

HEYBUDDY:

SHOP EASY, SHOP SMART

One cart at a time



Rania Sasi Kirana
18221168
II3160 - Integrated System Technology

A. DESKRIPSI SINGKAT LAYANAN

Berikut merupakan tautan layanan hasil implementasi *website application* dan *API documentation* dan *source code*.

Github backend : <https://github.com/raniakiranaa/Tubes-TST-API.git>

Documentation API : <https://smartcartchatbot.azurewebsites.net/docs>

Github frontend : <https://github.com/raniakiranaa/Tubes-TST-API-Frontend.git>

Website : <https://tubes-tst-api-frontend.vercel.app>

Layanan *smart cart* adalah layanan yang terintegrasi dalam domain retail grocery. Layanan ini dirancang untuk meningkatkan pengalaman belanja pembeli dengan menyediakan fitur-fitur yang dapat membantu mereka dalam pencarian produk hingga proses pembayaran, sehingga proses berbelanja menjadi lebih efisien dan nyaman. Fungsionalitas utama *smart cart* adalah sebagai *chatbot* yang memberikan pengguna informasi dan rekomendasi produk serta *cart management* dalam bentuk *streamlined transaction*. Implementasi yang dilakukan masih relatif sederhana, namun terdapat banyak peluang pengembangan yang luas, seperti pemanfaatan teknologi RFID, sensor, dan kecerdasan buatan (AI).

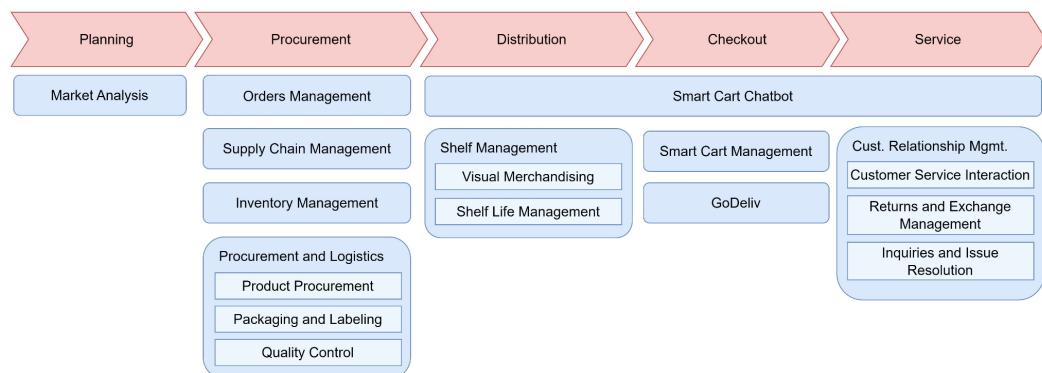
Chatbot yang merupakan *core* dari layanan *smart cart*, dapat membantu pembeli dalam mencari produk yang diinginkan, melihat ketersediaan produk, melihat daftar produk dan letak produk, serta mendapatkan rekomendasi merek yang digemari orang berdasarkan data transaksi jumlah pembelian merek produk. Selain itu, kapabilitas dari *smart cart* berupa *cart management* yang memungkinkan pengguna dapat di-*assign* ke keranjang dan menambah produk langsung ke keranjang. Ketika pengguna melakukan transaksi, data dari keranjang akan dipindahkan ke data transaksi dan data di keranjang akan dihapus. Begitu juga jika pengguna tidak menyelesaikan transaksi, pengguna dapat membatalkan transaksi dengan cara menghapus data keranjang yang akan menghapus seluruh data produk yang ada di keranjang. Terdapat juga autentikasi *user* dimana beberapa fungsi dibatasi hanya dapat dilakukan oleh pengguna yang memiliki *role* tertentu. *Role* dibagi menjadi 2, yaitu *admin* dan *customer*. Hal ini dapat terjadi dengan

cara memanfaatkan basis data yang berisi data *user*, produk, keranjang (*cart*), dan transaksi (*transaction*) beserta detailnya.

Untuk meningkatkan fungsionalitas *smart cart*, dilakukan integrasi dengan layanan Godeliv, yang merupakan layanan rekomendasi menu berdasarkan kalori, umur, dan jenis kelamin. Dengan layanan ini, rekomendasi yang diberikan akan semakin luas dan sesuai dengan preferensi masing-masing pengguna sehingga pengalaman belanja dapat menjadi lebih *personal*.

Secara garis besar, fungsionalitas utama layanan GoDeliv memberikan rekomendasi menu sehat dihitung dari kalori berdasarkan perhitungan berat dan tinggi pengguna. Selain itu, rekomendasi menu juga dapat diberikan berdasarkan umur serta jenis kelamin pengguna. Fungsionalitas akan diintegrasikan dengan *smart cart* sehingga memungkinkan pengguna mendapat rekomendasi menu sehat dan mencari produk yang dibutuhkan untuk menyusun menu tersebut di *retail grocery*.

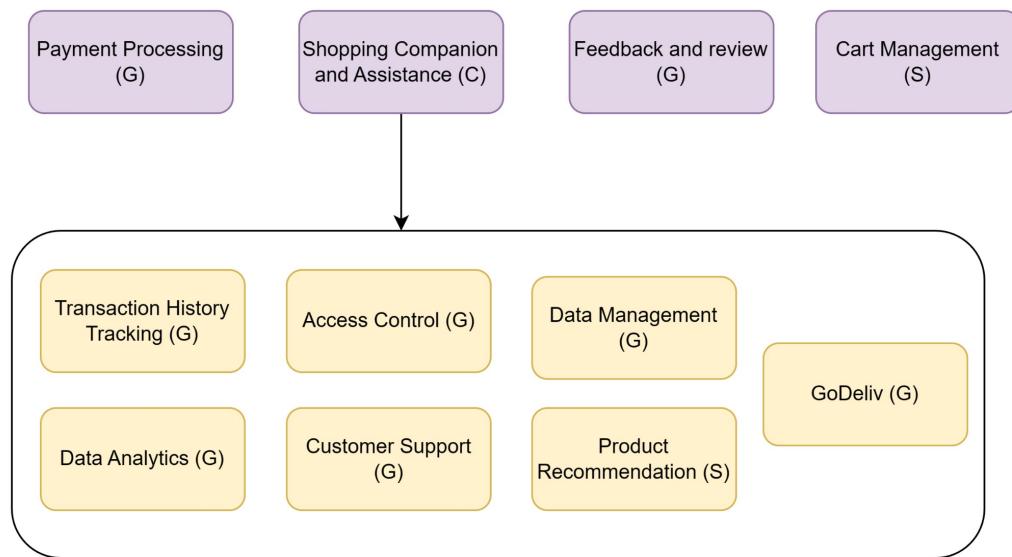
a. Business Capability



Secara garis besar, value stream *retail grocery* mencakup proses perencanaan, pengadaan produk, distribusi kepada pelanggan, *checkout* atau transaksi oleh pelanggan, dan *service*, yang di setiap tahap *value stream* memiliki *business capability*-nya masing-masing. *Business capability* yang akan diangkat pada layanan ini adalah *smart cart chatbot* yang merupakan *core service* dari *food retail* ini. *Smart cart chatbot* menjadi *core service* karena memiliki peran penting dan memberikan nilai tambah yang signifikan pada proses belanja di *food retail*. Selain itu, *smart cart* tidak dimiliki oleh layanan lain sehingga meningkatkan daya kompetitif perusahaan. *Smart cart* merupakan *business capability* dalam

proses distribusi kepada pelanggan dalam hal pemberian rekomendasi melalui *chatbot*, proses *checkout* atau *streamlined transaction* yang memungkinkan transaksi yang dilakukan pengguna lebih efisien, dan proses *service* yang juga diimplementasikan melalui *smart cart*.

