BAB III
PELAKSANAAN KERJA MAGANG

3.1. Kedudukan dan Koordinasi

Mahasiswa tergabung dalam tim *Engineering* sebagai *Software Developer Intern*, khususnya dalam tugas sebagai *Back-end Engineer*. Mahasiswa bekerja sama dengan dua *Software Developer* lainnya dan bertanggung jawab langsung kepada Denny Susanto selaku *Technical Lead* dari tim *Back-end Engineer* dan Glenn Pratama selaku *Engineering Manager*. Mahasiswa tergabung dalam proyek Fabelio NPS, yang ditugaskan untuk meningkatkan *Nett Promoter Score* (NPS), yang menjadi salah satu KPI dari bisnis Fabelio.

Setiap tim *Engineering* bekerja sama secara langsung dengan tim *Product* yang diwakili oleh *Product Manager*. Tim *Engineering* bekerja berdasarkan permintaan dari tim *Product* yang dituangkan dalam bentuk *Product Requirement Document* (PRD). *Back-end Engineer* juga berkoordinasi langsung dengan tim *Front-end Engineer* terkait kesepakatan alur dan model data yang dikirimkan oleh API yang dikembangkan dalam bentuk Kontrak API. Selain itu, setiap *Software Developer* akan berhubungan langsung dengan tim *Quality Assurance* dalam pengujian fitur yang dikembangkan, termasuk memberikan masukkan dan saran untuk daftar *test cases* yang akan diuji.

Setiap pegawai magang di Fabelio akan didampingi oleh salah satu pegawai tetap yang ada, disebut *buddy*. Selama praktik kerja magang berlangsung,

mahasiswa bekerja sama sekaligus didampingi oleh salah satu *Software Developer* lainnya, yaitu Deli Soetiawan.

3.2. Tugas yang Dilakukan

Cakupan pekerjaan yang dilaksanakan selama praktik kerja magang meliputi pengembangan, pemeliharaan, dan perbaikkan *Database*, API, *Microservices*, serta *codebase* dari beberapa sistem yang dimiliki oleh Fabelio, meliputi:

1) **Marvel,** sistem *middleware* internal dari Fabelio yang memiliki fungsi utama sebagai integrator data pesanan dari *website* fabelio.com menuju sistem pihak ketiga, yang terdiri dari sistem ERP dan sistem pengiriman logistik dari pihak ketiga. Seiring berjalannya waktu, Marvel terus dikembangkan dengan berbagai fitur tingkat lanjut lainnya, di antaranya untuk *Stock Transfer Order* (STO) antar gudang, pemetaan lokasi pengiriman untuk kalkulasi biaya pengiriman, konfigurasi pengiriman dan instalasi produk, integrasi dengan layanan CRM pihak ketiga, dan lain-lain. Pengembangan bukan hanya dari sisi *Back-end*, tetapi juga *Front-end*.
2) **Oracle Netsuite,** sistem ERP yang sudah disebutkan sebelumnya untuk mengelola secara terintegrasi berbagai komponen proses bisnis dari Fabelio, mulai dari keuangan, rantai pasok, inventaris, penjualan, dan lain-lain.
3) ***Microservices***, meliputi layanan-layanan berbasis web yang bersifat mikro dan independen satu sama lain untuk mendukung operasional bisnis dari Fabelio – termasuk salah satunya *Marvel*. Layanan mikro lainnya di mana mahasiswa pernah terlibat dalam proses pengembangannnya adalah *Morbius*

(untuk proses terkait pengiriman) dan *Fabelio WhatsApp* (untuk automasi pengiriman pesan WhatsApp ke konsumen).

Untuk pengembangan sistem selama praktik kerja magang, sepenuhnya pengembangan dilakukan menggunakan bahasa pemrograman JavaScript dengan berbagai *frameworks* dan *libraries* pendukungnya, seperti Express[5] yang digunakan sebagai *framework* pengembangan API, ReactJS[6] sebagai *framework* pengembangan *front-end* dari sistem Marvel, Axios[7] untuk *library* yang memungkinkan melakukan *HTTP Request* dari *Front-end* ke *Back-end*, Redux[8] untuk pengelolaan *State* pada ReactJS, Sinon[9] untuk pembuatan *Unit Test*, dan lain-lain. Untuk *database*, selama praktik kerja magang berlangsung, mahasiswa pernah mengelola *database* relasional maupun non-relasional, yaitu PostgreSQL untuk *database* relasional, dan MongoDB untuk *database* non-relasional.

Pekerjaan yang telah dilakukan selama praktik kerja magang sebagai *Software Developer Intern* di Fabelio adalah sebagai berikut:

Tabel 3.1. *Timeline* Kerja Magang

No	Deskripsi	Minggu ke													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Pengenalan sistem dan arsitektur kode														

[5] https://expressjs.com/
[6] https://reactjs.org/
[7] https://axios-http.com/
[8] https://redux.js.org/
[9] https://sinonjs.org/

No	Deskripsi	Minggu ke													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	Pengerjaan Proyek "Marvel Role"														
3	Pengerjaan Proyek "*Order Splitting*"														
4	Pengerjaan Proyek "*Pickup at Showroom*"														

Tabel 3.1. menjelaskan tentang pekerjaan yang dilakukan selama periode praktik kerja magang sebagai *Software Developer Intern* di Fabelio, dengan pada setiap bagian pekerjaan dibagi kembali menjadi beberapa aktivitas, seperti yang tertera pada Tabel 3.2. di bawah ini.

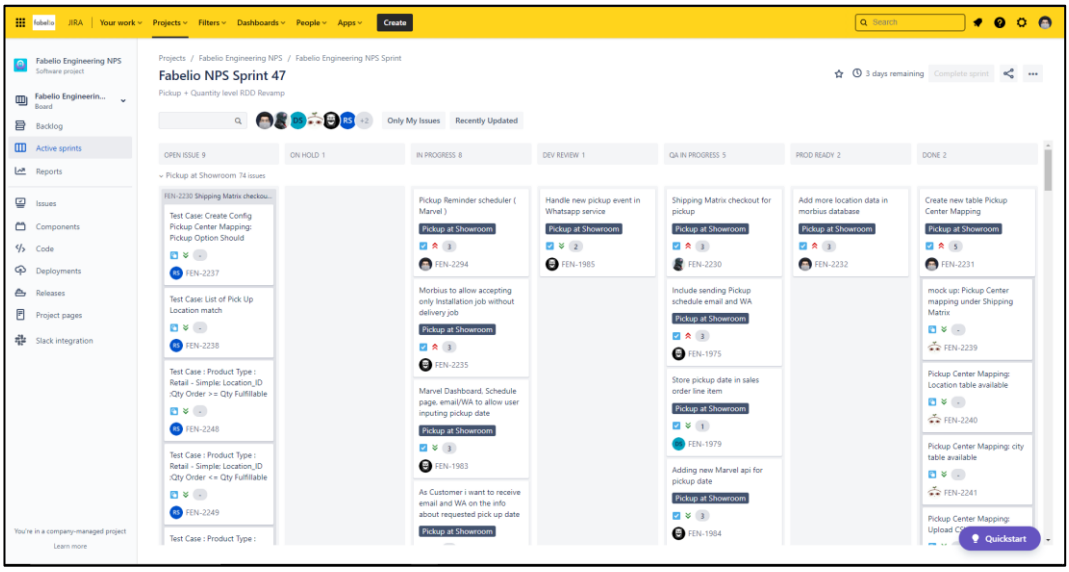
Tabel 3.2. *Timeline* Rincian Praktik Kerja Magang

No.	Kegiatan	Mulai (minggu ke-)	Selesai (minggu ke-)
1	**Pengenalan sistem dan arsitektur kode**		
	HR *Onboarding* dan *Engineering Onboarding*	3 Maret 2021 (1)	3 Maret 2021 (1)
	Melakukan *refactoring* pada *deprecated library*	4 Maret 2021 (1)	8 Maret 2021 (2)
	Melakukan *refactoring* pada *deprecated function*/*method* dari *library* pihak ketiga	9 Maret 2021 (2)	9 Maret 2021 (2)
	Menerapkan *code formatter* pada *codebase* Marvel UI	10 Maret 2021 (2)	19 Maret 2021 (3)
2	**Pengerjaan proyek "*Marvel Role*"**		
	Membuat menu "*Account*" pada Marvel UI	22 Maret 2021 (4)	24 Maret 2021 (4)

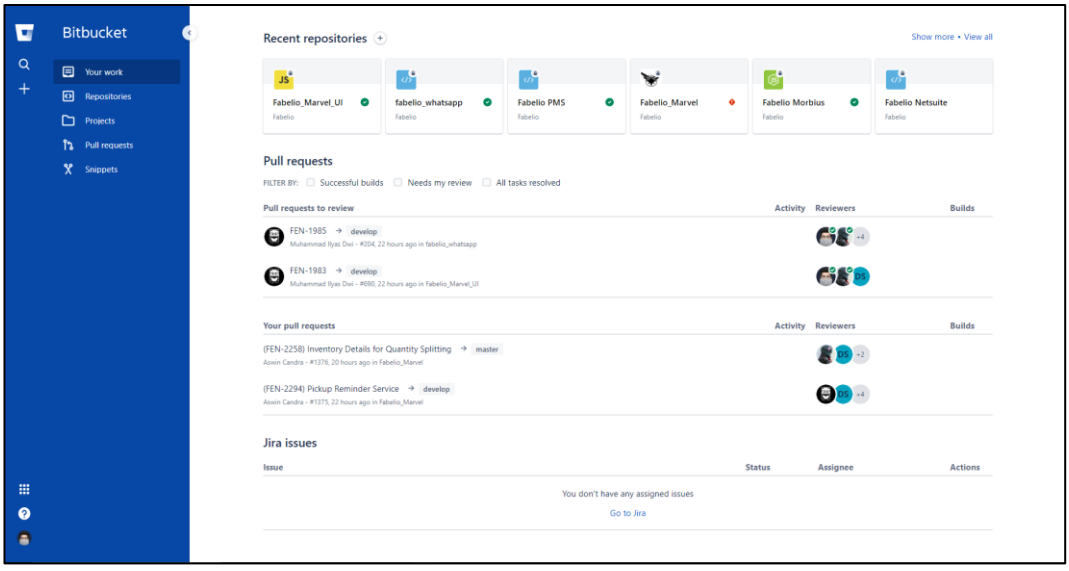
No.	Kegiatan	Mulai (minggu ke-)	Selesai (minggu ke-)
	Implementasi skema *login* gagal pada pengguna non-aktif Marvel	25 Maret 2021 (4)	26 Maret 2021 (4)
	Menambahkan objek data *permissions* pada *payload* JWT dan *state* dari ReactJS	29 Maret 2021 (5)	30 Maret 2021 (5)
	Mengimplementasi perizinan berdasarkan peran pengguna ke setiap komponen halaman Marvel	31 Maret 2021 (5)	9 April 2021 (6)
	Implementasi Halaman 403 pada Marvel UI	26 April 2021 (9)	30 April 2021 (9)
3	**Pengerjaan Proyek "*Order Splitting*"**		
	Menambah atribut "*Location*" di tingkat produk pada Marvel	12 April 2021 (7)	13 April 2021 (7)
	Menambah atribut "*Location*" di tingkat produk pada Netsuite	14 April 2021 (7)	16 April 2021 (7)
	Menambah atribut "*Carrier*" pada Opsi Pengiriman	19 April 2021 (8)	23 April 2021 (8)
	Quantity status splitting	21 Mei 2021 (12)	21 Mei 2021 (12)
4	**Pengerjaan Proyek "*Pickup at Showroom*"**		
	Pembuatan modul "*Pickup Center Mapping*"	3 Mei 2021 (10)	14 Mei 2021 (11)
	Penambahan Atribut Baru untuk Data Lokasi *Fulfillment*	17 Mei 2021 (12)	20 Mei 2021 (12)
	Pembuatan WhatsApp *Pickup Reminder*	24 Mei 2021 (13)	3 Juni 2021 (14)

