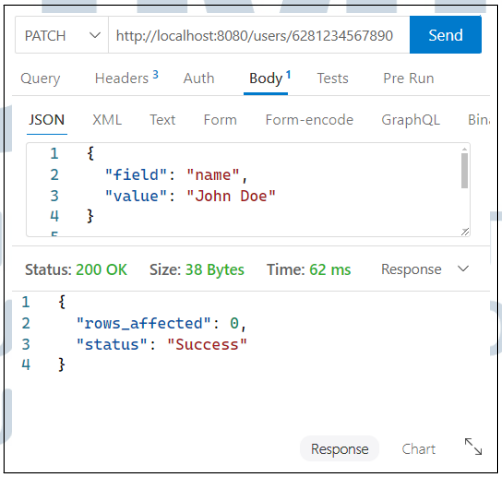


Gambar 3.39. Struktur tabel *"users"* pada basis data PostgreSQL

C.1 API Patch User

API *Patch User* dibangun agar sistem dapat menyimpan atau memperbarui data pengguna secara dinamis. Salah satu *endpoint* utama yang dibuat merupakan "PATCH /users/:phone_number", dimana akan berfungsi sebagai *controller* untuk melakukan operasi *"upsert"*, yang berarti akan melakukan *update* jika data sudah ada, atau *insert* jika data masih belum dimasukkan ke dalam *database*.

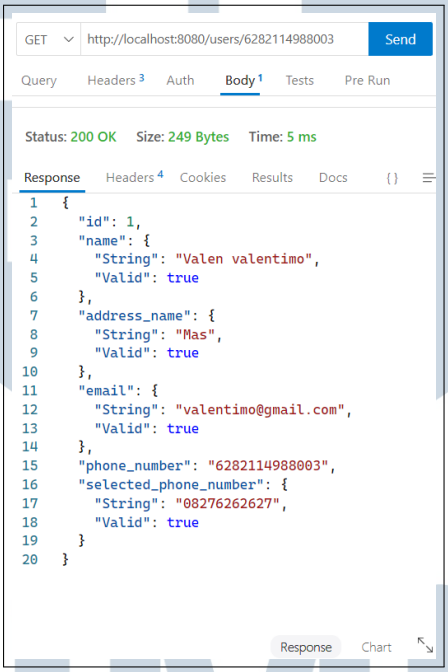
Pendekatan menggunakan *upsert* ini dilakukan agar *chatbot* dapat menyimpan data pengguna secara bertahap sesuai dengan alur percakapan. Gambar dari respons *"success"* dapat dilihat pada Gambar 3.40



Gambar 3.40. Contoh respons sukses API *upsert user*

C.2 API Get User

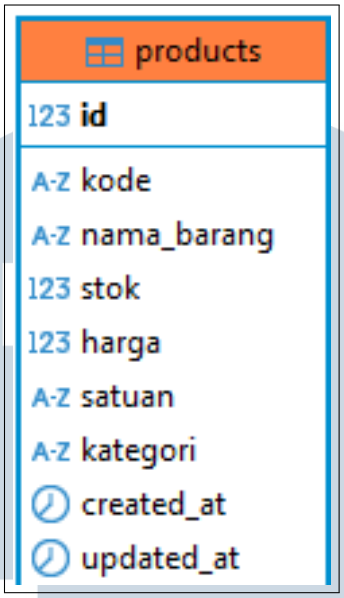
API *Get User*, dibuat untuk memungkinkan *frontend* untuk mengambil seluruh data pengguna berdasarkan nomor telepon tertentu. API ini digunakan oleh *chatbot* untuk menampilkan informasi kepada pengguna apakah data yang diambil sudah benar atau belum, serta juga untuk menghindari pertanyaan berulang bagi pengguna yang lama. Data yang dikembalikan akan berbentuk JSON dan contoh dari respons sukses untuk API *Get User* ini dapat dilihat pada Gambar 3.41