b. Core Subdomain dan Bounded Context



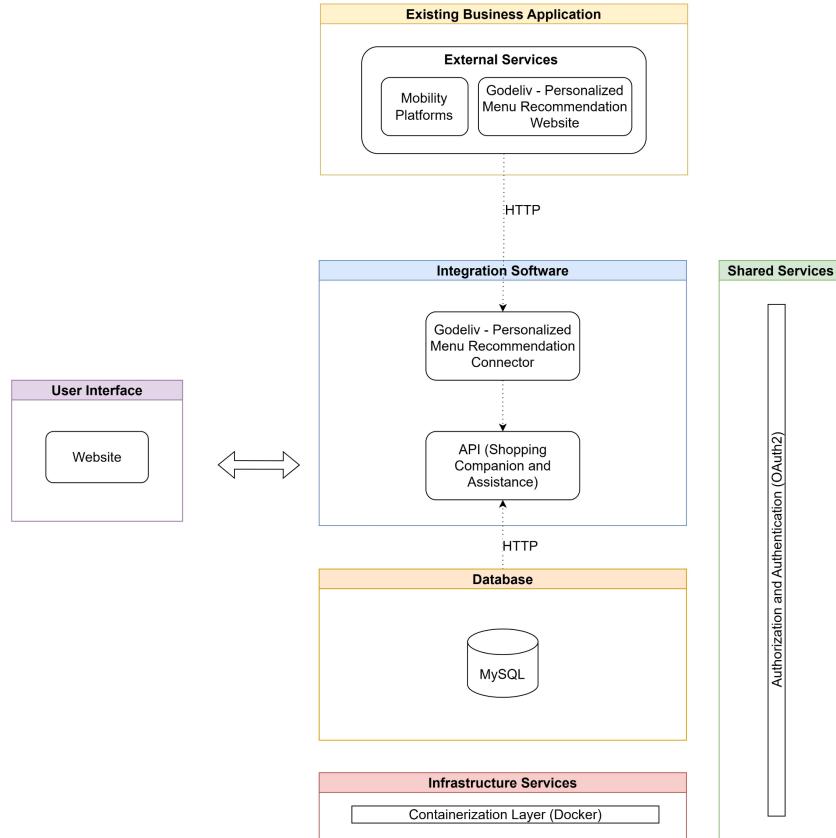
Dekomposisi *subdomain* dari *smart cart* terbagi menjadi 3 subdomain, yaitu *shopping companion and assistance*, *payment processing*, dan *feedback and review*. Layanan yang merupakan *core* adalah *shopping companion and assistance*. Layanan ini terdiri dari beberapa *bounded context* yang mewakili area fungsional yang berbeda:

- *Access Control*: pembatasan akses fungsionalitas sesuai hak akses pengguna menggunakan OAuth (*Open Authorization*). Selain itu, layanan ini juga menggunakan JWT (JSON Web Token) yang menjadi cara untuk mengirimkan informasi antar pihak secara aman dalam bentuk JSON.
- *Data Management*: manajemen data pengguna, produk, dan transaksi di basis data. Aplikasi ini memanfaatkan DBMS relasional, yaitu MySQL yang disediakan oleh Microsoft Azure. Pemilihan penggunaan DBMS relasional dibandingkan dengan NoSQL adalah dengan

mempertimbangkan representasi hubungan antar data yang dapat digambarkan dengan lebih jelas dan terstruktur.

- *Product recommendation*: memberikan rekomendasi produk/*brand* favorit berdasarkan data transaksi historis.
- *Transaction History Tracking*: manajemen data transaksi historis yang diatur di basis data MySQL.
- *Data Analytics*: menganalisis data transaksi.
- *Customer Support*: memberikan bantuan informasi untuk pengguna.
- GoDeliv: Penyedia layanan rekomendasi menu berdasarkan target kalori, umur, dan jenis kelamin pengguna. Dengan mengintegrasikan API layanan ini, *smart cart chatbot* dapat menawarkan pengalaman belanja yang lebih *personal* sesuai dengan masing-masing pengguna.

B. ARSITEKTUR SISTEM



Layanan yang akan diintegrasikan (*external services*) adalah GoDeliv - *Personalized Menu Recommendation*. Integrasi hanya akan dilakukan dengan cara *call endpoint* menggunakan protokol HTTP. *Shopping companion and assistance* juga menggunakan *cloud database Azure MySQL* dan terhubung menggunakan protokol HTTP. *Shared services* yang digunakan adalah OAuth untuk *user authentication* dan *authorization*. *User interface* berupa *website application* dan seluruh layanan akan dienkapsulasi dalam Docker sebagai *containerization layer*. Berikut merupakan beberapa penjelasan lebih lanjut mengenai arsitektur sistem.

a. Website Application

Smart cart dapat diakses oleh pengguna melalui *website application* dengan lingkungan pengembangan sebagai berikut.

- *Operating System* : Windows OS, Mac OS

- *DBMS* : MySQL
- *Development Tools* : React, Next js
- *Programming Language* : JavaScript
- *Filing System* : Github
- *Deployment* : Vercel

b. Docker Containerization

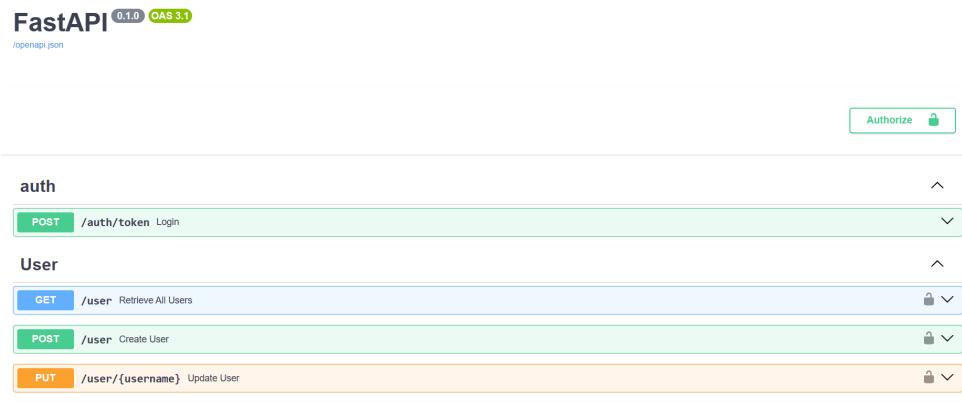
Docker adalah layanan yang menyediakan kemampuan untuk mengenkapsulasi dan menjalankan sebuah aplikasi dalam sebuah lingkungan terisolasi yang disebut *container*. Manfaat penggunaannya yaitu memungkinkan aplikasi untuk dijalankan di banyak *container* di waktu yang bersamaan pada *host* tertentu. Deploy Docker menggunakan layanan *container* Microsoft Azure. Berikut merupakan langkah-langkah yang dijalankan dalam melakukan *docker containerization*.

1. Membuat Dockerfile di direktori *source code*.

```
↳ Dockerfile
1 FROM python:3
2 ADD smartcart.py .
3 COPY . /Tubes-TST-API
4 WORKDIR /Tubes-TST-API
5 RUN pip install fastapi uvicorn mysql.connector.python
6 CMD ["uvicorn", "smartcart:app", "--host=0.0.0.0", "--port=80"]
```

2. Masukkan *command* “*docker image build --tag smartcart .*” di cmd.
3. Masuk ke portal Microsoft Azure, pilih menu bagian Container, *create Container Registry*. Masukkan data yang sesuai dengan saat men-*deploy database*. Setelah selesai di-*deploy*, pilih menu Access Keys dan pilih admin mode.
4. Masukkan *command* “*docker login {nama server} -u {username} -p {password}*” di cmd menggunakan data di menu Access Keys.
5. Masukkan *command* “*docker build -t {nama server}/{nama image docker}:{nama tag}* .”
6. Masukkan *command* “*docker push {nama server}/{nama image docker}:{nama tag}*”

7. *Create Container Instance* di menu bagian Container Azure. Masukkan data yang sesuai. Pilih opsi Azure Container Registry pada menu Image Source karena telah membuat Container Registry di langkah ke-3. Setelah selesai di-deploy, ambil *address* yang digunakan untuk mengakses Swagger UI dari FastAPI yang telah dibuat.