Proses pengembangan *software* dilaksanakan dengan pendekatan metode *Scrum Sprint* di mana setiap *sprint* berjangka waktu 2 minggu. Setiap tugas yang diberikan dan pelacakan kemajuan pekerjaan semuanya berada dalam satu *platform* bernama Attlasian Jira, yang juga terintegrasi dengan *Git management tool* bernama

Bitbucket untuk manajemen *source code*. Tampilan secara umum dari Jira dan Bitbucket dapat dilihat pada Gambar 3.1 dan Gambar 3.2.



Gambar 3.1. Tampilan Umum Attlasian Jira



Gambar 3.2. Tampilan Umum Bitbucket

3.3. Kegiatan Praktik Kerja Magang

3.3.1. *HR Onboarding* dan *Engineering Onboarding*

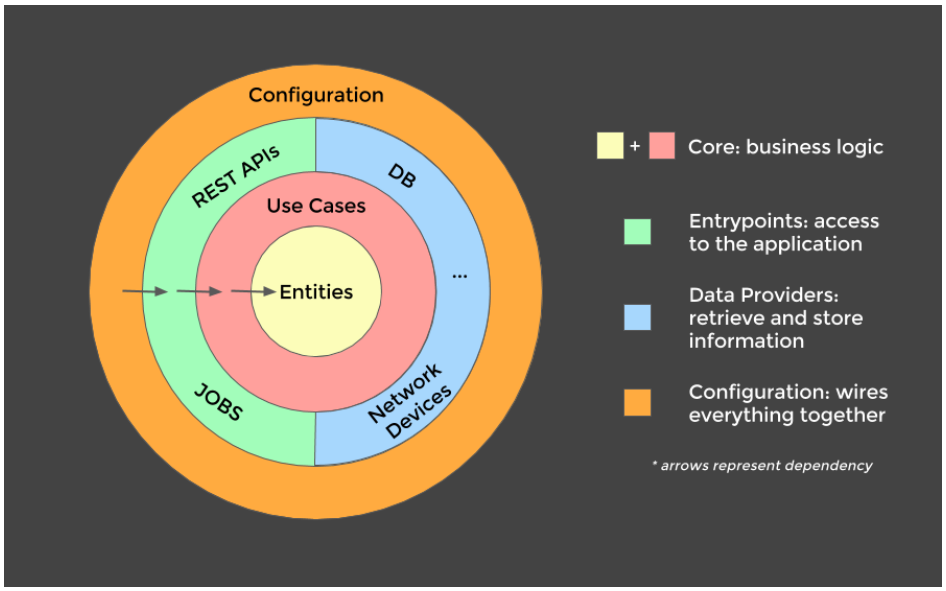
Pada saat *onboarding* dengan tim HR, mahasiswa diperkenalkan dengan istilah-istilah yang menjadi beberapa metrik Indikator Kinerja Utama dari Fabelio, seperti *Nett Merchandise Value* (NMV), *Nett Promote Score* (NPS), *Conversion Rate* (CR), *Average Order Value* (AOV), dan *Gross Merchandise Value* (GMV). Indikator-indikator tersebut yang diharapkan perusahaan kepada mahasiswa menjadi orientasi ketika mahasiswa melaksanakan pekerjaannya.

Ketika melaksanakan *onboarding* dengan tim *Engineering*, mahasiswa mulai diperkenalkan dengan sistem-sistem yang dikelola oleh tim *Product* & *Engineering* khususnya untuk *back-end*, yang beberapa di antaranya sudah disebutkan pada sub-bab sebelumnya. Selain itu, mulai diperkenalkan juga arsitektur *software* dari keseluruhan kode yang dimiliki oleh Fabelio. Fabelio menerapkan pendekatan *Clean Architecture*, sebuah arsitektur pengembangan perangkat lunak yang menjadikan *business logic* dan *use cases*-nya sebagai pusat dari arsitekturnya.

Gambar 3.3 menunjukkan struktur aplikasi dengan pendekatan *Clean Architecture*. Pada umumnya, aplikasi dengan *Clean Architecture* memiliki empat komponen utama [5], yaitu:

1) ***Entities***: merepresentasikan setiap *business domain*, biasanya dalam bentuk objek.

2) ***Use Cases***: merepresentasikan *business logic*, mengatur aliran data ke dan dari *Entities*, dan mengarahkan entitas tersebut untuk menggunakan *use case* yang ada.
3) ***Interface Adapters***: pintu untuk berinteraksi dengan aplikasi, memberikan *trigger* kepada *Use Cases* untuk menjalankan fungsinya. Biasanya *adapters* ini berbentuk REST API, *Cronn Job*, dan lain-lain; bertugas untuk mengonversi data dari format yang dimengerti oleh *Use Cases* dan *Entities*, ke format yang dimengerti oleh *frameworks* (misal: *web framework*) atau *drivers* lainnya (misal: *database*), begitu juga sebaliknya.
4) ***Frameworks & Adapters***: berisikan *frameworks* dan *drivers* eksternal, dan juga *file* konfigurasi untuk menjalin seluruh komponen menjadi satu kesatuan.



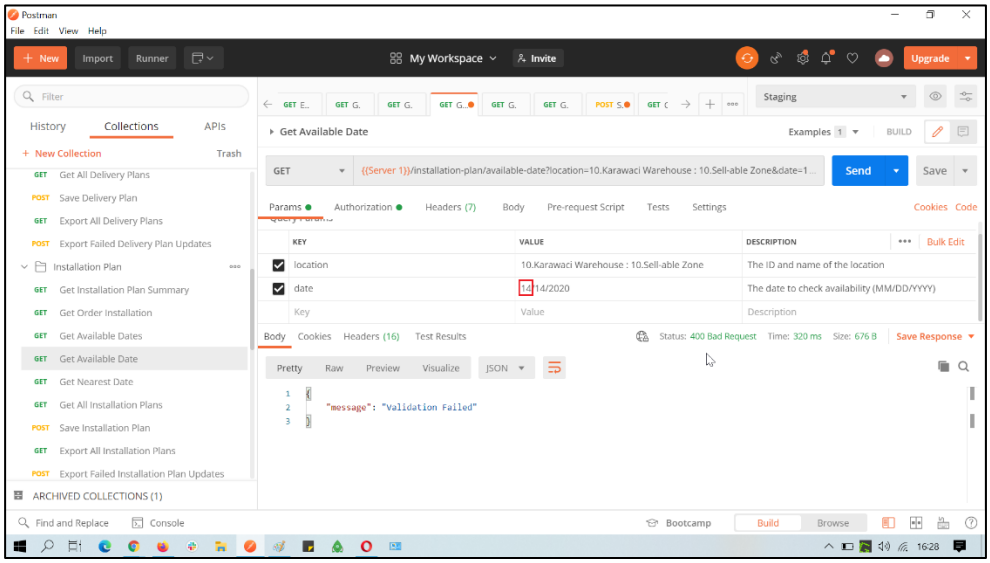
Gambar 3.3. Struktur Aplikasi dari *Clean Architecture*

Sumber: [6]

Arsitektur semacam ini membawa beberapa keuntungan [6], seperti:

1) **Bisnis-sentris:** ketika melihat struktur kode, kita dapat langsung mengetahui apa yang dikerjakan oleh satu *file* kode tersebut tanpa harus terlalu lama berkutat dengan detail teknis.
2) **Independen dari *Database*:** aplikasi terlihat benar-benar sebagai aplikasi bisnis, bukan hanya sebagai aplikasi CRUD ke *database*, yang masih menjadi mayoritas pola gaya pengembangan aplikasi saat ini.
3) ***Frameworks* terisolasi:** ini adalah bagian yang sering kali terus berubah seiring berjalannya bisnis dan teknologi, sehingga *Clean Architecture* menyiasati dengan mengisolasi bagian ini sehingga ketika ada perubahan pada *framework* atau dependensi eksternal lainnya, tidak akan ada perubahan yang terlalu masif pada *codebase* yang sudah ada. Perubahan dapat dilakukan dengan cepat tanpa kesulitan berarti, sehingga mendukung *Continuous Integration & Delivery* (CI/CD) pada pengembangan, yang dapat mendorong bisnis bergerak lebih cepat juga.
4) **Efektivitas pengujian**: mendukung pengujian hingga ke tingkat paling dasar, yaitu pengujian unit (*Unit Testing*), sehingga pengujian menjadi lebih handal serta dapat mengurangi pengujian secara grafis (GUI) untuk unit-unit yang memang tidak memungkinkan atau memerlukan pengujian secara grafis.

2. Melakukan *Refactoring* pada *Deprecated Library*



Gambar 3.4. Hasil *Refactoring Library* Validator *Joi*

Untuk memulai penyesuaian dengan arsitektur kode dan *framework* yang digunakan, mahasiswa diberi tugas awal untuk melakukan *refactoring library* yang sudah *deprecated*, yaitu dengan melakukan *uninstallation library* yang *depcrecated* tersebut, dan kemudian melakukan instalasi *library* penggantinya. Pada saat itu, *library* yang diganti adalah `joi` menjadi `@hapi/joi`, *library* yang digunakan sebagai validator data yang dikirim ke API. Setelah dilakukan *refactoring*, dipastikan validasi data tetap berjalan dengan baik. Gambar 3.4 menunjukkan salah satu contoh validasi data tetap berjalan dengan baik setelah dilakukan *refactoring library* dari Joi, di mana API akan memberi respon bahwa API menolak karena terdapat kegagalan validasi yang dikarenakan data masukkan berupa tanggal tidak valid.