Gambar 3.41. Contoh respons sukses dari API *Get User*

D Pembuatan Sinkronikasi Milvus dengan PostgreSQL

Salah satu fitur yang penting dalam sistem *chatbot* ini adalah kemampuannya untuk melakukan pencarian semantik terhadap produk-produk berdasarkan namanya dan deskripsi. Untuk melakukan hal tersebut, dibutuhkannya sinkronisasi data secara berkala dari *database* relasional, yaitu PostgreSQL, ke *vector database*, yaitu Milvus. Proses ini dilakukan menggunakan penjadwalan untuk melakukan pengecekan secara otomatis. Struktur tabel data dari tabel produk dapat dilihat pada Gambar 3.42

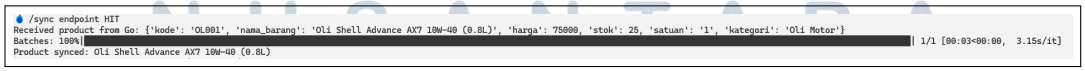


Gambar 3.42. Struktur tabel products di PostgreSQL

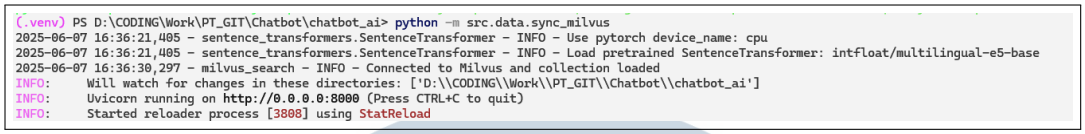
D.1 Proses Sinkronisasi

Proses sinkronisasi untuk saat ini dijalankan setiap 30 menit. Proses ini akan melakukan pengecekan terhadap baris-baris yang berada pada tabel *"products"* jika salah satu dari tabel tersebut mengalami perubahan dalam kurun waktu 31 menit terakhir. Hal ini dikarenakan untuk mendapatkan performa yang tetap cepat walaupun data yang berada pada *database* jauh lebih banyak.

Pada proses ini, digunakannya layanan "FastAPI" untuk menerima data produk, memproses teks menjadi *embedding* vektor, dan akhirnya menyimpan ke dalam koleksi Milvus yang telah disiapkan. Untuk menghindari duplikasi dan proses yang tidak diperlukan, setiap data pada produk yang dimasukkan akan dihitung masing-masing hash dan dibandingkan dengan hash yang sebelumnya tersimpan di Redis. Jika tidak ada perubahan maka proses untuk melakukan penyimpanan ke Milvus akan terlewati. Contoh dari *output* proses ini dapat terlihat pada Gambar 3.43, serta contoh dari proses *startup* Python FastAPI dapat terlihat pada Gambar 3.44.



Gambar 3.43. Contoh log proses FastAPI saat menerima data dan menyinkronkan ke Milvus



Gambar 3.44. Tampilan saat layanan FastAPI dinyalakan dan siap menerima sinkronisasi

E Pembuatan Chatbot WhatsApp

Untuk membangun antarmuka percakapan yang dapat diakses secara langsung oleh pengguna melalui platform WhatsApp, sistem *chatbot* ini dikembangkan menggunakan Python sebagai bahasa pemrograman dengan kerangka kerja Flask. Pemilihan platform WhatsApp digunakan karena platform tersebut merupakan salah satu aplikasi perpesanan yang paling banyak digunakan di Indonesia, sehingga relevan pada kebutuhan pengguna akhir yang tidak terbiasa menggunakan aplikasi tambahan.

Integrasi dengan WhatsApp dilakukan melalui WhatsApp Cloud API yang disediakan oleh Meta. Notifikasi yang dikirimkan oleh WhatsApp kepada *chatbot* ini menggunakan *endpoint* /webhook sebagai jembatan. Bot yang mendapatkan pesan ini akan memproses pesan tersebut dan mengarahkan sesuai dengan *state* percakapan dan *intent* pengguna dalam melakukan percakapan tersebut. Proses ini menggunakan sistem *state machine* yang telah dirancang dengan khusus.