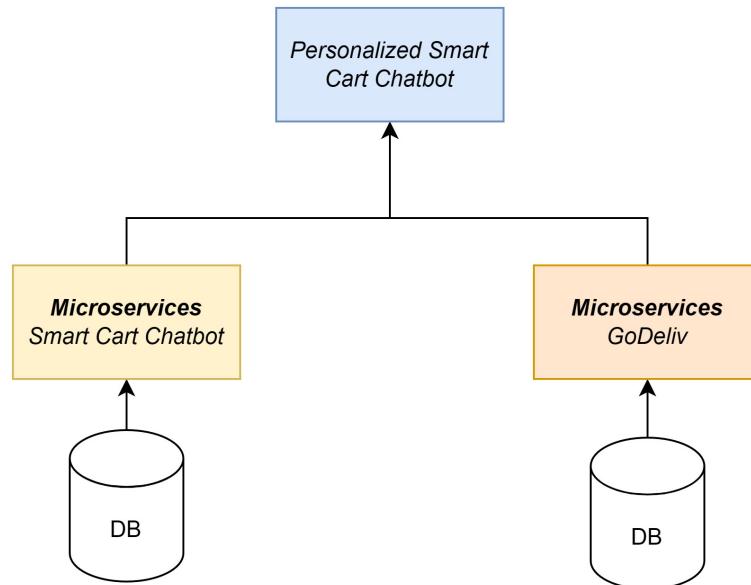
c. *Vercel Website Deployment*

Vercel merupakan *platform hosting* yang digunakan untuk membangun *website*. Alasan penggunaan Vercel adalah karena didukung banyak bahasa pemrograman dan *framework*, termasuk JavaScript, penggunaan yang mudah, serta integrasi dengan *platform* lain seperti Github yang baik sehingga memudahkan untuk melakukan *deployment website* dari *repository* Github. Berikut merupakan langkah melakukan *deploy*.

1. Mendaftarkan diri dan menghubungkan dengan akun Github sehingga memiliki akses ke *repository* yang ada di Github.
2. Melakukan *import repository* dari daftar *repository* yang sudah terintegrasi dengan Github.
3. Mengecek konfigurasi dan mengubah sesuai yang dibutuhkan.
4. Menekan tombol *deploy* dan menunggu proses *deployment* selesai.
5. Mengambil link *url website* dari *dashboard*.

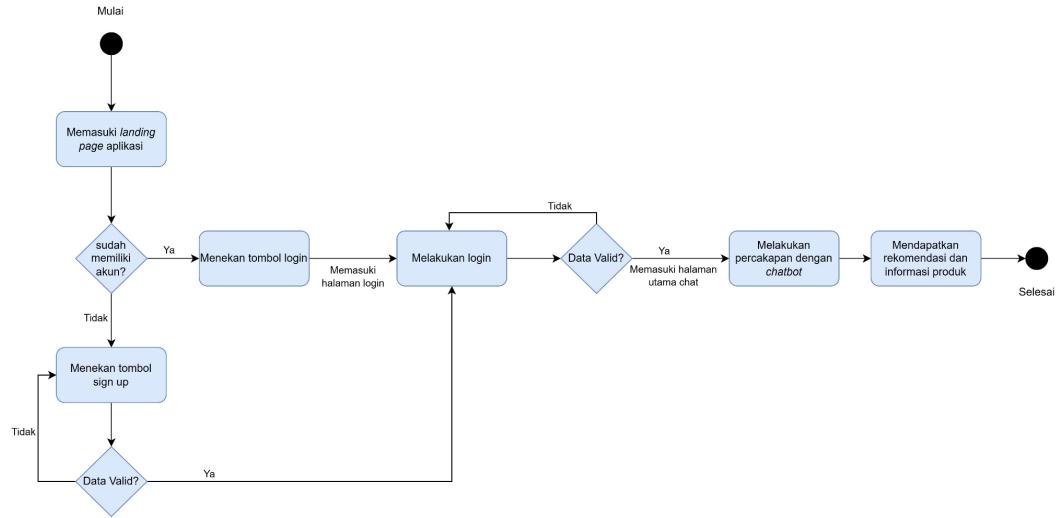
C. DESKRIPSI TEKNIS LAYANAN

Layanan ini menerapkan arsitektur mikroservis sehingga dapat diintegrasikan dengan layanan *microservices* lainnya. Layanan *smart cart chatbot* yang memberikan rekomendasi dan informasi produk berdasarkan data historis diintegrasikan dengan layanan GoDeliv yang dapat memberikan rekomendasi menu berdasarkan target kalori, umur, dan jenis kelamin pengguna. Berikut merupakan gambaran konsep integrasi *microservices* yang telah dilakukan.

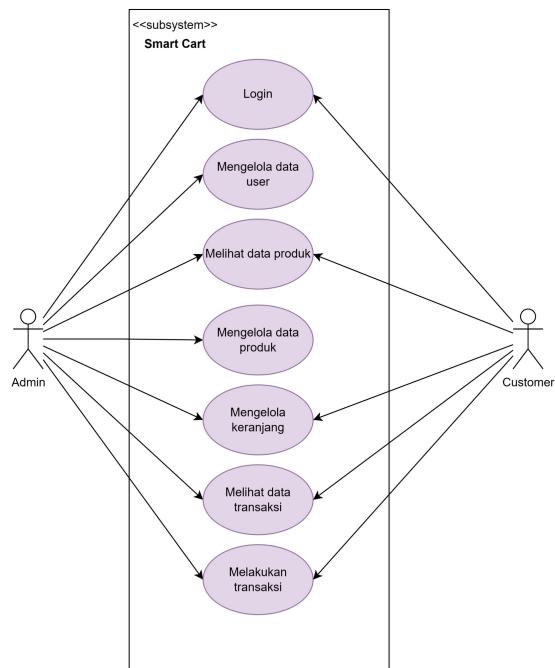


a. Smart Cart Chatbot

Chatbot yang merupakan *core* dari layanan *smart cart*, dapat membantu pembeli dalam mencari produk yang diinginkan, melihat ketersediaan produk, melihat daftar produk dan letak produk, serta mendapatkan rekomendasi merek yang digemari orang berdasarkan data transaksi jumlah pembelian merek produk. Berikut merupakan diagram alur dari aplikasi ini.



Pengguna dibagi menjadi 2, yaitu admin dan *customer* yang memiliki hak akses yang berbeda terhadap aplikasi *smart cart*. Admin dapat *login*, mengelola data user, melihat data produk, mengelola data produk, mengelola keranjang, melihat data transaksi, dan melakukan transaksi. *Customer* dapat *login*, melihat data produk, mengelola keranjang, melihat data transaksi, dan melakukan transaksi. Berikut merupakan diagram *use case smart cart* untuk menggambarkan interaksi antara pengguna dengan aplikasi.