3. Melakukan *Refactoring* pada *Deprecated Function*/*Method* dari *Library* Pihak Ketiga

Selain itu, mahasiswa juga diminta untuk melakukan *refactoring* kode untuk mengganti fungsi `equal()` pada library *assert* menjadi `strictEqual()` dikarenakan fungsi `equal()` yang juga sudah *deprecated*. Hal ini diterapkan pada setiap *Unit Test* di seluruh *codebase* yang dimiliki Fabelio.

4. Menerapkan *Code Formatter* pada *Codebase* Marvel UI

Pada akhir pekerjaan ini, mahasiswa diminta untuk menerapkan *code formatter* pada *codebase* dari Marvel UI. *Code Formatter* ditujukan agar kode menjadi lebih mudah dibaca oleh *developer* lain dan mengikuti kaidah *best practice* dari penulisan kode JavaScript. Tugas ini dilaksanakan dengan memanfaatkan salah satu *library* pemformat otomatis, yaitu StandardJS[10].

3.3.2. Pengerjaan Proyek "*Marvel Role*"

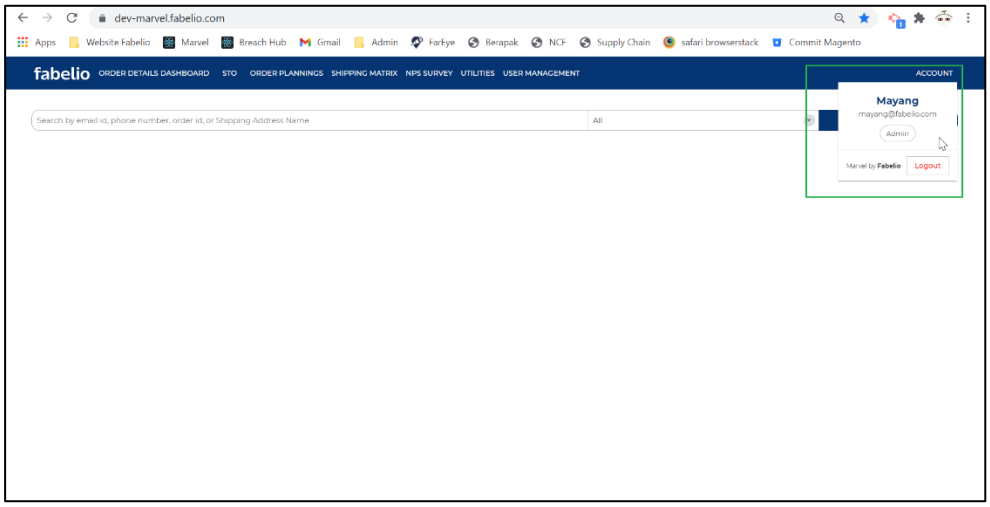
Marvel adalah *software* internal Fabelio, dimulai dengan tujuan sebagai jembatan antara sumber data penjualan Magento (fabelio.com) dan Netsuite sebagai sistem ERP. Tetapi seiring berjalannya waktu, Marvel terus berkembang dengan semakin banyaknya fitur yang ditambahkan ke Marvel, serta melibatkan lebih banyak pengguna dengan peran pada *platform* Marvel yang lebih beragam. Proyek "*Marvel Role*" ini ditujukan untuk mengembangkan sistem *Identity Access Management* (IAM), yang dapat

[10] https://standardjs.com/

menetapkan fungsi akses setiap pengguna di Marvel berdasarkan perannya masing-masing.

1. Membuat Menu "*Account*" pada Marvel UI

Mula-mula, mahasiswa diminta untuk menambahkan menu baru pada Marvel UI, yaitu "*Account*". Menu ini ditujukan untuk menunjukkan profil dari pengguna, berupa nama, *e-mail*, dan peran yang dimiliki oleh pengguna tersebut. Gambar 3.5 menunjukkan contoh tampilan tab menu "*Account*" ini.

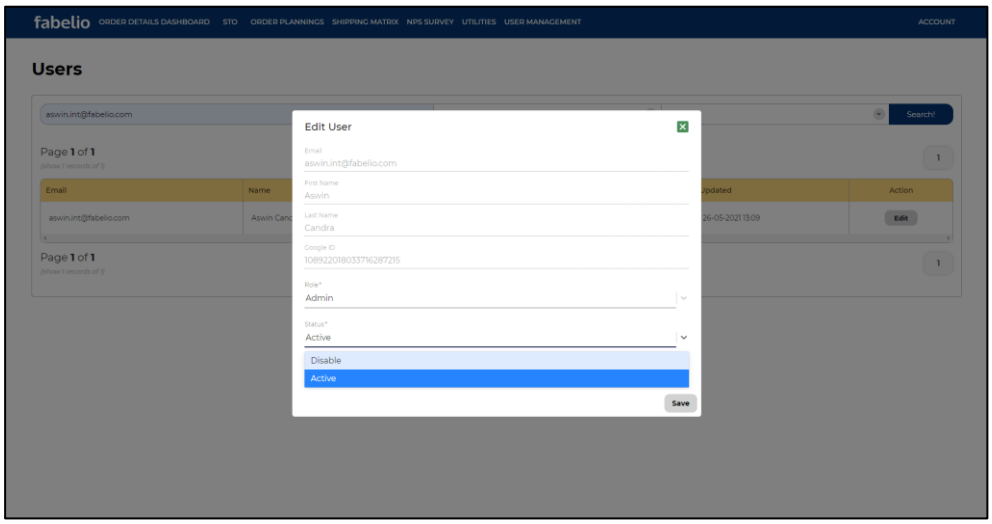


Gambar 3.5. Contoh Tampilan Tab Menu *Account*

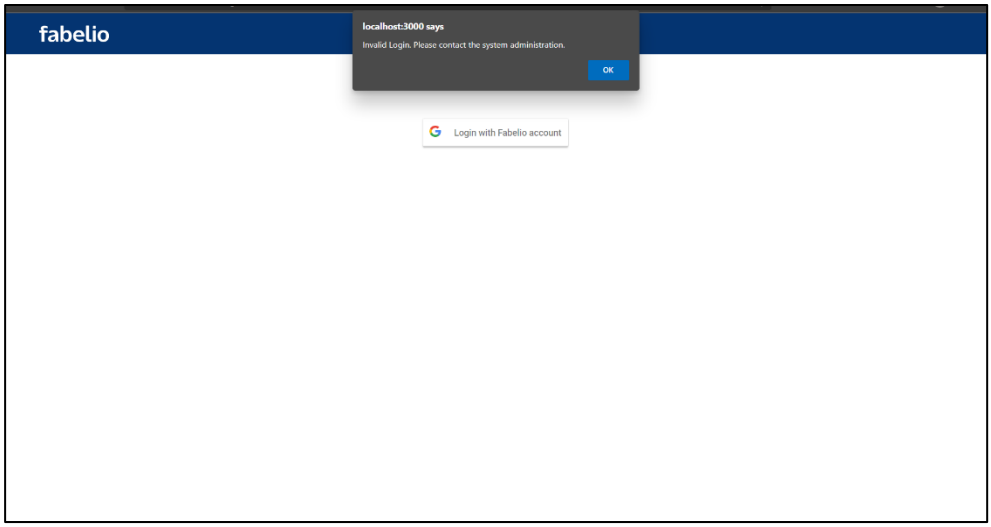
2. Implementasi Skema *Login* Gagal pada Pengguna Non-aktif Marvel

Lalu, mahasiswa diminta untuk membuat skema gagal *login* bagi pengguna yang statusnya sudah tidak aktif pada sistem Marvel. Status keaktifan pengguna ini juga dapat dikonfigurasi pada Marvel UI seperti yang ditunjukkan pada Gambar 3.6. Ketika pengguna dengan status tidak aktif mencoba melakukan *login*, akan muncul *pop-up* peringatan bahwa status pengguna tersebut sudah tidak aktif. Contoh dari skema ini dapat dilihat pada

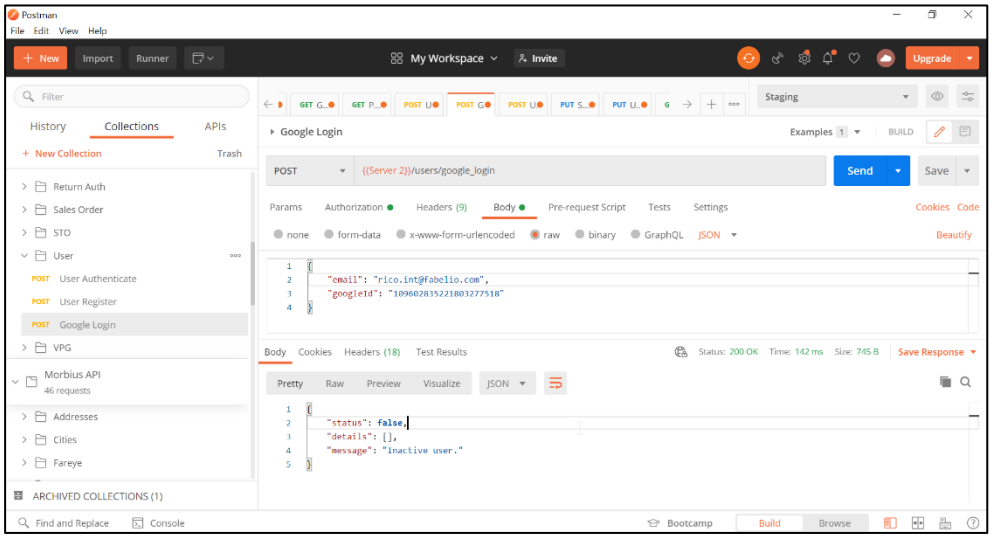
Gambar 3.7 yang menunjukkan skema dari sisi *front-end* (*end users*) dan Gambar 3.8 dari sisi *back-end* (API *call*).