Untuk membangun *chatbot* ini, teknologi Milvus dan Ngrok digunakan. Milvus digunakan sebagai *vector database* agar dapat menyimpan representasi vektor dari produk. Ini akan memperbolehkan sistem untuk melakukan pencarian barang secara semantik, sehingga proses pencarian dapat dicari secara cepat dengan kata kunci yang tidak literal. Teknologi Ngrok juga digunakan pada tahap pengembangan untuk mengekspos server Flask lokal agar dapat diakses secara publik dengan menggunakan internet. Ini penting dilakukan agar notifikasi WhatsApp Cloud API dapat mengakses *chatbot* yang sedang berjalan secara lokal.

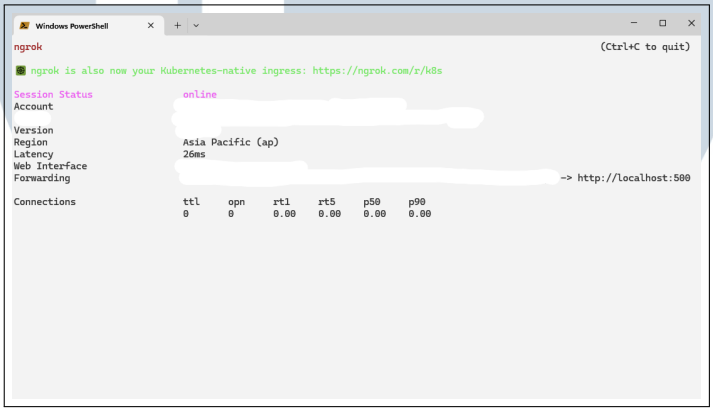
F Pembuatan Koneksi Chatbot dengan WhatsApp Cloud API

Untuk membuat sebuah koneksi antara *chatbot* lokal dengan WhatsApp, sistem ini akan menggunakan layanan *WhatsApp Cloud API* yang disediakan oleh Meta. Koneksi ini yang akan nantinya memungkinkan pengguna untuk berinteraksi dengan *chatbot* menggunakan WhatsApp.

F.1 Konfigurasi Webhook

Untuk melakukan penerimaan antara notifikasi dari WhatsApp dengan sistem *chatbot*, dibutuhkannya *webhook endpoint*. Endpoint ini bertugas untuk melakukan *"GET"* untuk pesan yang sedang masuk, dan *"POST"* untuk mengirimkan pesan kembali.

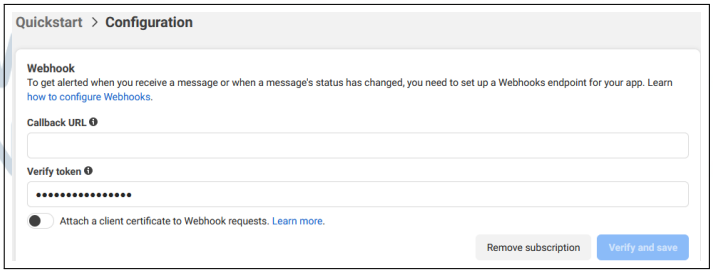
Pada saat melakukan *deploy* pada sistem lokal, *endpoint* ini tidak dapat diakses melalui internet. oleh karena itu, digunakanlah teknologi Ngrok untuk membangun sebuah terowongan dari *localhost* menuju URL publik. Contoh dari Ngrok yang sedang aktif dapat dilihat pada Gambar 3.45



Gambar 3.45. Koneksi lokal Flask dengan URL publik menggunakan Ngrok

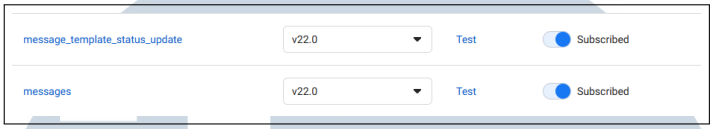
F.2 Setup pada WhatApp Cloud API

Pada platform Meta Developer, pengembang harus melakukan beberapa konfigurasi agar *webhook* yang terbuat dapat berfungsi. Konfigurasi ini adalah dengan menentukan *Callback* URL, yang merupakan URL publik yang dapat diakses oleh WhatsApp, serta *Verify Token*, yang digunakan sebagai *password* untuk dapat mengakses *WhatsApp Cloud API*.