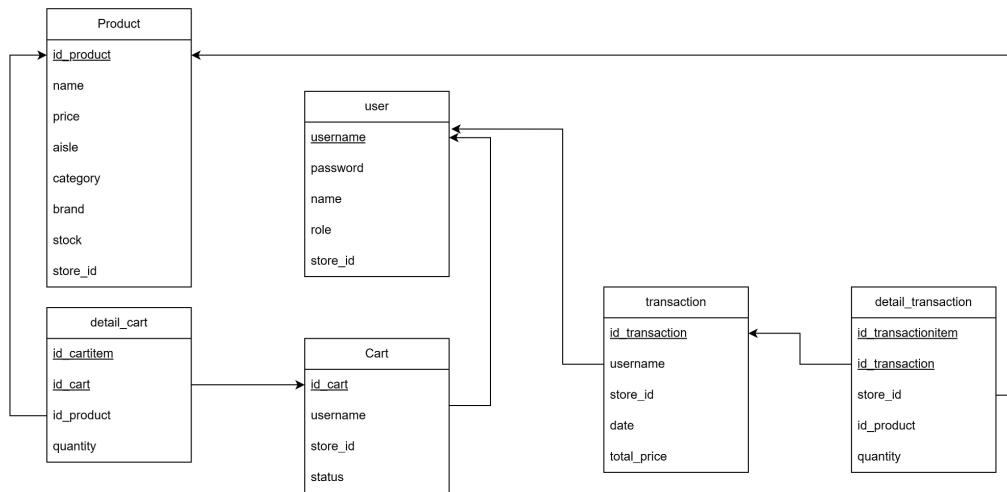


Berdasarkan struktur tersebut, relasi *product* berisi atribut id produk, nama produk, harga, nama lorong, nama kategori, merk, dan stok produk yang tersedia. Relasi *user* berisi *username*, *password*, dan nama pengguna. *Password* yang disimpan merupakan *password* yang telah dienkripsi sehingga data pengguna terjaga. *Username* dan *password* ini yang kemudian akan digunakan untuk mengautentikasi hak akses dan data pengguna. Relasi *cart* memiliki atribut id keranjang, *username* pengguna yang sedang menggunakan keranjang, dan statusnya. Status keranjang dapat berupa ‘*true*’ atau ‘*false*.’ Jika status bernilai ‘*true*’, maka terdapat pengguna yang sedang menggunakan keranjang tersebut. *Username* pengguna juga akan disimpan di *database*. Jika status bernilai ‘*false*’, maka keranjang bersifat *inactive* atau tidak ada pengguna yang sedang menggunakan keranjang tersebut dan atribut *username* akan bernilai *Null*. Barang yang berada di keranjang disimpan di relasi *detail_cart*, yang berisi atribut id cartitem, id keranjang, id produk, dan kuantitas. Kemudian, pada relasi *transaction*, tersimpan atribut id transaksi, *username* pengguna yang melakukan transaksi, tanggal dan waktu transaksi, serta total harga transaksi. Detail transaksi berada di relasi *detail_transaction*, yang berisi id transactionitem, id_transaksi, id produk, dan kuantitas. Atribut *store_id* digunakan untuk mengidentifikasi darimana data toko berasal, mempertimbangkan layanan yang akan diintegrasikan dengan layanan lain untuk memberikan pemisah antar data toko.

Ketika pengguna melakukan register, *role* akan otomatis diatur menjadi “*customer*.” Pengguna yang dapat mengubah *role* tersebut hanyalah “*admin*” yang memiliki hak akses untuk mengelola data *user*. Ketika melakukan *login*, halaman yang muncul diatur berdasarkan *role* dari pengguna dan hak akses yang dimiliki untuk membuka halaman-halaman tertentu.

Aplikasi ini memanfaatkan DBMS relasional, yaitu MySQL yang disediakan oleh Azure. Pemilihan penggunaan DBMS relasional dibandingkan dengan NoSQL adalah dengan mempertimbangkan representasi hubungan antar data yang dapat digambarkan dengan lebih jelas dan terstruktur. Selain itu, SQL

menyediakan antarmuka yang bagus untuk mengambil dan memanipulasi data di *database* dengan mudah. Berikut merupakan struktur basis data *smartcart*.



Selain *constraint primary key* dan *foreign key* yang telah digambarkan pada diagram pada bagian struktur basis data, beberapa fungsi *trigger* juga diterapkan di basis data. Id produk pada relasi *product* dan id transaksi pada relasi *transaction* ditetapkan sebagai atribut *auto increment* yang berarti akan menambah sendiri ketika memasukkan data baru. Selain itu, id cart item pada relasi *detail_cart* yang menggambarkan id barang yang berada di suatu cart serta id transaction item pada relasi *detail_transaction* yang menggambarkan id barang yang tercatat di suatu transaksi juga diterapkan trigger dimana ketika suatu barang ditambahkan di suatu keranjang dan/atau transaksi, id akan bertambah sendiri.

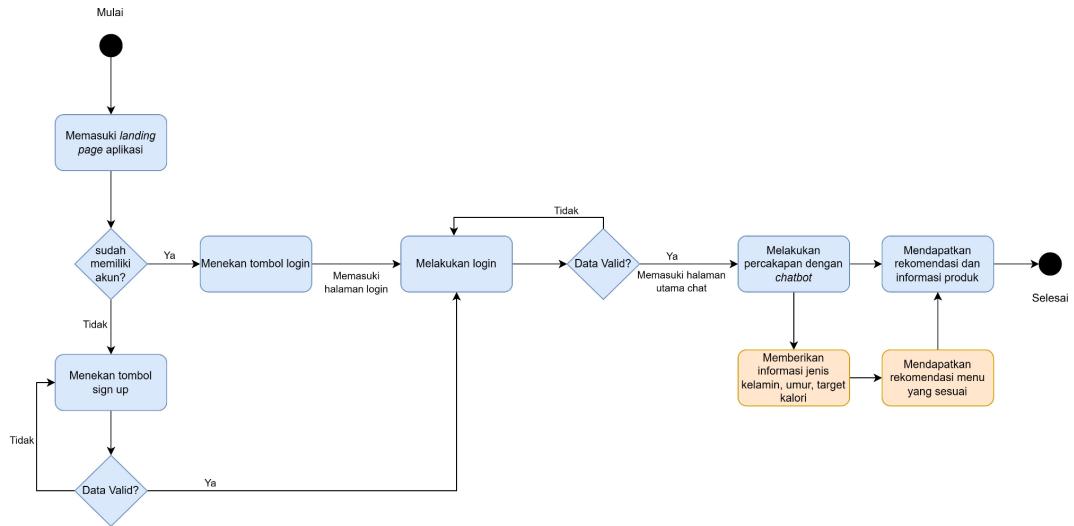
Aplikasi dibuat menggunakan *framework web* FastAPI dan *server web* HTTP Uvicorn yang digunakan untuk menjalankan aplikasi FastAPI. Struktur aplikasi menggunakan struktur *model-view-controller* (MVC), dalam konteks aplikasi ini yaitu disebut *model*, *routes*, dan *database*. Model terdiri dari *cart*, *product*, *token*, *transaction*, dan *user*. Model ini mencakup definisi dari entitas serta contoh skema JSON nya. *Routes* berisi *cart*, *product*, *transaction*, dan *user*. *Routes* mengelola permintaan dari pengguna dan menentukan bagaimana permintaan tersebut harus ditangani termasuk segala kondisionalnya. Ini mencakup pemetaan URL atau *routing* ke fungsi atau metode yang dijalankan. *Database* merupakan modul yang digunakan untuk mengelola koneksi ke

database, yang dalam konteks aplikasi ini menggunakan MySQL Azure dan koneksi dilakukan melalui *extension mysql-connector-python*.

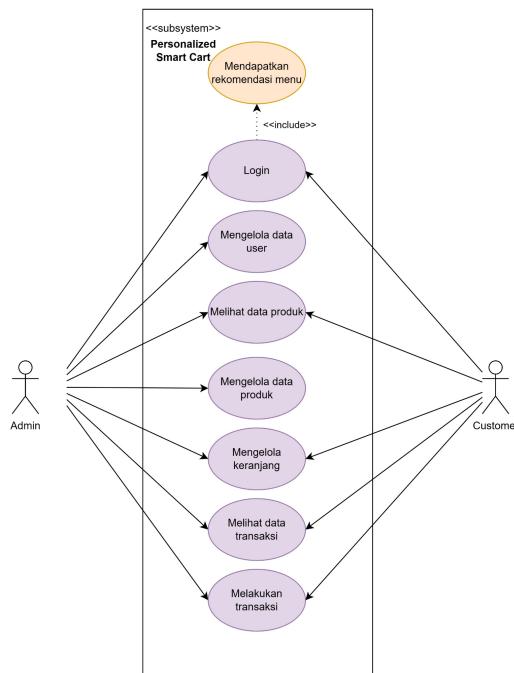
Terdapat juga *services* yang digunakan dalam aplikasi ini yaitu *user authentication* menggunakan OAuth2 dari FastAPI dan JWT (JSON Web Token). *Password* pengguna yang masuk ke *database* juga di enkripsi menggunakan *library* passlib untuk meningkatkan keamanan data pengguna. Saat pengguna mencoba mengakses fungsi yang memerlukan otentikasi, aplikasi akan meminta pengguna untuk melakukan login. Login dilakukan melalui layanan OAuth2 dimana pengguna memberikan kredensial seperti *username* dan *password*. Layanan OAuth2 kemudian akan melakukan verifikasi *username* dan *password* apakah sesuai dengan data di basis data. Setelah kredensial diterima, layanan OAuth2 menghasilkan token akses dalam format JWT (JSON Web Token) yang berisi *username* pengguna dan juga lama waktu sesi login *expired*. Ketika sesi login *expired*, maka pengguna harus melakukan login ulang aplikasi. Aplikasi mengirim token akses dan mengizinkan akses jika token valid. Aplikasi memeriksa dan mendekode token akses JWT untuk mendapatkan informasi pengguna. Token JWT ditandatangani oleh layanan OAuth2. Saat melakukan verifikasi token, penerima menggunakan *secret key* atau kunci rahasia dan algoritma yang hanya dikenal oleh penyedia layanan OAuth2.