Gambar 3.6. Halaman Marvel untuk Konfigurasi Status Keaktifan Pengguna



Gambar 3.7. Skema *Login* Gagal untuk Pengguna Non-aktif Marvel (*Front-end*)



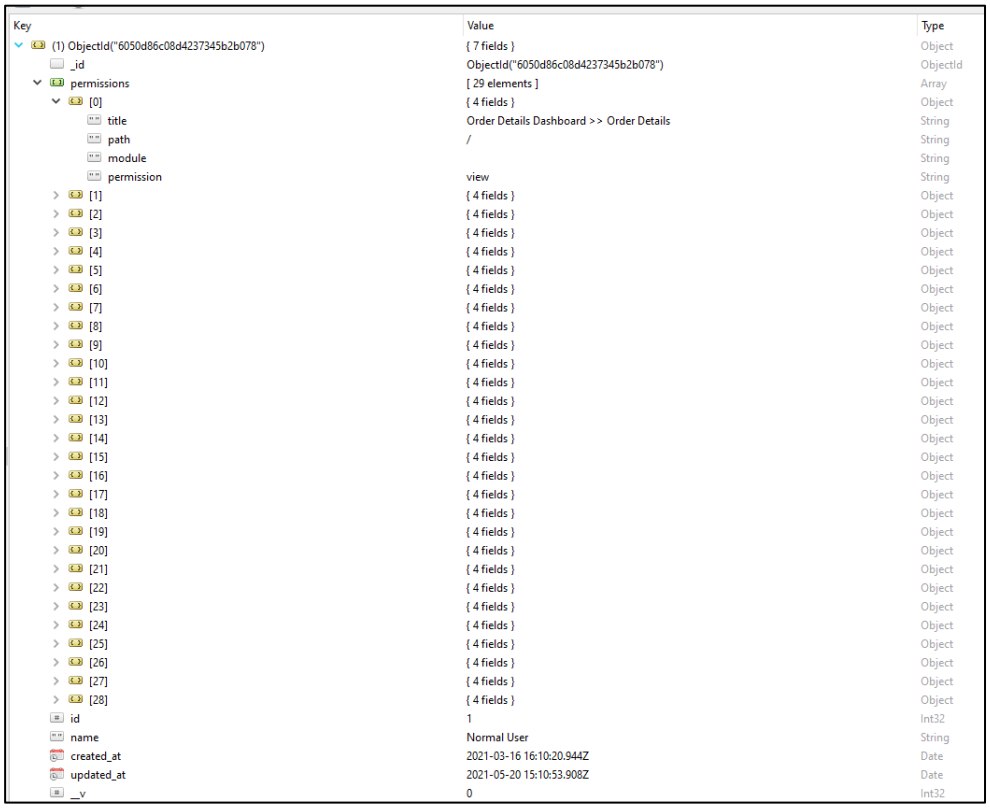
Gambar 3.8. Skema *Login* Gagal untuk Pengguna Non-aktif Marvel (*Back-end*)

3. Menambahkan Objek Data *Permissions* pada *Payload* JWT dan *State* dari ReactJS

Fitur-fitur IAM dari Marvel dapat dicapai dengan alur proses sebagai berikut: setiap data pengguna yang tersimpan dalam *collection* `users` di *database* MongoDB memiliki atribut `status` yang digunakan untuk mengetahui status keaktifan pengguna, `role` untuk menampung *Role* ID dari pengguna tersebut, dan pastinya informasi-informasi dasar seperti nama, *e-mail*, dan lainnya. *Role* ID yang dimiliki pengguna terhubung dengan *collection* lain yang bernama `roles`, yang menampung *array of objects* berisikan setiap daftar perizinan untuk peran bersangkutan. Skema dari kedua *collection* tersebut dapat dilihat pada Gambar 3.9 dan Gambar 3.10.

Key	Value	Type
(1) ObjectId("6045f2afcedbec0006f282ba")	{ 10 fields }	Object
_id	ObjectId("6045f2afcedbec0006f282ba")	ObjectId
first_name	Aswin	String
last_name	Candra	String
role	0	Int32
googleId	108922018033716287215	String
email	aswin.int@fabelio.com	String
created_at	2021-03-08 09:47:27.806Z	Date
updated_at	2021-05-26 06:29:54.731Z	Date
__v	0	Int32
active	1	Int32

Gambar 3.9. Skema *Collection users*



Key	Value	Type
(1) ObjectId("6050d86c08d4237345b2b078")	{ 7 fields }	Object
_id	ObjectId("6050d86c08d4237345b2b078")	ObjectId
permissions	[29 elements]	Array
[0]	{ 4 fields }	Object
title	Order Details Dashboard >> Order Details	String
path	/	String
module		String
permission	view	String
[1]	{ 4 fields }	Object
[2]	{ 4 fields }	Object
[3]	{ 4 fields }	Object
[4]	{ 4 fields }	Object
[5]	{ 4 fields }	Object
[6]	{ 4 fields }	Object
[7]	{ 4 fields }	Object
[8]	{ 4 fields }	Object
[9]	{ 4 fields }	Object
[10]	{ 4 fields }	Object
[11]	{ 4 fields }	Object
[12]	{ 4 fields }	Object
[13]	{ 4 fields }	Object
[14]	{ 4 fields }	Object
[15]	{ 4 fields }	Object
[16]	{ 4 fields }	Object
[17]	{ 4 fields }	Object
[18]	{ 4 fields }	Object
[19]	{ 4 fields }	Object
[20]	{ 4 fields }	Object
[21]	{ 4 fields }	Object
[22]	{ 4 fields }	Object
[23]	{ 4 fields }	Object
[24]	{ 4 fields }	Object
[25]	{ 4 fields }	Object
[26]	{ 4 fields }	Object
[27]	{ 4 fields }	Object
[28]	{ 4 fields }	Object
id	1	Int32
name	Normal User	String
created_at	2021-03-16 16:10:20.944Z	Date
updated_at	2021-05-20 15:10:53.908Z	Date
__v	0	Int32

Gambar 3.10. Skema *Collection roles*

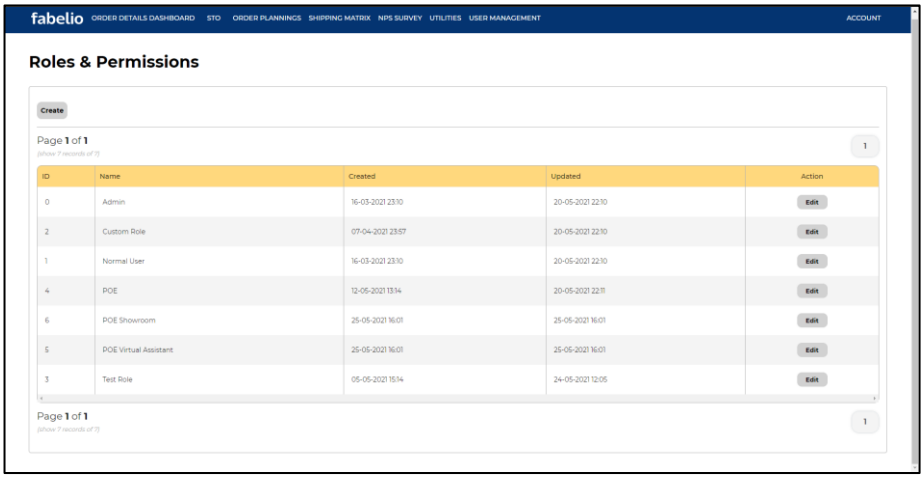
Ketika pengguna melakukan *login*, data-data dari kedua *collection* tersebut akan diterima oleh dua pihak, yaitu Redux dan JWT. Redux merupakan *library* pengelolaan *State* dari ReactJS. *State* ditujukan agar komponen *front-end* dari ReactJS dapat "menyimpan" atau "mengingat" data yang diregistrasikan pada *State* tersebut, sehingga kita tidak perlu melakukan penarikan data ke *database* berkali-kali untuk data yang muncul pada sisi

front-end secara berulang-ulang. JWT, singkatan dari JSON *Web Token*, merupakan metode yang umum digunakan pada pengembangan *web service*, di mana JWT berfungsi untuk melakukan enkripsi suatu *payload* data yang akan digunakan untuk *web service* itu sendiri. Token ini dapat digunakan pada sisi *back-end*, dalam hal ini digunakan sebagai kredensial ketika pengguna mengakses suatu halaman yang perlu melakukan *request* HTTP kepada API.

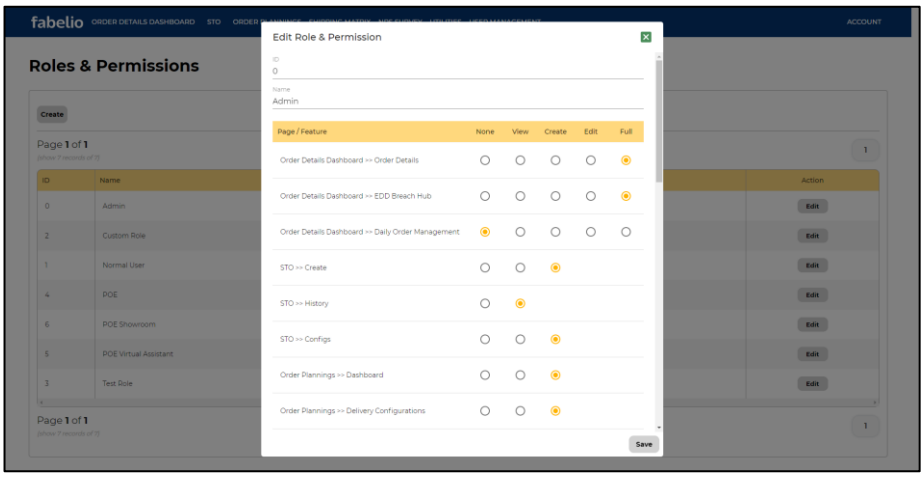
Ketika hal-hal di atas sudah dilakukan, hal tersisa yang perlu dilakukan adalah mengimplementasi di sisi *front-end* untuk setiap fungsi yang diinginkan berdasarkan *State* dan *payload* JWT yang sudah dimiliki. Hal-hal di atas yang dipersiapkan mahasiswa sebelum mengimplementasi fitur IAM lebih lanjut.

4. Mengimplementasi Perizinan Berdasarkan Peran Pengguna ke Setiap Komponen Halaman Marvel

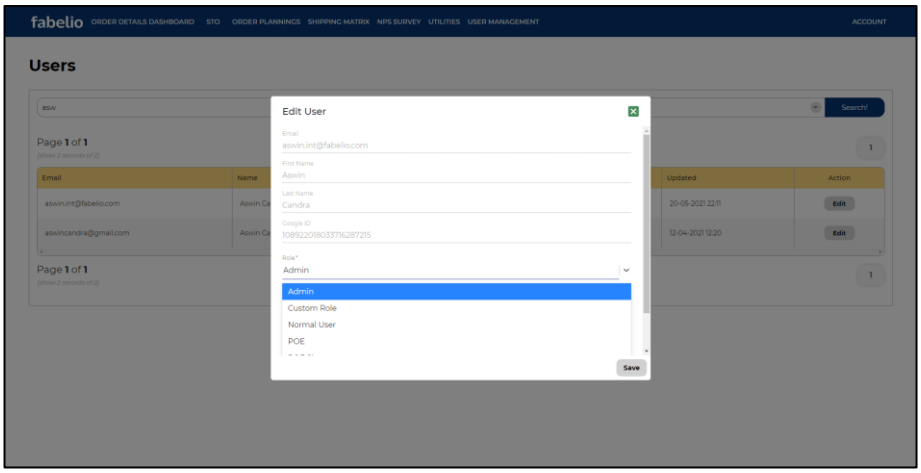
Sebelumnya, Marvel hanya memiliki dua peran pengguna, yaitu Admin dan pengguna normal. Proyek *Marvel Role* memungkinkan Admin Marvel dapat membuat dan memperbaharui *role* serta memasangkan *role* tersebut ke setiap pengguna yang sesuai. Fitur-fitur ini ditunjukkan pada Gambar 3.11 s.d. Gambar 3.13.