Gambar 3.46. Form pengisian Callback URL dan Verify Token di Meta Developer

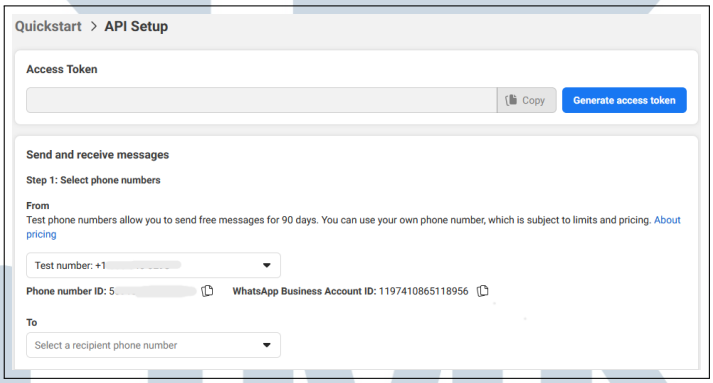
Setelah verifikasi oleh Meta sudah berhasil, *event messages* dan *message_template_status_update* perlu di-*subscribe* agar pesan dapat diterima dan dikirimkan balik.



Gambar 3.47. Webhook event yang perlu diaktifkan untuk mengelola pesan

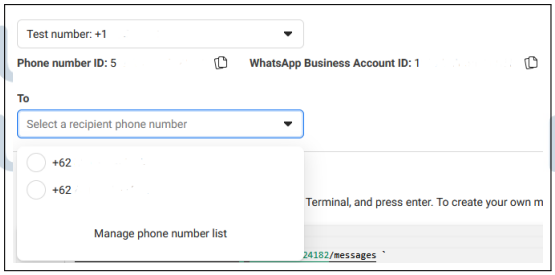
F.3 Setup Token dan Phone Number

Untuk kebutuhan pengembangan, meta juga telah menyediakan nomor WhatsApp uji coba dan *Access Token* yang digunakan untuk melakukan autentikasi setiap kali mengirim pesan melalui API. Informasi ini diperlukan untuk dimasukkan ke dalam file .env dan digunakan dalam kode saat sedang mengirimkan pesan.



Gambar 3.48. Pengambilan Access Token dan pemilihan nomor pengirim

Untuk agar pengembang dapat mengakses nomor uji coba ini, WhatsApp memerlukan untuk melakukan verifikasi dan mendaftarkan nomor yang ingin digunakan pada masa testing.



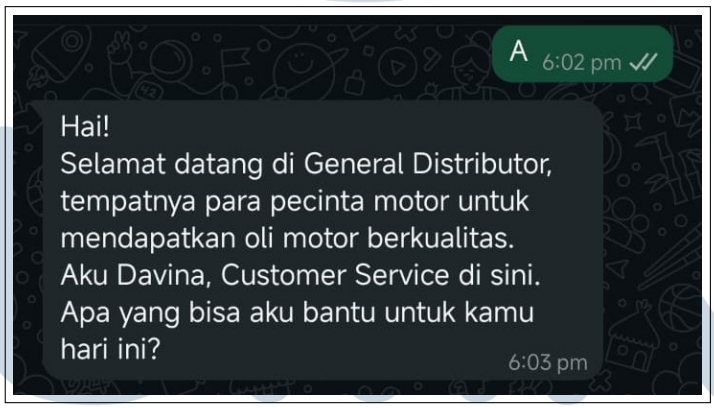
Gambar 3.49. Nomor-nomor penerima uji coba dapat didaftarkan untuk interaksi

G State dalam Alur Percakapan Chatbot WhatsApp

Untuk mengatur percakapan agar lebih terstruktur dan responsif, chatbot WhatsApp ini menggunakan pendekatan *state machine*. Setiap percakapan timbal balik dibagi menjadi beberapa *state* atau keadaan, dimana masing-masing akan menangani satu konteks atau kebutuhan spesifik dari pengguna.