b. *Personalized Smart Cart Chatbot*

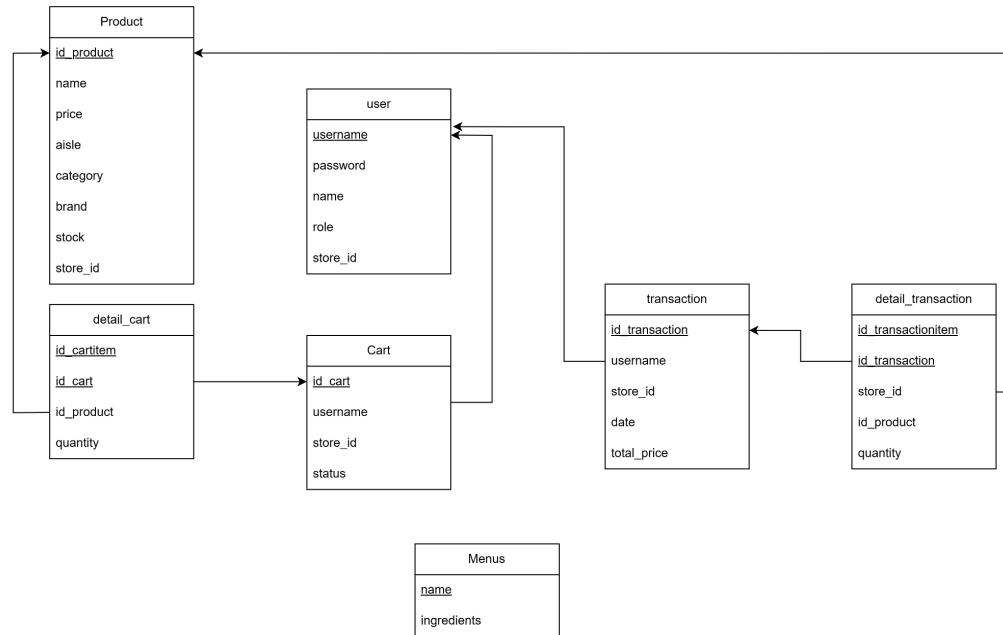
Setelah dilakukan integrasi dengan GoDeliv, layanan bertambah dalam fungsionalitas *chatbotnya*. Diagram alur tidak jauh berbeda karena layanan ditambahkan di dalam *chatbot* sehingga perbedaannya hanyalah di bagian *backend* dan tidak berpengaruh besar dalam interaksi dengan pengguna. Berikut merupakan diagram alur setelah dilakukan integrasi.



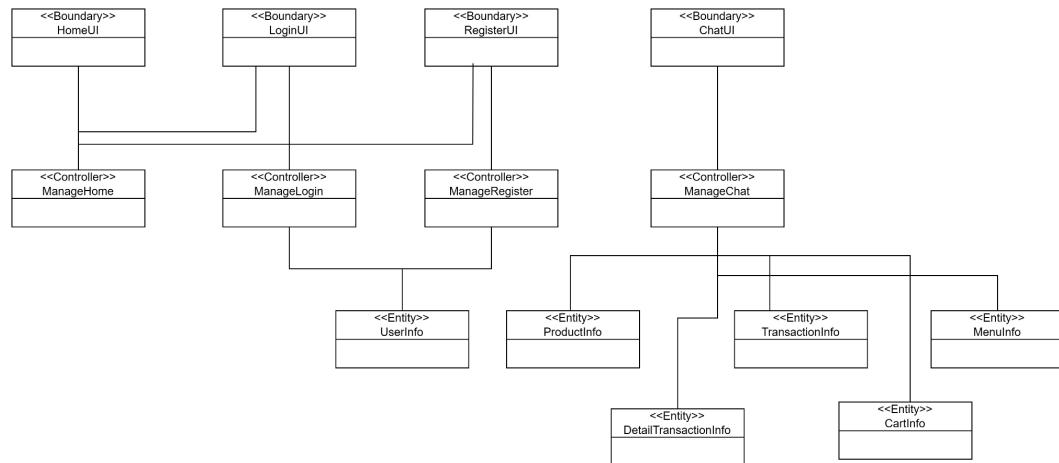
GoDeliv memerlukan *authorization* dari pengguna yang ingin menggunakan layanannya. Maka dari itu, dalam proses integrasi, *smart cart* mendaftarkan satu pengguna yang bertindak sebagai admin di layanan GoDeliv sehingga *customer* didaftarkan ke GoDeliv melalui otentikasi admin *smart cart*. Hal ini dilakukan dengan menggunakan OAuth *bearer token* JWT. Setiap ada pengguna yang akan mengakses rekomendasi dari GoDeliv, otentikasi yang digunakan untuk mengakses menggunakan akses token dari admin *smart cart*. Berikut merupakan *use case personalized smart cart chabot*.



Pada struktur basis data, dilakukan sedikit perubahan, yaitu penambahan tabel menu yang berisi nama menu dan daftar produk yang digunakan untuk membuat menu tersebut. Hal ini digunakan sehingga menu yang direkomendasikan oleh GoDeliv dapat tersambung dengan daftar produk yang ada di *smart cart*. Berikut merupakan struktur basis data setelah integrasi.



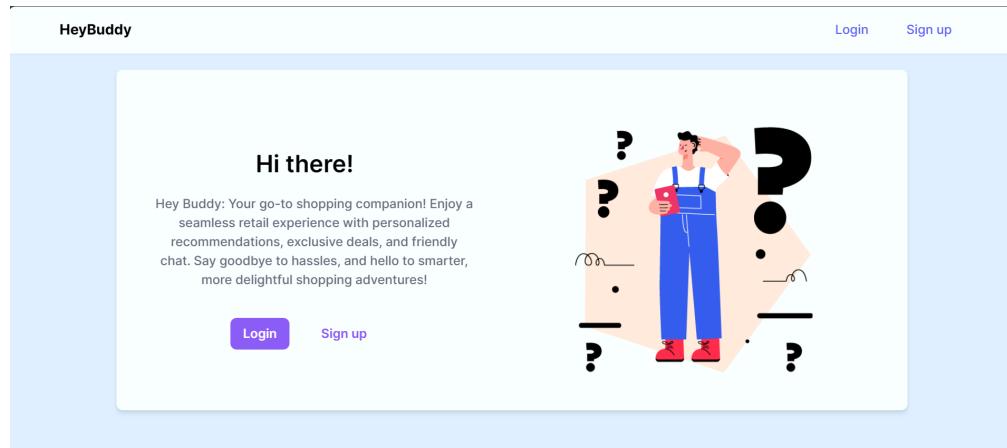
Berikut merupakan *class diagram* dari *personalized smart cart chatbot* untuk menggambarkan hubungan setiap objek yang ada di aplikasi.



Berikut merupakan hasil tampilan antarmuka *website personalized smart cart chatbot*.

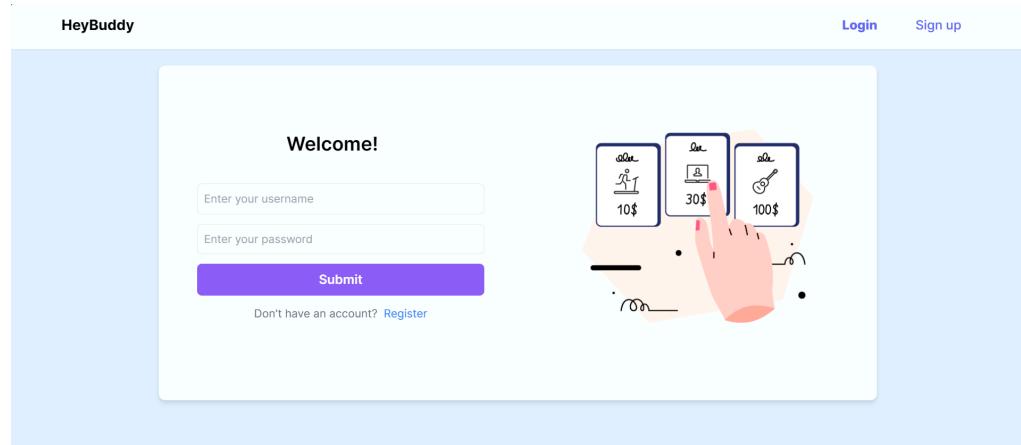
1. Landing Page

Halaman ini merupakan tampilan antarmuka pertama kali ketika pengguna mengakses aplikasi. Terdapat tombol-tombol yang dapat mengarahkan pengguna ke halaman *login* maupun *register*.