Gambar 3.11. Tampilan Halaman Peran Pengguna pada Marvel



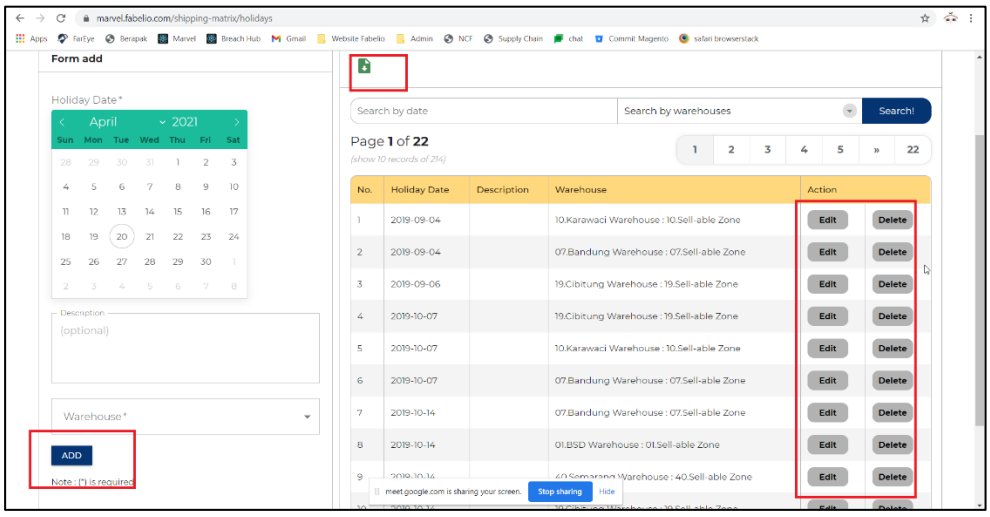
Gambar 3.12. Fitur Pembuatan atau Pembaharuan *Role* pada Marvel



Gambar 3.13. Halaman Daftar Pengguna Marvel dan Fitur Pemberian *Role*

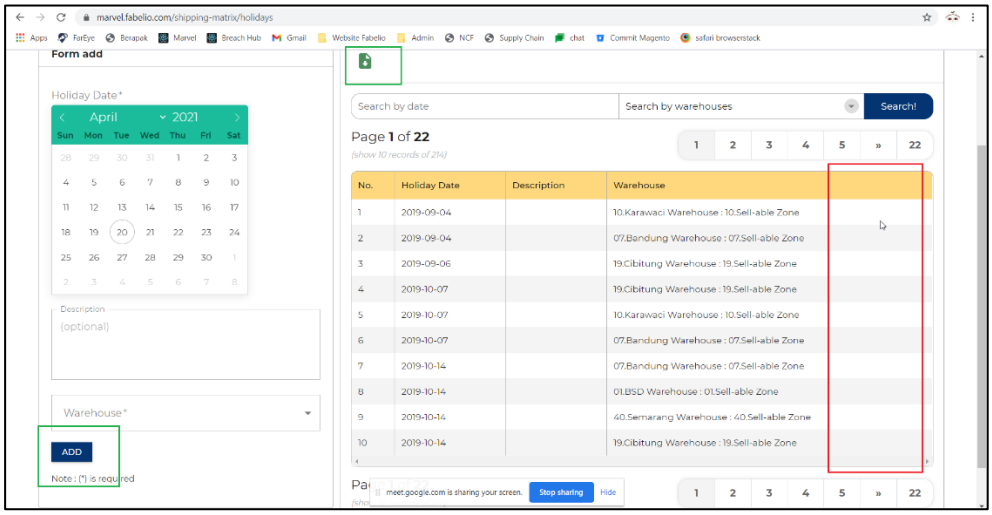
Pada Gambar 3.12, kolom pertama menunjukkan daftar seluruh fitur yang membutuhkan konfigurasi perizinan. Pada kolom-kolom berikutnya, terdapat *radio button* untuk menetapkan tingkat perizinan *role* tersebut. Semakin ke kanan (level izin "*Full*"), artinya pengguna dapat mengakses fitur tersebut secara penuh. Setiap fitur memiliki tingkat izin maksimal yang berbeda-beda, dan setiap tingkatnya memiliki spesifikasi izinnya masing-masing, dan semuanya sudah tertera pada *Product Requirement Document* yang disusun oleh Tim Produk.

Sebagai contoh, Gambar 3.14 s.d. 3.17 menunjukkan implementasi IAM pada halaman *Shipping Matrix/Holidays* di Marvel UI. Tingkat izin maksimal dari halaman ini adalah *Edit*. Gambar 3.14 menunjukkan tampilan halaman *Shipping Matrix/Holidays* bagi pengguna dengan peran yang memiliki izin *Edit*, di mana pengguna memiliki akses untuk melihat, mencari, menambah, mengubah, menghapus, dan mengunduh daftar hari libur gudang/*showroom*.

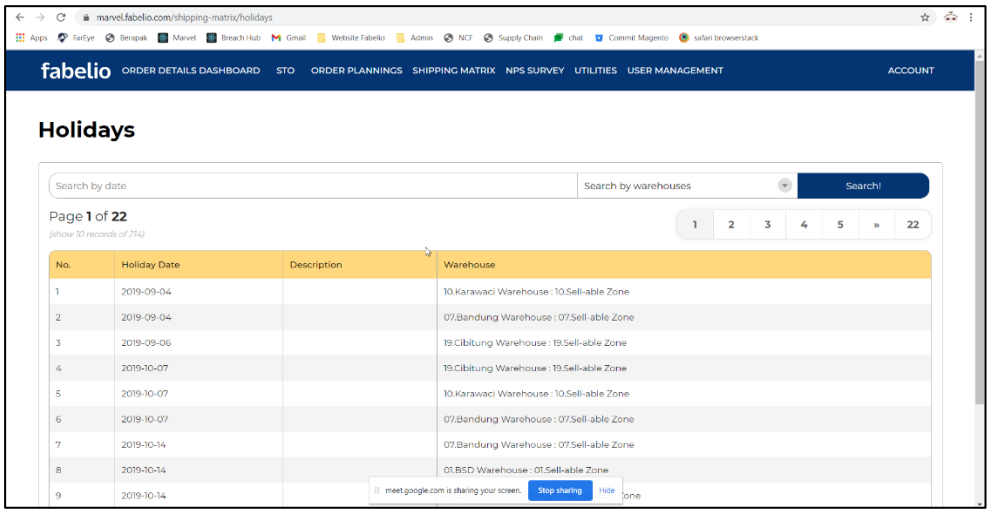


Gambar 3.14. Implementasi Proyek *Marvel Role* dengan Tingkat Izin Pengguna: *Edit*

Gambar 3.15 menunjukkan tampilan halaman *Shipping Matrix/Holidays* bagi pengguna dengan peran yang memiliki izin *Create*, di mana pengguna tetap bisa melihat, mencari, menambah, dan mengunduh hari libur, tetapi tidak bisa melakukan perubahan atau penghapusan hari libur gudang/*showroom*.



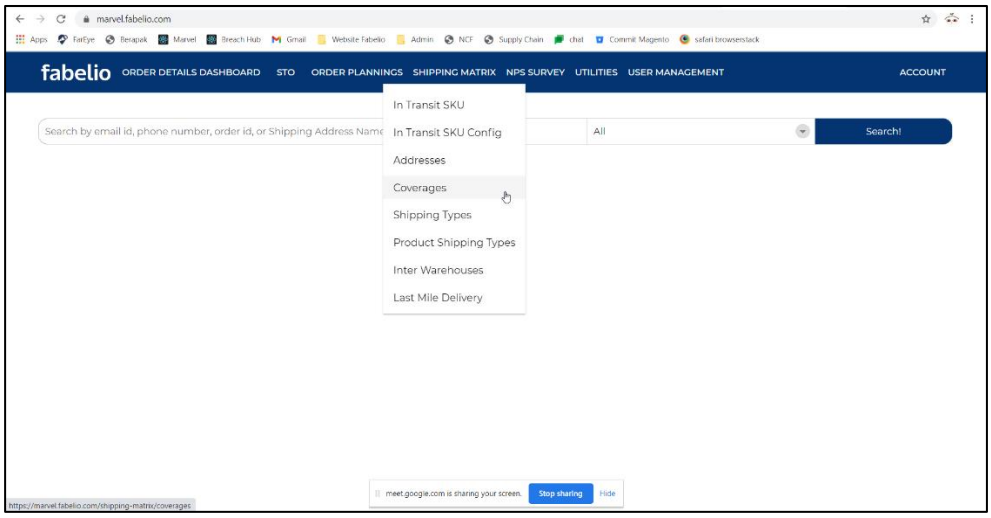
Gambar 3.15. Implementasi Proyek *Marvel Role* dengan Tingkat Izin Pengguna: *Create*



Gambar 3.16. Implementasi Proyek *Marvel Role* dengan Tingkat Izin Pengguna: *View*

Gambar 3.16 menunjukkan tampilan halaman *Shipping Matrix/Holidays* bagi pengguna dengan peran yang memiliki izin *View*, di mana pengguna hanya dapat melihat dan melakukan pencarian daftar hari libur gudang/*showroom*.

Gambar 3.17 menunjukkan kondisi ketika pengguna masuk sebagai *role* yang memiliki tingkat izin *None* pada halaman *Shipping Matrix/Holidays*, atau sama saja dengan tidak memiliki izin pada halaman tersebut. Pengguna tidak akan menemukan sub-menu Holidays pada Marvel UI yang seharusnya terletak di bawah sub-menu Shipping Types.