G.1 STATE_INITIAL

State ini merupakan titik awal dari percakapan *chatbot*. State ini akan secara otomatis diakses jika pengguna melakukan inisiasi *chat* terhadap *chatbot*. Sistem kemudian akan mengecek apakah pengguna sudah memiliki sapaan dan apakah pengguna tersebut sudah pernah disambut. Jika belum, maka pengguna akan diarahkan ke *state STATE_ADDRESS_NAME*. Namun jika data sudah lengkap, maka *chatbot* akan langsung mendeteksi *intent* dari pengguna melalui model LLM. *Intent* ini dapat berupa rekomendasi ataupun pemesanan dalam bentuk *single* ataupun *bulk*. Contoh dari percakapan *STATE_INITIAL* dapat dilihat pada Gambar 3.50.

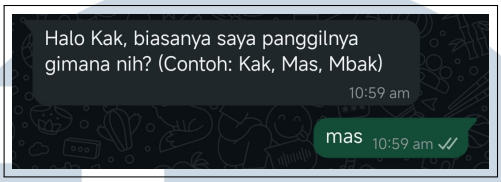


Gambar 3.50. State INITIAL – mengenali intent pertama dari pengguna

G.2 STATE_ADDRESS_NAME

Jika *adress_name* dari pengguna belum tersedia, *chatbot* akan menanyakan bagaimana pengguna ingin dipanggil. Hal ini akan membuat interaksi antara pengguna dan *chatbot* menjadi lebih personal dan humanis. Jawaban dari pengguna nanti akan disimpan ke dalam database melalui *upser_user* dan juga ke *user_state*. Setelah itu, bot akan kembali ke *STATE_INITIAL* untuk menyapa dengan

menggunakan sapaan yang baru diberikan. Contoh dari *STATE_ADDRESS_NAME* ini dapat dilihat pada Gambar 3.51.



Gambar 3.51. *State Address Name* – Bot menanyakan sapaan untuk berinteraksi lebih personal.

G.3 STATE_RECOMMENDATION

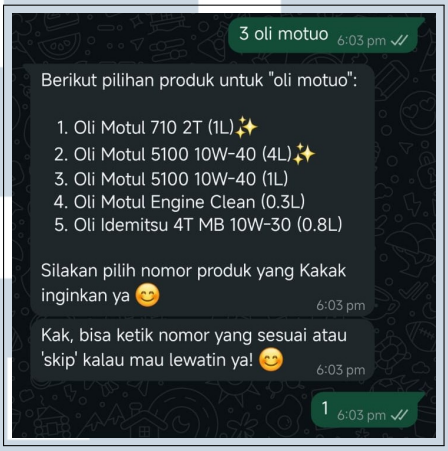
Jika pengguna mengetik rekomendasi atau kalimat sejenisnya, sistem akan berpindah menuju state ini. *Chatbot* kemudian akan mendeteksi *intent* ini dan akan mencari dan menampilkan daftar oli yang direkomendasikan berdasarkan kebutuhan pengguna. User kemudian diperintahkan untuk memilih produk dan juga kuantitasnya. Sistem juga akan melihat ketersediaan barang tersebut sebelum tercatat. Contoh dari *state* ini dapat dilihat pada Gambar 3.52.

Gambar 3.52. State Recommendation – Daftar oli muncul sesuai kategori, misalnya untuk scooter.

G.4 STATE_DIRECT_SEARCH

Jika pengguna langsung mengetikkan nama oli atau jumlah ditambah dengan nama oli seperti ”3 oli motul”, sistem akan langsung mendeteksinya sebagai

pencarian langsung. Bot kemudian akan melakukan pencarian pada *database* Milvus dan menampilkan hasil terdekat dalam bentuk daftar pilihan. Setelah dipilih, pesanan akan ditambahkan ke daftar order sementara. Contoh dari percakapan *state* ini dapat dilihat pada Gambar 3.53.