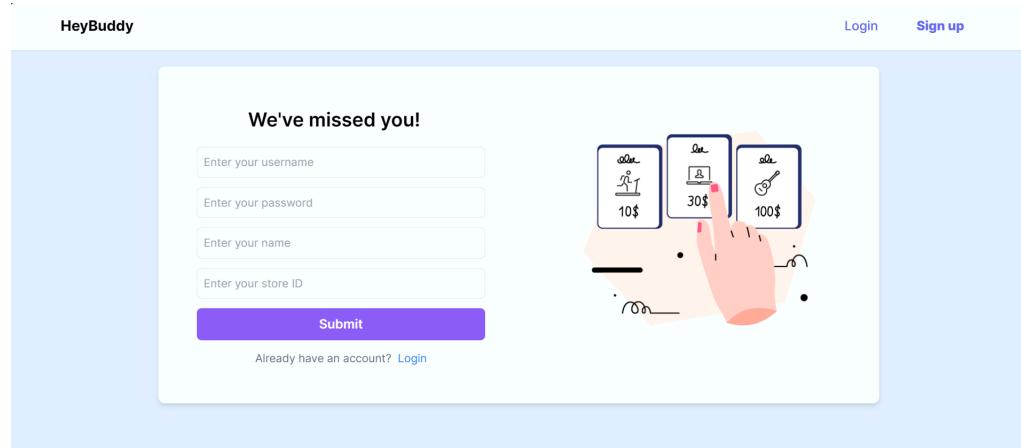
2. Halaman Login

Halaman ini merupakan tampilan antarmuka untuk pengguna melakukan login. Terdapat *form* yang meminta masukan *username* dan *password* dari pengguna. Terdapat juga tombol *submit* yang akan dilakukan otentifikasi *username* dan *password* masukan ketika ditekan. Jika *username* maupun *password* pengguna salah, maka akan diarahkan kembali ke halaman login. Selain itu, terdapat juga *hyperlink* Register yang akan mengarahkan pengguna ke halaman *register* ketika ditekan. Pada navbar, penanda di halaman mana *user* berada akan di-*bold*.



3. Halaman Register

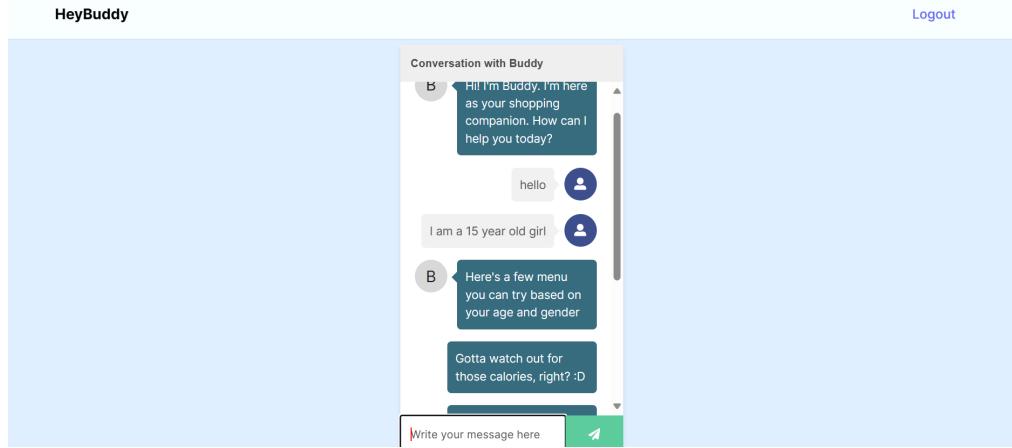
Halaman ini merupakan tampilan antarmuka untuk pengguna melakukan register. Terdapat *form* yang meminta masukan *username*, *password*, *name*, dan ID toko pengguna. Jika data masukan tidak valid, maka pengguna akan diarahkan kembali ke halaman register. Terdapat juga *hyperlink* Login yang akan mengarahkan pengguna ke halaman login ketika ditekan.



4. Halaman Chat

Halaman ini merupakan halaman utama dimana pengguna akan pertama diarahkan ketika berhasil melakukan login. Di halaman ini, pengguna dapat berinteraksi dengan *chatbot* untuk meminta informasi maupun

rekomendasi produk maupun menu sesuai dengan target kalori, umur, dan jenis kelamin pengguna. Terdapat juga tombol Logout di Navbar yang ketika ditekan akan mengarahkan pengguna kembali ke *landing page*.



D. DOKUMENTASI API

Terdapat beberapa *endpoint* dari layanan GoDeliv yang digunakan di dalam *endpoint Smart Cart* ketika melakukan integrasi. Berikut merupakan *endpoint API* yang ada pada layanan *personalized smart cart chatbot* secara keseluruhan.

No	Endpoint	Method	Fungsi	Authorization	Role
1	/token	POST	Melakukan login untuk mendapatkan token	False	semua
2	/user	GET	Mengambil semua <i>user</i>	True	admin
3	/user	POST	Melakukan registrasi <i>user</i>	False	semua
4	/user/{username}	PUT	Melakukan perubahan pada data <i>user</i>	True	admin
5	/user/{username}	DELETE	Menghapus <i>user</i> sesuai username masukan	True	admin
6	/product	GET	Mengambil semua produk	True	semua
7	/product/{name_product}	GET	Mengambil rekomendasi produk berdasarkan nama produk masukan	True	semua
8	/product	POST	Menambahkan produk baru di database	True	admin

9	/product/{id_product}	PUT	Mengubah data produk di <i>database</i>	True	admin
10	/product/{id_product}	DELETE	Menghapus produk di <i>database</i>	True	admin
11	/cart	GET	Mengambil informasi produk yang ada di <i>cart user</i>	True	semua
12	/cart	PUT	Melakukan <i>assign cart</i> ke <i>user</i>	True	semua
13	/detail_cart	POST	Menambah atau mengurangi produk di <i>cart</i>	True	semua
14	/cart	DELETE	Menghapus semua produk di <i>cart user</i> dan menghapus <i>cart user</i>	True	semua
15	/transaction/{username}	GET	Mengambil transaksi historis <i>user</i> sesuai <i>username</i> masukan	True	semua
16	/detail_transaction	GET	Mengambil informasi transaksi historis dan memberikan rekomendasi produk	True	semua
17	/detail_transa	GET	Memberikan	True	semua

	ction/{id_product}		rekомендasi produk yang sering dibeli dengan produk sesuai id masukan		
18	/transaction	POST	Melakukan <i>checkout cart</i> , produk <i>cart</i> dipindahkan ke transaksi dan <i>cart user</i> dihapus	True	semua
19	/recommendations/{username}	GET	Memberikan rekomendasi menu dari layanan GoDeliv berdasarkan target kalori	True	semua
20	/recommendations	GET	Memberikan rekomendasi menu dari layanan GoDeliv berdasarkan umur dan jenis kelamin	True	semua
21	/menus/{menu}	GET	Memberikan produk-produk yang dibutuhkan untuk membuat sebuah menu	True	semua
22	/recommendations	POST	Menambahkan data <i>user</i> di layanan GoDeliv	True	semua

Berikut merupakan dokumentasi *endpoint* API yang digunakan di layanan *personalized smart cart chatbot*.