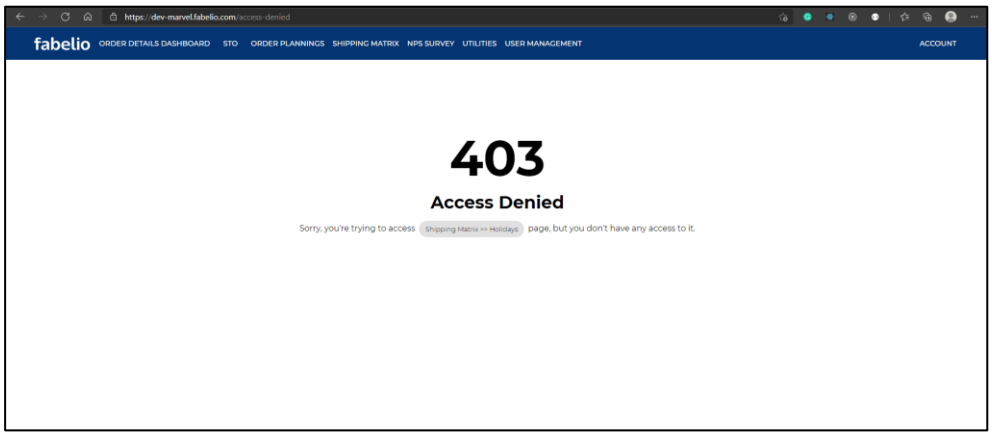


Gambar 3.17. Implementasi Proyek *Marvel Role* dengan Tingkat Izin Pengguna: *None*

5. Implementasi Halaman 403 pada Marvel UI

Untuk meningkatkan keamanan, mahasiswa juga membuat halaman baru berupa Halaman 403, yaitu halaman yang akan dituju ketika pengguna berusaha mengakses halaman tertentu melalui URL padahal pengguna tidak

memiliki izin ke halaman tersebut. Halaman 403 tersebut untuk halaman *Shipping Matrix/Holidays* dapat dilihat pada Gambar 3.18.



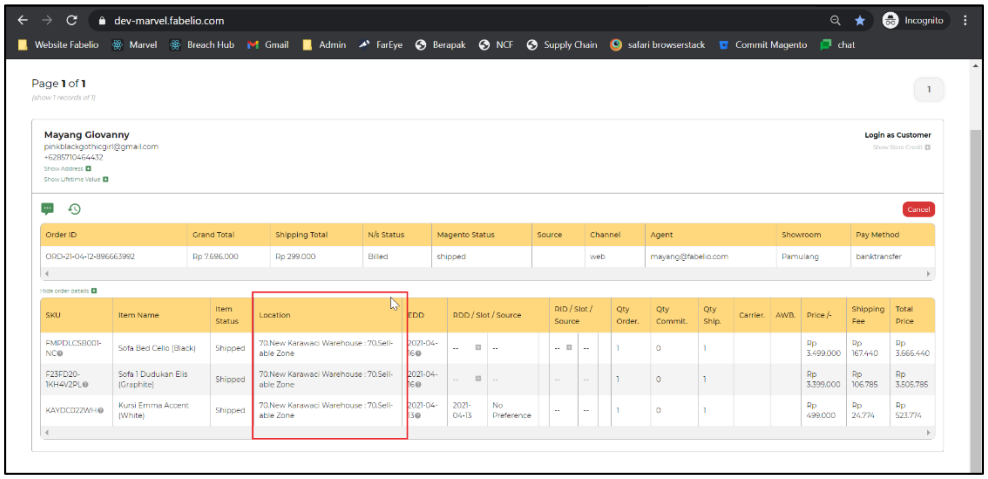
Gambar 3.18. Contoh Tampilan Halaman 403 untuk Pembatasan Akses Halaman

3.3.3. Pengerjaan Proyek "*Order Splitting*"

1. Menambah Atribut "*Location*" di Tingkat Produk pada Marvel

Pada proyek ini, salah satu pekerjaan yang ditugaskan pada mahasiswa adalah untuk memindahkan data lokasi *fulfillment* dari yang tadinya berada di tingkat pesanan menjadi ke tingkat produk yang dipesan. Sebelumnya, satu pesanan hanya memiliki satu lokasi *fulfillment*, yaitu gudang atau *showroom* yang terdekat dari lokasi konsumen, tanpa mempertimbangan ketersediaaan stok produk di lokasi *fulfillment* tersebut. Hal ini mengakibatkan peningkatan *lead time*, bahkan sering terjadi bahwa barang tidak sampai sesuai dengan estimasi waktu yang diberikan. Hal ini dikarenakan harusnya dilakukan perpindahan stok (*Stock Transfer*) dari satu lokasi *fulfillment* ke lokasi lainnya.

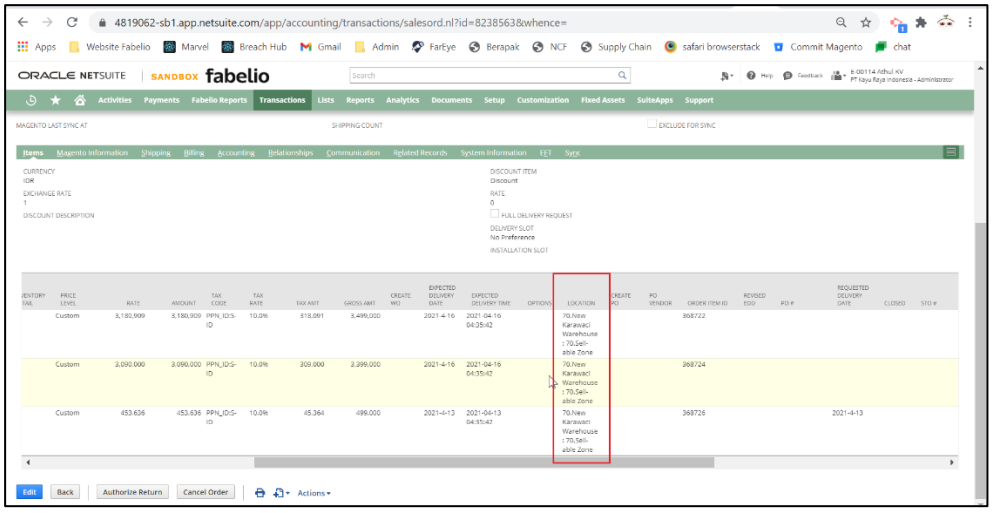
Untuk melaksanakan tugas tersebut, mahasiswa perlu mendefinisikan kembali skema dari *collection* `orders` pada MongoDB, *collection* yang menampung data seluruh pesanan konsumen, sehingga terdapat atribut baru bernama `location` di setiap produk yang dipesan. Ketika pesanan dibuat, maka proses *back-end* akan melakukan kalkulasi jarak antara lokasi *fulfillment* dengan lokasi konsumen. Tetapi tidak sampai di situ, sistem Marvel akan memanggil *service* lain bernama Berapak, *microservice* yang juga dikembangkan Fabelio secara internal untuk melakukan pengecekan stok. Ketika terdapat stok di lokasi tersebut, maka lokasi tersebut yang akan menjadi lokasi *fulfillment* atas produk tersebut. Ketika stok tidak ditemukan, maka akan dilakukan pencarian kembali ke lokasi lainnya. Atribut "*location*" yang sudah berada di tingkat produk kemudian juga ditampilkan pada Marvel UI seperti yang ditunjukkan pada Gambar 3.19.



Gambar 3.19. Implementasi *Order Splitting* pada Marvel

2. Menambah Atribut "*Location*" di Tingkat Produk pada Netsuite

Perlu dipastikan juga bahwa Marvel mengirimkan data lokasi *fulfillment* yang sudah berada di tingkat produk ke Netsuite, sehingga di Netsuite pun dapat ditambahkan kolom baru, yaitu kolom lokasi pada tingkat produk. Gambar 3.20 menunjukkan hasil implementasi *Order Splitting* dengan memindahkan data lokasi *fulfillment* ke tingkat produk pada Netsuite.



Gambar 3.20. Implementasi *Order Splitting* pada Netsuite

3. Menambah Atribut "*Carrier*" pada Opsi Pengiriman

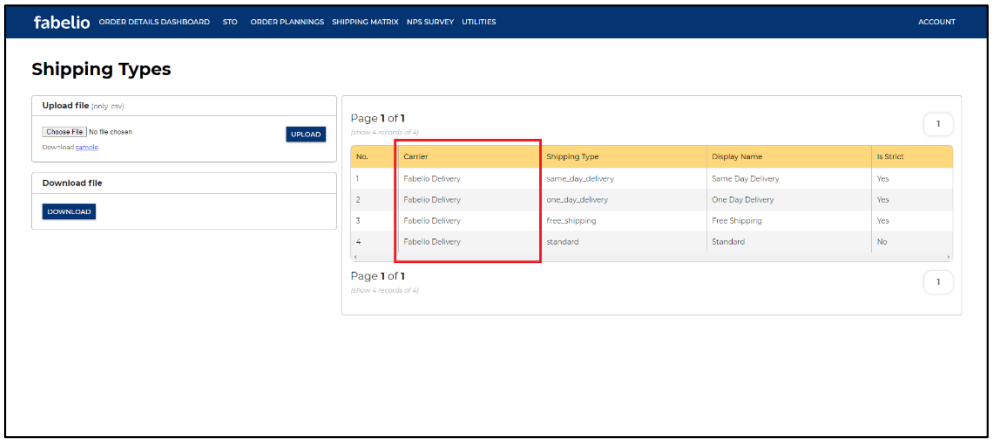
Tugas lain yang diberikan untuk mendukung proyek ini adalah menambahkan kolom baru pada metode pengiriman, yaitu "*Carrier*". Sebelumnya, opsi pengiriman yang dimiliki oleh Fabelio hanyalah pengiriman milik Fabelio, hasil kerja sama Fabelio dengan salah satu perusahaan 3PL, yaitu Fareye. Metode pengiriman ini memiliki beberapa varian lagi, seperti pengiriman standar, satu hari sampai, dan lain-lain. Ke depannya, Fabelio ingin menambahkan metode pengiriman lain dengan

berbagai variasi jenis pengiriman juga, sehingga dibutuhkan satu kolom baru yang selanjutnya disebut "*Carrier*" untuk menampung metode pengiriman atau penyedia layanan pengiriman. Kali ini mahasiswa bekerja dengan *database* PostgreSQL dan *microservice* internal yang bernama Morbius.

Pada praktiknya, *developer* tidak diperbolehkan untuk melakukan perubahan pada *database* langsung menggunakan SQL, khususnya pada *database* di *environment production*. *Developer* akan menggunakan yang dinamakan *migration script* untuk manipulasi skema tabel dan *seeder script* untuk manipulasi data. Hal ini ditujukan untuk tetap menjaga integritas data, di mana kita tetap dapat melacak histori perubahan demi perubahan dari database yang dimiliki. *Script* tersebut tetap dalam bahasa JavaScript, tetap tersimpan pada *source code*, yang kemudian dieksekusi ke *database* menggunakan *engine* ORM Sequelize[11], *engine* yang dapat memetakan bahasa Pemrograman Berorientasi Objek menjadi sintaks *database query*.