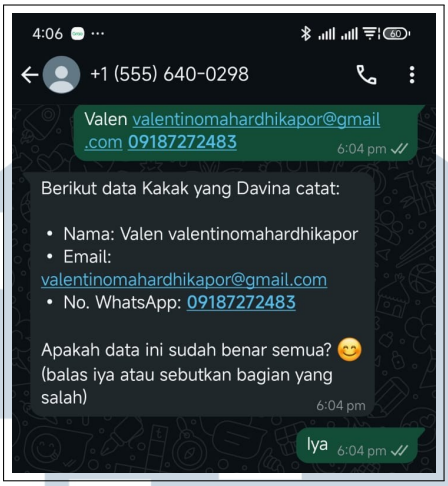


Gambar 3.53. State Direct Search – Menampilkan hasil pencarian berdasarkan permintaan pengguna.

G.5 STATE_PERSONAL_DATA

Setelah produk dipesan, *chatbot* kemudian akan mengarahkan pengguna untuk mengisi data pribadi seperti nama, email, dan nomor yang ingin dituju. Pada tahap ini, *chatbot* pertama akan melakukan pengecekan terhadap kelengkapan data pengguna dari *database* PostgreSQL. Jika belum lengkap, *chatbot* akan meminta pengguna untuk bagian-bagian yang kekurangan. Data ini diperlukan untuk pembuatan *invoice* dan validasi pembelian. Contoh dari percakapan untuk *state* ini dapat dilihat pada Gambar 3.54.

UNIVERSITAS MULTIMEDIA NUSANTARA

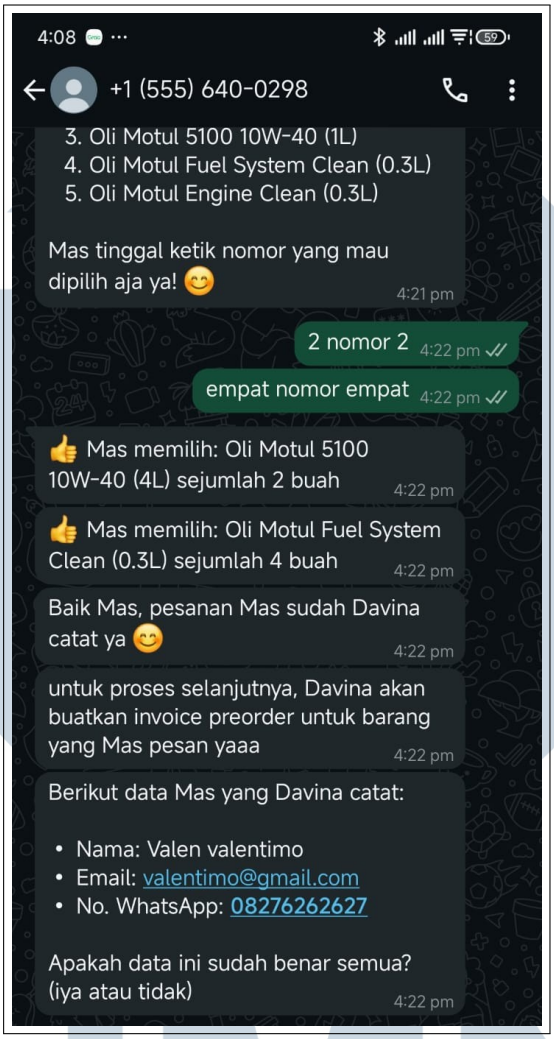


Gambar 3.54. State Personal Data – Bot menanyakan nama, email, dan nomor WhatsApp user.

G.6 STATE_PERSONAL_VERIFY

Setelah pengguna telah melakukan pengisian data, *chatbot* kemudian akan menyusun ulang data tersebut dan menampilkan ringkasan data ke pengguna untuk melakukan konfirmasi. Jika pengguna menjawab iya, maka *flow* akan lanjut ke proses *checkout*. Namun jika ada kesalahan, pengguna dapat menjawab kesalahan tersebut seperti ”email salah”, maka sistem akan mendeteksi bagian mana yang salah dan diperlukannya koreksi. Setelah itu, *chabot* akan mengarahkan pengguna kembali menuju *STATE_PERSONAL_DATA* dengan informasi yang terbaru. Contoh dari percakapan ini dapat dilihat pada Gambar 3.55.