1. Create User

Endpoint	<code>{host}/user</code>
Method	POST
Request	<p>Header Content-type: application/x-www-form-urlencoded</p> <p>Request Body Username: string Password: string Name: string Store_id: string</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body “User yuklesgo has been added”</p> <p>422 - Unprocessable Entity Response body { “Detail” : “Username is taken. Choose another username” }</p> <p>422 - Unprocessable Entity Response body { “Detail” : “Data exceeds the length limit. Fill out a shorter data” }</p>

2. Login

Endpoint	<code>{host}/token</code>
Method	POST
Request	<p>Header Content-type: application/x-www-form-urlencoded</p>

	<p>Request Body Form Data</p> <p>Username: string</p> <p>Password: string</p>
Response	<p>Example:</p> <p>200 - Successful Response</p> <p>Response body</p> <pre>{ "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJudXRyaXNhc2kifQ.qgu8mLtstmyqMxDLIjkNhX2c-_Arehqt5pYy8sHIM", "token_type": "bearer" }</pre> <p>401 - Unauthorized</p> <p>Response body</p> <pre>{ "Detail": "Invalid username or password" }</pre>

3. Retrieve all users

Endpoint	<code>{host}/user</code>
Method	GET
Request	<p>Header</p> <p>Authorization : bearer token</p>
Response	<p>Example:</p> <p>200 - Successful Response</p> <p>Response body</p> <pre>[[{ "aintrist": "\$2b\$12\$BK/SmoBh2fKL9p2683yrpeigqIxpKy5QUoJfBTqbeE6iCHo7xU5NG", "trista", "customer", "A" }]</pre>

	<pre>["audarcy", "\$2b\$12\$DRB7GcziwAFF4S9eToz5WeCQF2Ur.VONGqQ Kj95B9l9Tj1417ghtC", "odre", "customer", "A"], ["nutrisasi", "\$2b\$12\$AZ2R/41pCpSOMj38gQ/NgucDR3lV8pIAGthm0 6XJf71zueBOXTKQC", "sasi", "admin", "A"]]</pre> <p>401 - Unauthorized Response body</p> <pre>{ "Detail" : "User is unauthorized" }</pre> <p>404 - Not Found Response body</p> <pre>{ "Detail" : "No user found" }</pre>
--	--

4. Update user

Endpoint	<code>{ {host} }/user/{username}</code>
Method	PUT
Request	<p>Header Authorization : bearer token</p> <p>Request Body Username : string Password : string (optional, default: None) Name: string (optional, default: None)</p>

	Role: string (optional, default: None)
Response	<p>Example:</p> <p>200 - Successful Response Response body “User yuklesgo has been updated”</p> <p>401 - Unauthorized Response body</p> <pre>{ "Detail": "User is unauthorized" }</pre> <p>422 - Unprocessable Entity Response body</p> <pre>{ "Detail": "Role is not valid. Role can only be between admin or customer" }</pre> <p>422 - Unprocessable Entity Response body</p> <pre>{ "Detail": "No new input data" }</pre>

5. Delete user

Endpoint	<code>{host}/user/{username}</code>
Method	DELETE
Request	<p>Header Authorization : bearer token</p> <p>Request Body Username : string</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body</p>

	<p>“User with username yuklesgo has been deleted”</p> <p>401 - Unauthorized Response body { “Detail” : “User is unauthorized” }</p> <p>404 - Not Found Response body { “Detail” : “No user found with username yuklesgo” }</p>
--	--

6. Retrieve all product

Endpoint	<code>{host}/product</code>
Method	GET
Request	Header Authorization : bearer token
Response	<p>Example:</p> <p>200 - Successful Response Response body</p> <pre>[1, "Pensil", 1500, "Faber-Castell", 10, "Alat Tulis", "Stationary", "A"], [2, "Pensil", 2000, "Kenko", 75, "Alat Tulis", "Stationary",]</pre>

	<pre> "A"] } 404 - Not Found Response body { "Detail" : "No product found" } </pre>
--	--

7. Create product

Endpoint	<code>{host}/product</code>
Method	POST
Request	<p>Header Authorization : bearer token</p> <p>Request Body Name: string Price: integer Brand : string Stock: integer Category: string Aisle: string</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body “Product has been added”</p> <p>401 - Unauthorized Response body { “Detail” : “User is unauthorized” }</p> <p>422 - Unprocessable Entity Response body { “Detail” : “Price and stock value should be at least 0. Product meja has not been added” }</p>

8. Search info product

Endpoint	<code>{host}/product/{name_product}</code>
Method	GET
Request	<p>Header Authorization : bearer token</p> <p>Request Body name_product: string</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body "milk is in aisle Non-Dairy\nAlmond Milk are available with brand NutriDelight for Rp.12000\nSoy Milk are available with brand SoyGoodness for Rp.10000\nOat Milk are available with brand OatBoost for Rp.15000\nWhole Milk are available with brand FarmFreshDairy for Rp.8000\n"</p> <p>404 - Not Found Response body { "Detail" : "No information found for product meja" }</p>

9. Update product

Endpoint	<code>{host}/product/{id_product}</code>
Method	PUT
Request	<p>Header Authorization : bearer token</p> <p>Request Body id_product: integer Price: integer (optional, default: 0) Stock: string (optional, default: 0)</p>
Response	Example:

	<p>200 - Successful Response Response body “Product with id_product 1 has been updated”</p> <p>401 - Unauthorized Response body { “Detail” : “User is unauthorized” }</p> <p>404 - Not Found Response body { “Detail” : “No product found with id 1000” }</p>
--	--

10. Delete product

Endpoint	<code>{host}/product/{id_product}</code>
Method	DELETE
Request	Header Authorization : bearer token Request Body <code>id_product</code> : integer
Response	Example: <p>200 - Successful Response Response body “Product with id_product 1 has been deleted”</p> <p>401 - Unauthorized Response body { “Detail” : “User is unauthorized” }</p> <p>404 - Not Found Response body {</p>

	<pre>{ "Detail": "No product found with id_product 1" }</pre>
--	---

11. Get info cart

Endpoint	<code>{host}/cart</code>
Method	GET
Request	<p>Header Authorization : bearer token</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body</p> <pre>[[1, 1, 1, 1], ["Total price in cart is 1500"]]</pre> <p>422 - Unprocessable Entity Response body</p> <pre>{ "Detail": "Please assign user to cart" }</pre> <p>404 - Not Found Response body</p> <pre>{ "Detail": "No product found in cart with id 1" }</pre>

12. Assign cart

Endpoint	<code>{host}/cart</code>
Method	PUT

Request	Header Authorization : bearer token
Response	<p>Example:</p> <p>200 - Successful Response Response body "User nutrisasi is using cart with id 1"</p> <p>404 - Not Found Response body { "Detail" : "No empty cart is found. Sorry :("} }</p> <p>422 - Unprocessable Entity Response body { "Detail" : "User is already assigned to cart with id 1"} }</p>

13. Delete user cart

Endpoint	{ {host} }/cart
Method	DELETE
Request	Header Authorization : bearer token
Response	<p>Example:</p> <p>200 - Successful Response Response body "Cart has been emptied and reassigned"</p> <p>422 - Unprocessable Entity Response body { "Detail" : "No cart assigned to user found"} }</p>

14. Add item to cart

Endpoint	{ {host} }/detail_cart
----------	------------------------

Method	POST
Request	<p>Header Authorization : bearer token</p> <p>Request Body id_product: integer addClick: boolean</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body “Product with id 1 has been added to cart”</p> <p>200 - Successful Response Response body “Cart has been updated”</p> <p>404 - Not Found Response body { “Detail” : “No product found under id 1” }</p>

15. Get transaction

Endpoint	<code>{host}/transaction/{username}</code>
Method	GET
Request	<p>Header Authorization : bearer token</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body [[19, "2023-12-08T18:55:31", 1500, "nutrisasi", "A"]</p>

	<p>]</p> <p>404 - Not Found</p> <p>Response body</p> <pre>{ "Detail" : "No transaction found for user nutrisasi" }</pre>
--	---

16. Get information transaction

Endpoint	<code>{host}/detail_transaction</code>
Method	GET
Request	<p>Header Authorization : bearer token</p> <p>Request Body name_product: string (optional, default: None)</p>
Response	<p>Example:</p> <p>200 - Successful Response</p> <p>Response body</p> <pre>[[1, 1, 1, 5, "A"], [1, 2, 2, 4, "A"], [1, 3, 5, 4, "A"]]</pre>

	<pre>]]</pre> <p>200 - Successful Response Response body "The most bought brand for product pensil is Faber-Castell"</p> <p>404 - Not Found Response body { "Detail" : "No transaction found for product: banana" }</p> <p>404 - Not Found Response body { "Detail" : "No product found with name = banana" }</p>
--	--

17. Product recommendation

Endpoint	<code>{host}/detail_transaction/{id_product}</code>
Method	GET
Request	<p>Header Authorization : bearer token</p> <p>Request Body id_product: integer</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body "Product frequently bought with Pensil: ['Pensil Warna', 'Printer', 'Penghapus']"</p> <p>200 - Successful Response Response body "We have no information on this. Be the first to try it out!"</p> <p>404 - Not Found Response body {</p>