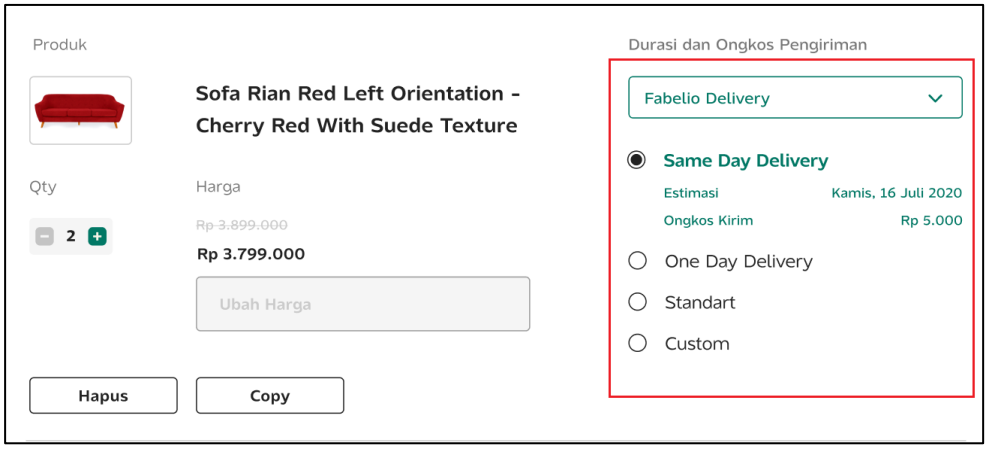
Setelah memanipulasi skema *database*, memperbaharui API dan juga Marvel UI (halaman Shipping Matrix/Shipping Types), maka diperoleh hasil akhir pada Marvel UI yang ditunjukkan pada Gambar 3.21, di mana sudah terdapat kolom baru bernama "*Carrier*". *Carrier* yang ada saat ini, yaitu yang dimiliki oleh Fabelio sendiri, dinamakan *Fabelio Delivery*.

[11] https://sequelize.org/



Gambar 3.21. Implementasi Penambahan Atribut "*Carrier*" pada Jenis Pengiriman di Marvel UI

Data tersebut akan digunakan oleh *microservice* Morbius untuk mengirimkan data metode pengiriman ke halaman *checkout* dari *website* e-commerce fabelio.com. Hal ini ditunjukkan pada Gambar 3.22.



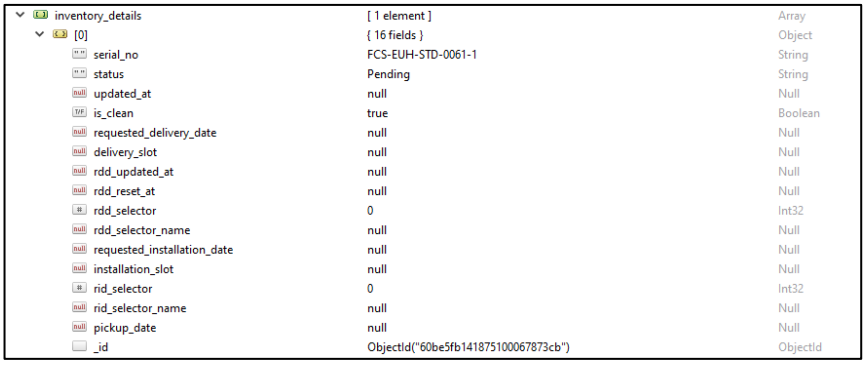
Gambar 3.22. Pemanfaatan Atribut "*Carrier*" untuk Halaman *Checkout* fabelio.com

4. *Quantity Status Splitting*

Terakhir, pada proyek ini mahasiswa diminta untuk melakukan pemisahan unit kuantitas dari setiap produk yang dipesan. Pada kasus tertentu, produk yang dipesan dengan kuantitas lebih dari satu buah dapat

berada dalam tahapan proses yang berbeda antara satu dengan lainnya, terutama pada produk yang berukuran besar atau membutuhkan perakitan. Jika status pesanan suatu produk sebelumnya ada dalam tingkat produk, Tim Produk menginginkan untuk menurunkan status proses ke tingkat kuantitas dari produk. Misalnya, suatu barang X dipesan sebanyak tiga buah. Maka status pesanan dari ketiga kuantitas dari barang X tersebut akan diperlakukan secara terpisah sekalipun dipesan dalam waktu bersamaan. Hal ini ditujukan untuk mendukung transparansi status produk secara lebih granular.

Mahasiswa diminta untuk melakukan penambahan skema dengan menambah satu atribut berupa *array of objects* di dalam masing-masing atribut `items` yang dipesan yang dinamakan `inventory_details` pada *collection* **`orders`**. Jumlah elemen *array* dari atribut `inventory_details` adalah sama dengan jumlah kuantitas yang dipesan untuk barang yang bersangkutan. Masing-masing objek dari `inventory_details` akan memilki kode unik yang dinamakan `serial_no` yang dihasilkan oleh *microservice* Marvel pada saat suatu pesanan terbuat dan akan dikirim ke *database* MongoDB.



Gambar 3.23. Atribut Baru "*inventory_details*" pada Skema *Collection orders*

Gambar 3.23 menunjukkan atribut baru yang telah dibuat bernama `inventory_details` pada *collection* **`orders`**. Atribut baru ini akan nantinya digunakan oleh *engine State Machines*, dalam hal ini memanfaatkan *library* XState[12], yang akan memperbaharui status dari masing-masing unit produk berdasarkan *trigger* dari *state* status atau proses sebelumnya, dan kemudian memberikan *trigger* ke *state* status atau proses selanjutnya.

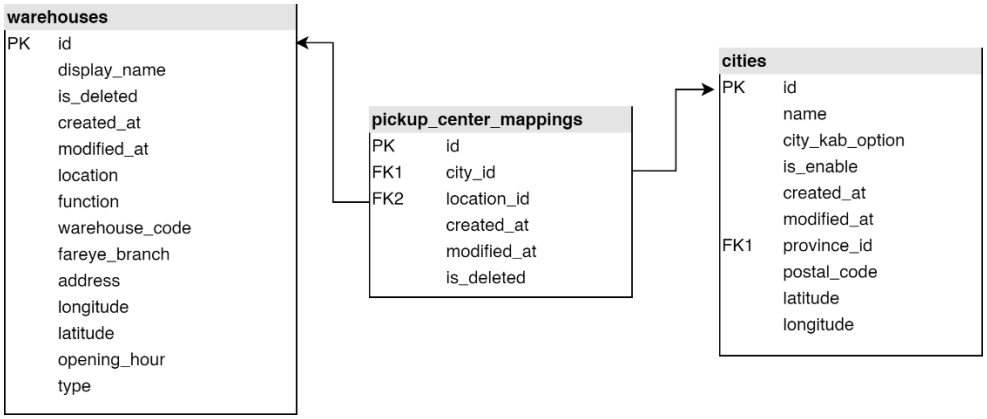
3.3.4. Pengerjaan Proyek "*Pickup at Showroom*"

Dengan rencana untuk meningkatkan *footfall* (jumlah kunjungan), *Average Order Value* (AOV), dan biaya per *footfall* untuk *showroom*, Tim Operasional menginisiasi strategi yang memungkinkan pilihan *Pickup at Showroom* pada *website* fabelio.com. Pada saat *checkout*, konsumen akan memiliki pilihan pengambilan barang sendiri di *showroom* terdekat dari lokasinya.

1. Pembuatan Modul *Pickup Center Mapping*

Modul ini memungkinkan tim operasional untuk melakukan konfigurasi pemetaan lokasi *pickup* terdekat berdasarkan setiap kota yang ada di Indonesia. Pada tugas ini, Mahasiswa membuat tabel baru bernama **`pickup_center_mappings`** pada *database* PostgreSQL. Skema dari tabel **`pickup_center_mappings`** dapat dilihat pada Gambar 3.24.

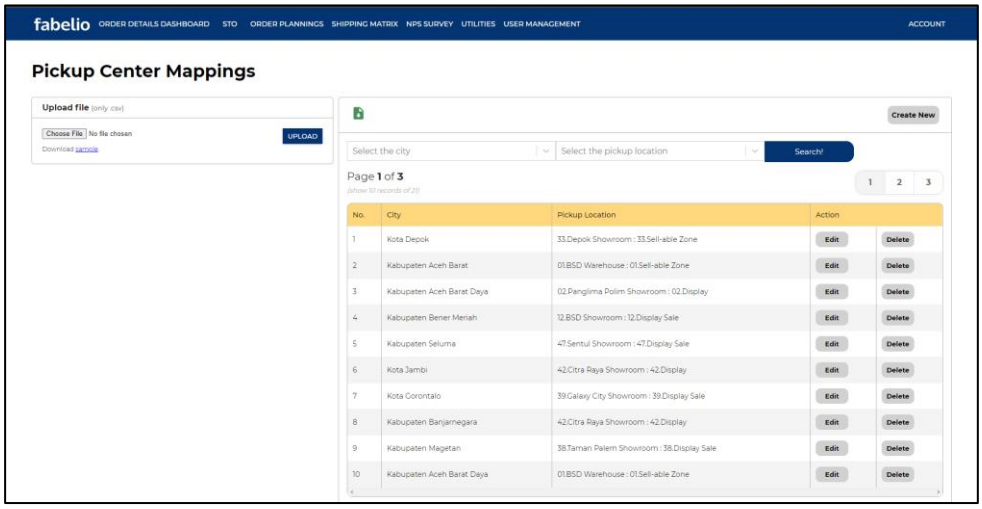
[12] https://xstate.js.org/



Gambar 3.24. Skema Tabel *Pickup Center Mappings*

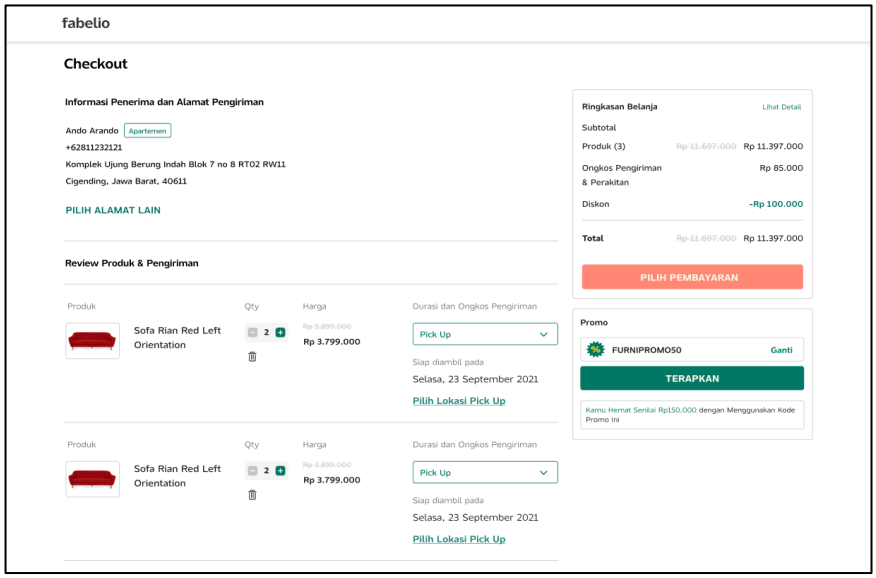
Dapat dilihat pada Gambar 3.24 bahwa tabel `pickup_center_mappings` terhubung dengan dua tabel lainnya yang sudah tersedia, yaitu tabel `cities` menggunakan *Foreign Key* `city_id` dan tabel `warehouses` menggunakan *Foreign Key* `location_id`. Tabel `pickup_center_mappings` mengambil kolom nama kota (`name`) dan jenis pemerintahan (`city_kab_option`) dari tabel `cities`, serta nama lokasi *fulfillment* (`display_name`) dari tabel `warehouses`.