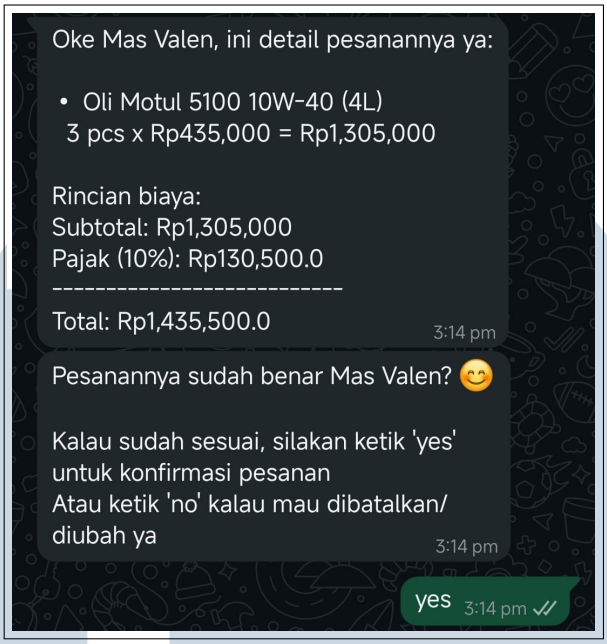
UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.55. State Personal Verify – User diminta konfirmasi apakah data yang dikumpulkan sudah benar.

G.7 STATE_CONFIRMATION

State ini akan menyusun seluruh pesanan yang dipilih oleh pengguna, seperti barang-barang, jumlah, subtotal, pajak, dan total. *Chatbot* kemudian akan menampilkan sebuah ringkasan dalam format *invoice preview*. Lalu pengguna akan diminta apakah pesanan tersebut sudah benar atau belum. Jika iya, maka akan memicu penyimpanan order serta pengiriman invoice dalam bentuk PDF. Jika tidak, pengguna akan diarahkan menuju *state STATE_MODIFY_ORDER*. Contoh dari percakapan ini dapat dilihat pada Gambar 3.56.

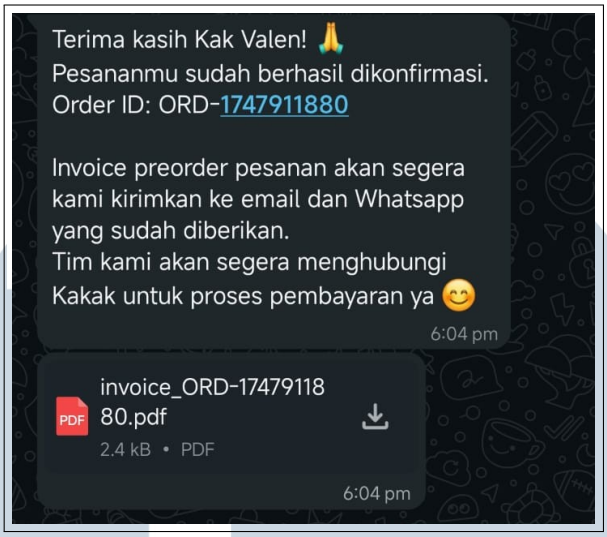


Gambar 3.56. State Konfirmasi – Ringkasan detail pesanan tampil untuk dikonfirmasi.

G.8 STATE_DONE

Jika pengguna telah melakukan konfirmasi, order yang tersimpan akan disimpan ke sistem dan *invoice* PDF akan dikirimkan melalui WhatsApp. Chatbot kemudian akan memasuki *STATE_DONE*, dan hanya akan merespons *prompt* ringan. Jika *user* ingin menambahkan pesanan maka *chatbot* akan mengarahkan *user* kembali menuju *STATE_INITIAL*. Contoh dari percakapan ini dapat dilihat pada Gambar 3.57.