	<p>“Detail” : “No information found for product with ID = 1”</p> <p>}</p>
--	---

18. Create transaction

Endpoint	{ {host} }/transaction
Method	POST
Request	Header Authorization : bearer token
Response	<p>Example:</p> <p>200 - Successful Response Response body “Transaction success”</p> <p>422 - Unprocessable Entity Response body { “Detail” : “Please assign cart to user nutrisasi” }</p>

19. Get recommendation godeliv

Endpoint	{ {host} }/recommendations/{user_name}
Method	GET
Request	<p>Header Authorization : bearer token</p> <p>Request Body User_name : string Target_kalori: integer</p>
Response	<p>Example:</p> <p>200 - Successful Response Response body “Unfortunately, we don’t have menus lower than your target calories”</p>

	<p>200 - Successful Response</p> <p>Response body</p> <p>"Here's a few menu you can try based on your target calories\nSteamed Vegetables has 250 calories\nProtein Smoothie has 200 calories\nOatmeal has 300 calories\nSteamed Bun has 340 calories\n"</p>
--	---

20. Get standard recommendation godeliv

Endpoint	<code>{host}/recommendations</code>
Method	GET
Request	<p>Header</p> <p>Authorization : bearer token</p> <p>Request Body</p> <p>Username : string (optional, default: None) umur: integer (optional, default: None) jenis_kelamin: string (optional, default: None)</p>
Response	<p>Example:</p> <p>200 - Successful Response</p> <p>Response body</p> <p>"Here's a few menu you can try\nCaesar Salad has 500 calories\nSteamed Vegetables has 250 calories\nProtein Smoothie has 200 calories\nOatmeal has 300 calories\nSteak has 600 calories\nBurrito has 450 calories\nPotato and Cheese has 700 calories\nSteamed Bun has 340 calories\n"</p> <p>200 - Successful Response</p> <p>Response body</p> <p>"Here's a few menu you can try based on your age and gender\nGotta watch out for those calories, right? :D\nCaesar Salad has 500 calories\nSteamed Vegetables has 250 calories\nProtein Smoothie has 200 calories\nOatmeal has 300 calories\nBurrito has 450 calories\nSteamed Bun has 340 calories\n"</p>

21. Add user godeliv

Endpoint	<code>{host}/recommendations</code>
----------	-------------------------------------

Method	POST
Request	<p>Header Authorization : bearer token</p> <p>Request Body User_name : string jenis_kelamin: string umur: integer target_kalori: integer</p>
Response	<p>Example:</p> <p>200 - Successful Response</p> <p>Response body</p> <pre>{ "Message" : "User berhasil ditambahkan" }</pre>

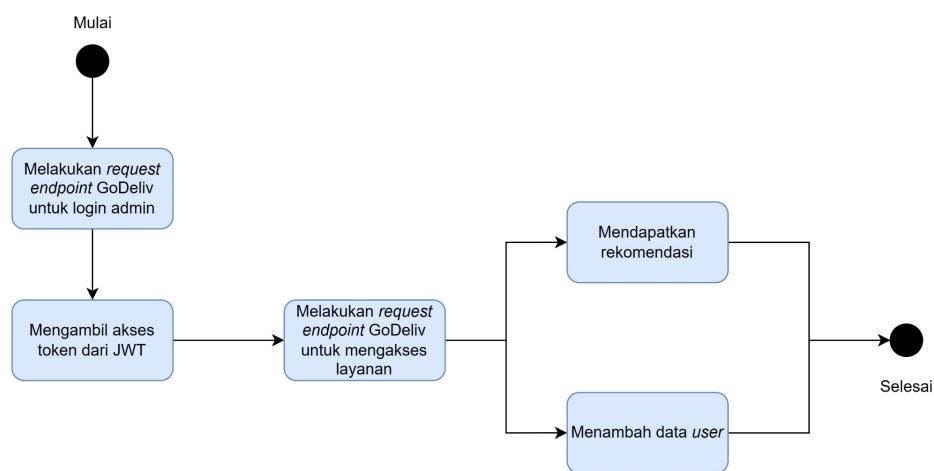
22. Get recipe

Endpoint	<code>{host}/menus/{menu}</code>
Method	GET
Request	<p>Header Authorization : bearer token</p> <p>Request Body Menu: string</p>
Response	<p>Example:</p> <p>200 - Successful Response</p> <p>Response body</p> <p>"To prepare a burrito, you'll need: ground beef, onion, rice, cheese, salsa, and tortillas"</p> <p>404 - Not Found</p> <p>Response body</p> <pre>{ "Detail" : "Sorry, we don't have the information at the moment" }</pre>

E. ANALISIS

Dalam proses pengembangan layanan *personalized smart cart chatbot*, dilakukan beberapa modifikasi dari desain awal dengan beberapa pertimbangan. Pada waktu proses integrasi, struktur basis data belum memiliki atribut ID *store*. Namun, karena layanan *smart cart chatbot* juga akan diintegrasikan oleh layanan lain, saya berpikir untuk menambahkan atribut ID *store*. Salah satu alasan utamanya adalah karena *chatbot* menggunakan data transaksi historis dan data-data produk di suatu toko. Ketika *chatbot* diintegrasikan, maka *chatbot* harus dapat memberikan rekomendasi yang sesuai dengan toko dimana *cart* berada. Pada akhirnya, saya memutuskan untuk menambahkan atribut *store* ID sehingga produk, transaksi, *user*, dan *cart* akan memiliki identifikasi sesuai toko.

Kemudian dalam proses mengintegrasikan layanan *smart cart chatbot* dengan layanan GoDeliv, didapatkan bahwa layanan GoDeliv harus melalui proses *authorization* atau login untuk dapat mengakses layanannya. Kemudian, *user* yang sudah melalui proses *authorization* dapat mendaftarkan data *user* lain. Maka dari itu, saya memutuskan untuk hanya mendaftarkan satu *user* saja sebagai admin dari *smart cart*, dan admin tersebut yang akan mendaftarkan data *user* lain. Hal ini saya lakukan agar *user smart cart* tidak perlu melakukan registrasi lagi di layanan GoDeliv untuk mendapatkan rekomendasinya. Berikut merupakan diagram alur proses integrasi dengan layanan GoDeliv.



Layanan kemudian di-*deploy* menggunakan Docker *container instance* dengan langkah-langkah yang sudah disebutkan sebelumnya pada bagian [docker containerization](#). Untuk bagian *testing*, saya mencoba semua *endpoint* di Swagger UI yang sudah di *deploy* menggunakan Docker. Dari hasil *testing*, didapatkan keseluruhan *endpoint* dapat diakses dengan benar. Setelah itu, saya melakukan *deploy* di Vercel dengan langkah-langkah yang sudah disebutkan sebelumnya pada bagian [Vercel deployment](#). Pada saat ini, folder *frontend* dan *backend* masih disatukan di 1 *repository*. Namun, ternyata untuk melakukan *deploy* di Vercel yang memang digunakan untuk *deploy repository frontend*, sehingga ketika saya mencoba melakukan *deploy* dengan *repository* yang berisi *frontend* dan *backend*, Vercel tidak mengenali *framework frontend* pada *repository* ketika melakukan *build website* sehingga proses *deployment* gagal. Setelah itu, saya membuat *repository* baru untuk *frontend* saja dan mengulangi kembali proses *deployment* menggunakan Vercel.

Deployment berhasil dilakukan dan saya dapat mengakses *website* melalui link *url* Vercel app. Namun, ketika saya mencoba login, saya tidak dapat mengakses *endpoint API* dengan *error* yang mengindikasikan bahwa protokol HTTP tidak didukung. Setelah saya membaca dokumentasi, ternyata memang Vercel menggunakan protokol HTTPS sehingga ketika berusaha mengakses *endpoint API* menggunakan protokol HTTP, akses gagal. Sehingga saya mengganti *link url API* dengan *resource web application* di Microsoft Azure yang menggunakan protokol HTTPS. Ketika saya sudah mengganti *link API*, *website* hasil *deploy* di Vercel berhasil berjalan dengan lancar dan *endpoint* dapat diakses.