Setelah itu, dibuat juga API pada *microservice* Morbius untuk dapat memperoleh, menambah, memperbaharui, menghapus, *import* CSV, dan *export* CSV data dari *Pickup Center Mapping* ini. API ini yang kemudian akan digunakan oleh Marvel UI sehingga fitur-fitur tersebut dapat digunakan langsung oleh *end-users*. Tampilan halaman *Pickup Center Mappings* pada Marvel dapat dilihat pada Gambar 3.25.

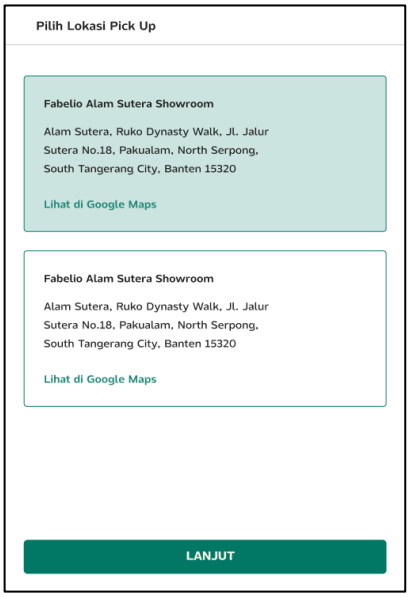


Gambar 3.25. Halaman Konfirguasi *Pickup Center Mappings* pada Marvel

Hasil konfigurasi *Pickup Center Mapping* ini akan dikirim oleh *microservice* Morbius untuk diimplementasi pada *website* fabelio.com oleh Tim *Front-end Engineer*, di mana akan disajikan pilihan "*Pickup*" bagi konsumen. Pilihan *showroom* akan disesuaikan dengan konfigurasi *Pickup Center Mappings* di Marvel. Implementasi pada *website* fabelio.com dapat dilihat pada Gambar 3.24 dan Gambar 3.25.



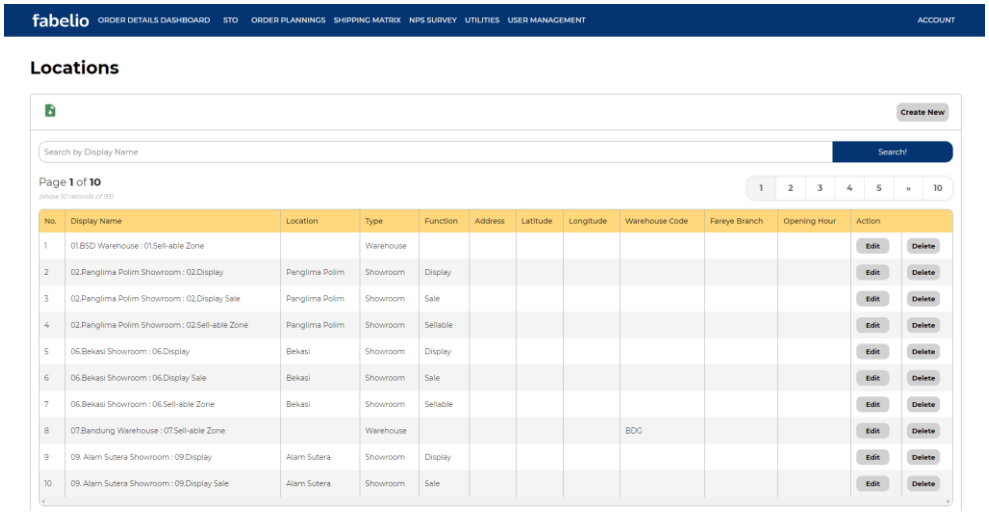
Gambar 3.26. Halaman *Checkout* Hasil Implementasi *Pickup at Showroom*



Gambar 3.27. Pilihan Lokasi *Pickup* Hasil Implementasi *Pickup at Showroom*

2. Penambahan Atribut Baru untuk Data Lokasi *Fulfillment*

Tugas ini hampir sama dengan tugas pembuatan modul *Pickup Center Mappings* sebelumnya, tetapi untuk lokasi *fulfillment* atau *pickup*, tabelnya sudah tersedia dengan nama `warehouses`. Mahasiswa diminta menambahkan beberapa kolom baru dari skema yang sudah ada, seperti `address` (alamat), `longitude` (garis bujur), `latitude` (garis lintang), `opening_hour` (jam buka), dan `type` (*warehouse* atau *showroom*). Mahasiswa juga diminta untuk menambahkan fungsi pada API yang sebelumnya hanya terdapat fungsi untuk memperoleh data, ditambah menjadi dapat menambah data baru, memperbaharui, menghapus, dan *export* data lokasi dalam unduhan file CSV. Setelah seluruh komponen *back-end* sudah siap, mahasiswa diminta membuat halaman konfigurasinya pada Marvel dengan nama "*Locations*". Tampilan halaman konfigurasi lokasi *fulfillment*/*pickup* ditunjukkan pada Gambar 3.28.



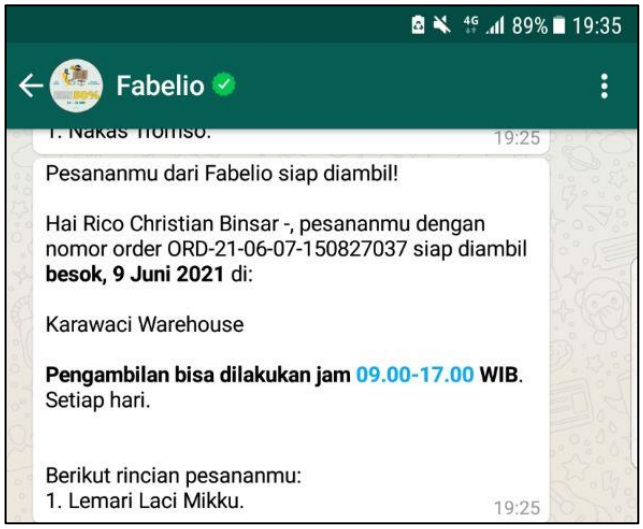
Gambar 3.28. Tampilan Halaman Konfigurasi *Locations* pada Marvel

Tugas ini ditujukan untuk meningkatkan kualitas data dari lokasi *fulfillment* atau *pickup* demi mendukung fitur-fitur yang akan datang, termasuk salah satunya untuk *Pickup Center Mappings*.

3. Pembuatan WhatsApp *Pickup Reminder*

Pada tugas ini, mahasiswa membuat sebuah *cron job*, yaitu sebuah mekanisme pada *web server* untuk menjalankan sebuah *script* secara otomatis setiap kurun waktu tertentu. *Job* atau tugas yang dilakukan dalam hal ini adalah menjalankan *microservice* Marvel untuk melakukan pencarian pada *collection* `orders` terhadap produk-produk yang dipesan dengan metode *pickup*, di mana tanggal *pickup* jatuh pada H+1 dari hari ini. Setelah diperoleh seluruh barang pesanan yang akan diambil langsung oleh konsumen (*pickup*) pada hari esok, seluruh data pesanan tersebut akan ditampung dalam satu *payload* dan dikirim sebagai *topic* ke layanan SNS (*Simple Notification Service*) dari AWS.

Setelah itu, layanan SNS akan mem-*publish topic* tersebut ke *subscriber*-nya, dalam hal ini adalah *microservice* Fabelio WhatsApp. *Microservice* Fabelio WhatsApp akan menghasilkan pesan otomatis berupa pengingat *pickup* ke nomor WhatsApp yang juga sudah terdapat di dalam *payload* yang dikirim. Pesan pengingat tersebut berisi informasi detail mengenai barang apa yang sudah siap untuk diambil, lokasi pengambilan, dan lain-lain. Gambar 3.29 menunjukkan contoh hasil akhir fitur pemberitahuan WhatsApp untuk pengingat H-1 pengambilan produk di *Showroom*.



Gambar 3.29. Pemberitahuan WhatsApp untuk H-1 Pengambilan Produk

3.4. Kendala dan Solusi

3.4.1. Kendala

Selama menjalankan praktik kerja magang sebagai *Software Developer Intern* di Fabelio, mahasiswa menghadapi beberapa rintangan sekaligus sebagai pembelajaran, yaitu:

1. Sempat terjadi suatu isu pada saat fitur yang mahasiswa kembangkan sudah masuk tahap pengujian yang memerlukan pemesanan produk di *website staging*. Ternyata produk yang dipesan adalah produk yang juga *live* di *website production*. Sehingga, pesanan yang harusnya hanya menjadi pesanan fiktif malah menjadi pesanan yang benar-benar terbuat. Pesanan sudah sampai ke pengrajin, sudah diproduksi, hingga sudah siap dikirim ke gudang.
2. *Bugs* yang tidak terduga ketika fitur yang dikembangkan sudah sampai tahap pengujian. Hal ini cukup menguras waktu karena setelah perbaikkan *bug*, kode harus diulas kembali oleh *developer* lain, kemudian diuji kembali oleh tim QA. Hal tersebut juga menyebabkan penilaian *developer* menjadi kurang baik karena *progress* pekerjaan yang maju-mundur.

3.4.2. Solusi

Berdasarkan kendala di atas, berikut solusi yang diambil mahasiswa untuk mengatasi kendala yang ada:

1. Mengajukan rekomendasi untuk pemisahan katalog produk antara *website staging* dan *website production*, atau setidaknya memberikan daftar produk yang tidak *live* di *website production*.
2. Memperkaya *Unit Test* dengan seluruh kemungkinan kejadian yang ada. Pengendalian kemungkinan *error* dalam kode juga dibutuhkan,

sehingga tetap ada proses yang dijalankan sesuai dengan anomali yang sudah didefinisikan, setidaknya berupa *log*.