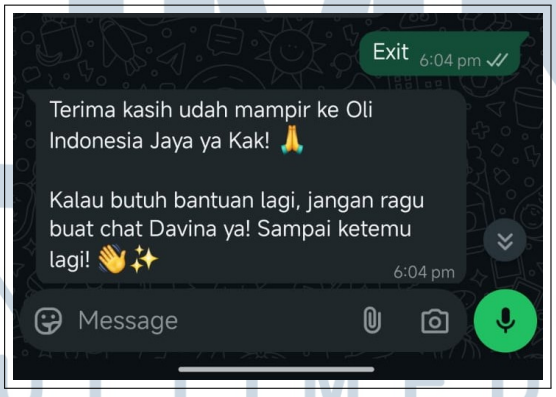
UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.57. State Done – Invoice berhasil dikirim, proses pemesanan selesai.

G.9 STATE_ZERO

State ini akan diaktifkan jika *user* telah selesai dari semua proses dan mengetik *"exit"*. *Chatbot* kemudian akan mengucapkan terima kasih dan akan mengakhiri sesi ini. State ini ada untuk menjaga agar *state* tetap aman dan tidak akan *carry over* dari sesi sebelumnya. Sesi kemudian akan melakukan pengulangan dari *STATE_INITIAL* jika user mengirimkan pesan berikutnya. Contoh dari *state* ini dapat dilihat pada Gambar 3.58.



Gambar 3.58. State Zero – User keluar dari sesi dan chatbot mereset state ke awal.

H Testing

H.1 Functional Testing

Functional testing juga dilakukan terhadap sistem chatbot WhatsApp yang dikembangkan selama masa magang. Pengujian ini difokuskan pada validasi transisi antar state dalam alur percakapan, serta interaksi backend dengan database PostgreSQL dan vector store Milvus. Seluruh *state* sudah berhasil dijalankan oleh internal dan juga sudah berfungsi dengan baik.

Fitur tambahan seperti sinkronisasi data produk dan riwayat transaksi antara PostgreSQL dan Milvus juga telah berhasil diuji secara fungsional. Sistem mampu menyimpan hasil pencarian dan interaksi pengguna secara efisien.

Namun, pengujian ini masih terbatas pada tim pengembang internal dan belum dilakukan secara menyeluruh pada pengguna eksternal. Oleh karena itu, proses *usability* testing belum dapat dilaporkan untuk *chatbot*, dan direncanakan akan dilakukan pada tahap implementasi lanjutan.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala yang Ditemukan

Beberapa tantangan utama yang dihadapi selama masa melakukan magang serta pengembangan adalah sebagai berikut:

1. Dalam proyek barter Tukerin dan *chatbot* WhatsApp, *backend* menggunakan bahasa pemrograman Go belum pernah dipelajari sebelumnya. Hal ini menimbulkan kebingungan saat harus membaca, memahami, atau mengintegrasikan logika *backend* yang nantinya akan dibuat.

2. Dalam melakukan proyek Tukerin, waktu pengerjaan sangat terbatas sehingga dalam dalam melakukan pekerjaan akan selalu dikejar waktu.

3. Karena waktu yang terbatas, *source code* yang telah dibuat cenderung berantakan. Banyak fungsi dan nama variabel yang saling bertabrakan dan tidak terpisah secara modular.

3.4.2 Solusi yang Digunakan

Untuk mengatasi kendala-kendala yang diatas, beberapa pendekatan berikut dilakukan:

1. Kendala dalam bahasa pemograman baru seperti Go diatasi dengan menggunakan waktu dalam melakukan eksplorasi untuk mempelajari dan langsung mengimplementasikan potongan kode yang ada hingga mempercepat pemahaman.
2. Untuk mempercepat pengembangan demo aplikasi Tukerin, beberapa bagian *frontend*, seperti *chat*, akan dilakukan *hardcode* tanpa bergantungan dengan *backend* API.
3. Setelah fitur utama selesai dan waktu lebih longgar, dilakukannya pembersihan kode saat adanya waktu luang, sehingga dapat didokumentasikan lebih mudah.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA