BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Kedudukan dalam perusahaan selama menjalani proses kerja magang adalah sebagai *Backend Engineer Intern* pada tim *Payment* di divisi *Tech Platform* dengan mentor atau supervisi yaitu Bapak Dekadwinavinta Candranugraha yang memiliki jabatan sebagai *Technical Lead* dari tim Payment. Selama menjalani proses kerja magang, seluruh tugas diberikan oleh mentor dan melapor kembali ke mentor apabila ada kendala atau tugas sudah selesai.

Didalam tim *Payment*, terdapat beberapa *job desk* berbeda yang saling berkaitan untuk mengembangkan fitur *Payment* pada tiket.com, yaitu:

- *Technical Lead* yang bertugas sebagai pemimpin pada tim dan memberikan arahan terhadap tugas-tugas yang sedang dikerjakan oleh setiap anggota tim.
- *Scrum Master* yang bertugas mengatur *development life cycle* dalam tim.
- *Product Manager* bertugas mengatur tugas-tugas yang akan dikerjakan oleh tim dalam suatu *sprint* dan membuat prioritas tentang tugas-tugas yang harus segera diselesaikan. *Product Manager* juga bertugas menjembatani antara tim *developer* dan tim *business* maupun UI/UX.
- *Quality Assurance* bertugas menguji program yang telah dikembangkan untuk menemukan kesalahan atau bug pada program agar program dapat berjalan tanpa masalah berarti.
- *Backend Engineer* bertugas mengelola data pada basis data, membuat API ataupun *endpoint* untuk diakses oleh *Frontend Engineer*.

- *Frontend Engineer* bertugas mengolah data dari restAPI yang telah dibuat oleh *Backend Engineer* untuk mengembangkan fitur *Payment* sesuai dengan rancangan antarmuka yang telah dibuat

Manajemen pengembangan piranti lunak (SDLC) yang digunakan oleh tiket.com adalah *agile scrum*. *Scrum* adalah sebuah framework atau kerangka kerja yang bisa digunakan untuk proyek yang memiliki skala besar seperti tiket.com. Dalam metode *scrum*, seorang *Product Manager* akan melakukan *listing* terhadap seluruh tugas-tugas yang akan diberikan pada tim *Payment* dalam sebuah *backlog*. Setiap 2 minggu sekali akan dilakukan *Sprint Planning*, yaitu sebuah rapat untuk menentukan tugas-tugas apa yang akan diambil oleh *developer* dari *backlog* yang telah dibuat sebelumnya. Setiap tugas memiliki bobot tertentu yang akan digunakan untuk mengevaluasi kerja tim.

Rapat koordinasi lainnya yang dilakukan oleh tim *Payment* adalah *weekly sync-up* dan *Standup Meeting*. *Weekly sync-up* merupakan rapat mingguan yang dilakukan untuk mengetahui progress maupun kendala apa saja yang dialami oleh tim dalam menjalani tugas dari *sprint planning* sebelumnya. Untuk tim *Payment*, *weekly sync-up* diadakan rutin setiap hari rabu. Pada *weekly sync-up* juga dilakukan diskusi tentang bagaimana performa tim sejauh ini dan apa saja yang perlu di-*improve* kedepannya.

Standup Meeting merupakan rapat harian yang dilakukan oleh seluruh anggota tim *Payment*. *Standup Meeting* pada tim *Payment* diadakan setiap pukul 10:30 WIB. Rapat ini diadakan untuk membahas tentang apa saja yang telah dilakukan oleh setiap anggota tim pada hari sebelumnya dan apa yang akan dilakukan hari ini. Tujuan diadakannya rapat harian ini adalah untuk melihat

progres harian tim dan kendala apa yang ditemukan pada kemarin agar bisa diselesaikan hari ini.

Setiap anggota tim menggunakan *git workflow* saat menyelesaikan tugas-tugas yang telah diberikan saat *sprint planning*. Saat memulai tugas baru, anggota tim akan membuat *branch* baru dari *repository* yang dikerjakan. Setelah tugas selesai, anggota tim melakukan *push* ke *branch* baru yang telah dibuat sebelumnya. Selanjutnya, dilakukan *pull request* untuk menggabungkan *branch* baru tersebut dengan *branch development*. Lalu *senior engineer* dan *technical lead* akan melakukan *review* terhadap tugas tersebut. Apabila disetujui, maka tugas baru tersebut selesai dan *branch* tersebut akan digabungkan dengan *branch development*.

3.2 Tugas yang dilakukan

Selama menjalani proses kerja magang di PT Global Tiket Network sebagai *Backend Engineer Intern*, tugas-tugas yang diberikan adalah tugas-tugas yang berhubungan dengan ranah tim *Payment*, yaitu bagian pembayaran. Tugas-tugas tersebut diberikan melalui *Sprint Planning* ataupun diberikan langsung oleh mentor atau supervisi. Bagian *Backend* pada tim *Payment* menggunakan bahasa pemrograman Java dengan framework Spring Boot. Selain itu, digunakan juga Redis yang berguna untuk *caching* dan kafka yang merupakan sebuah *message broker* untuk melakukan komunikasi antar *microservices*. *Database* yang digunakan adalah mongoDB dengan noSQL. NoSQL adalah sistem *database* non-relasional, dimana data disimpan berdasarkan *key* dan *value*. Setiap tugas yang telah diselesaikan akan dilakukan pengecekan atau *code review* melalui github. Tugas-tugas yang dilakukan meliputi pembuatan *controller* restAPI, integrasi metode

pembayaran baru, memperbaiki CRUD yang memiliki *error*, integrasi antar *microservices*, dan penambahan *variable* untuk validasi email, nomor *handphone*, dan *apps version*.

3.3 Uraian Pelaksanaan Kerja Magang

3.3.1 Proses Pelaksanaan

Dalam menjalani proses kerja magang, realisasi kerja magang dapat dilihat pada Tabel 3.1 berikut.

Tabel 3.1 Realisasi Kerja Magang

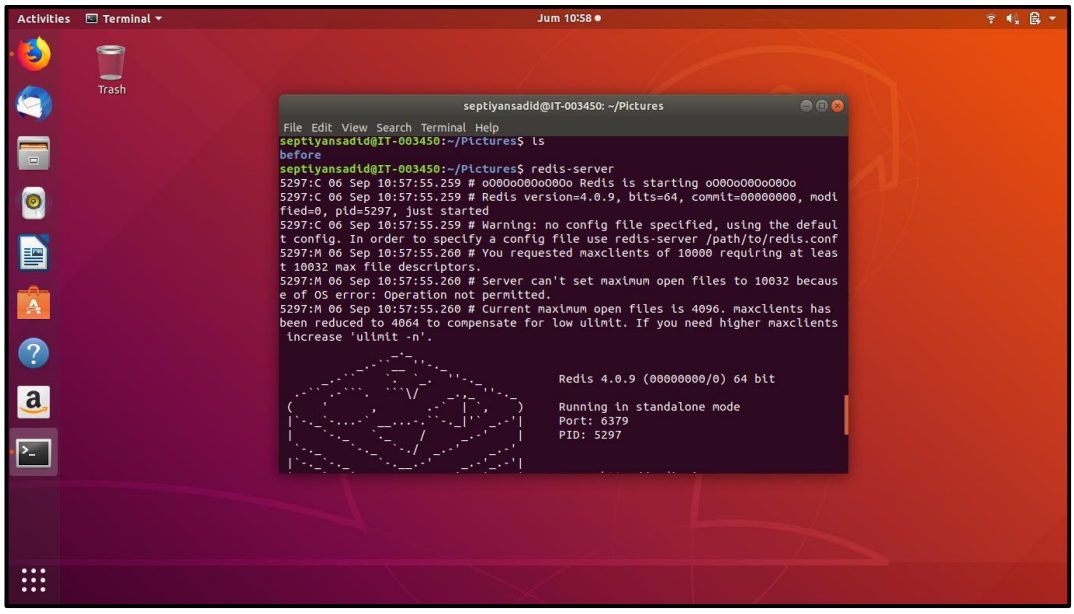
Minggu ke-	Pekerjaan yang dilakukan
1	• Mempelajari *framework* Java Spring Boot dan tools lain yang digunakan • Membuat aplikasi simple yaitu sebuah *controller* rest API
2	• Membuat CRUD untuk *table* atau *collection* Variable Source pada *microservice* Promo Code
3	• Memperbaiki CRUD pada *table* atau *collection* System Parameter di *microservice* Credit Card dan Payment Core
4	• Melanjutkan pekerjaan pada minggu ke-3 • Menambahkan enum pada currency *controller*
5	• Melakukan integrasi metode pembayaran baru yaitu pay at hotel pada *microservice* Order Aggregate
6	• Melakukan integrasi notifikasi pembayaran ke *microservice* Whatsapp
7	• Menambahkan *variable* validasi email dan *phone number* pada *microservice* Promo Code
8	• Menambahkan *variable* validasi *apps version* untuk android dan IOS pada *microservice* Promo Code
9	• Membuat *service* metode pembayaran baru yaitu akulaku pada m*icroservice* Payment Core

Tabel 3.1 Realisasi Kerja Magang (lanjutan)

Minggu ke-	Pekerjaan yang dilakukan
10	• Membuat *data migration* untuk akulaku pada *microservice* Payment Core
11	• *Standby support* akulaku
12	• *Standby support* akulaku

3.3.1.1 Mempelajari *framework* dan *tools* yang digunakan.

Pada awal proses kerja magang, tugas yang pertama diberikan adalah untuk mempelajari bahasa pemrograman serta *framework* dan *tools* lain yang digunakan oleh tim Payment. Sistem operasi yang digunakan adalah Ubuntu dengan versi 18.04 LTS. Ubuntu digunakan karena memiliki *environment* yang mudah digunakan oleh *developer* untuk menggunakan *threading dan caching.* Bahasa pemrograman yang digunakan adalah Java dengan *framework* Spring Boot. Proses pembelajaran dilakukan dengan membaca dokumentasi serta menonton video tutorial yang menjelaskan tentang topik-topik tersebut. Setelah mempelajari bahasa pemrograman Java dan framework Spring Boot, dilanjutkan dengan membuat controller CRUD sederhana dengan menggunakan Java Spring Boot tersebut. Selanjutnya dilakukan pembelajaran tentang metode *caching* dengan menggunakan redis-server, sehingga setiap proses yang berurusan dengan *database* tidak akan langsung mengakses *database* melainkan mengecek terlebih dahulu kedalam *cache*. Tampilan saat menjalankan redis-server dapat dilihat pada Gambar 3.1.



Gambar 3.1 Menjalankan redis server

3.3.1.2 Membuat CRUD pada *collection* Variable Source di Microservice Promocode

Tugas selanjutnya yang diberikan adalah membuat CRUD Controller pada tabel Variable Source di *microservice* Promocode. *Microservice* Promocode merupakan *microservice* yang mengelola modul Promo Code. Sedangkan Variable Source merupakan *variable* yang digunakan untuk membuat rules pada suatu Promo Code. Contohnya apabila suatu promo code ingin membuat suatu aturan dimana promo tersebut hanya berlaku pada penerbangan, maka akan diambil Variable Source yang berasal dari penerbangan. Data pada *collection* Variable Source dapat dilihat pada Tabel 3.2.

Tabel 3.2 Struktur tabel atau *collection* Variable Source

No	Kolom	Tipe Data	Keterangan
1	Id	objectId	Id di auto generate oleh mongoDB

Tabel 3.2 Struktur Tabel atau collection Variable Source (lanjutan)

No	Kolom	Tipe Data	Keterangan
2	Label	String	Merupakan sebuah label untuk suatu Variable Source
3	Value	String	Merupakan keterangan lengkap dari label
4	createdBy	String	Menyimpan siapa yang telah membuat data ini.
5	createdDate	Date	Merupakan jenjang waktu saat data dibuat
6	updatedDate	Date	Merupakan jenjang waktu saat ada perubahan pada data
7	isDeleted	Boolean	Sebuah flag untuk mengetahui apakah data ini sudah di hapus atau belum

Tahapan selanjutnya adalah melakukan pembuatan *endpoint* untuk *controller* CRUD pada *Variable Source*. *Endpoint* yang dibuat memiliki 4 *method*, yaitu GET, POST, PUT, dan DELETE.

Tabel 3.3 Parameter header

Name	Type	Required
storeId	String	True
channelId	String	True
requestId	String	True
serviceId	String	True
username	String	True

Untuk mengakses API yang dibuat, diperlukan *parameter header* seperti pada Tabel 3.3. *Parameter header* tersebut digunakan untuk mengakses seluruh tugas yang dilakukan.

Tabel 3.4 Parameter *body* untuk *method* GET

Name	Type	Required
Id	Path/String	True

Method GET pada *endpoint* CRUD *Variable Source* digunakan untuk mengambil data berdasarkan id. *Method* tersebut membutuhkan parameter seperti yang ditunjukkan pada Tabel 3.4. Yang dimaksud dengan *path/string* adalah *variable* tersebut berada pada *path* url atau *endpoint* dan memiliki tipe data *string*.

Tabel 3.5 Parameter body untuk *method* POST

Name	Type	Required
Label	String	True
Value	String	True

Method POST pada endpoint CRUD *Variable Source* digunakan untuk menambahkan data baru ke dalam collection *Variable Source*. *Method* tersebut membutuhkan parameter seperti yang ditunjukkan pada Tabel 3.5.

Tabel 3.6 Parameter body untuk method PUT

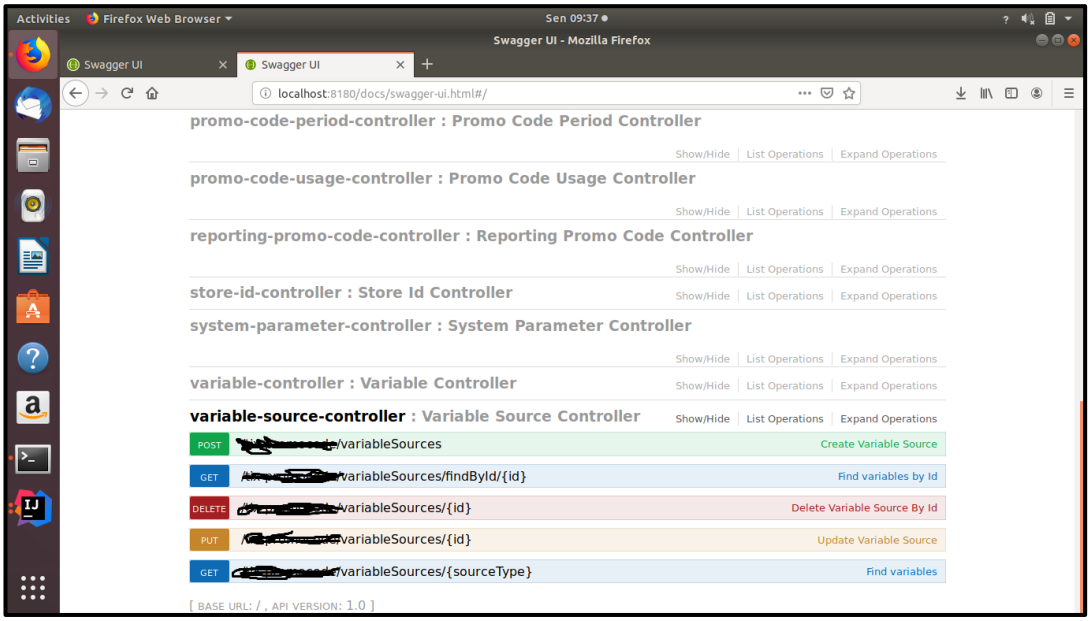
Name	Type	Required
Id	Path / String	True
Label	String	True
Value	String	True

Method PUT pada *endpoint* CRUD *Variable Source* digunakan untuk mengubah data yang telah ada dalam *collection Variable Source*. *Method* ini membutuhkan *path variable* yang berupa string yaitu Id dan isi berupa label dan value seperti yang ditunjukkan pada Tabel 3.6.

Tabel 3.7 Parameter body untuk method DELETE

Name	Type	Required
Id	Path/ String	True

Method DELETE Pada Endpoint CRUD *Variable Source* digunakan untuk melakukan *soft delete* pada data yang telah ada dalam *collection Variable Source*. *Soft delete* merupakan metode melakukan *delete* dengan menggunakan *flag*, dimana data yang asli tidak dihapus dari *database*. *Method* ini membutuhkan *path variable* berupa id untuk menghapus data yang diinginkan seperti yang ditunjukkan pada table 3.7.



Gambar 3.2 Tampilan controller Variable Source pada dashboard

Berikut merupakan tampilan *dashboard* API dari API yang telah dibuat. Gambar 3.2 menunjukkan *method* apa saja yang tersedia pada API yang telah dibuat.

3.3.1.3 **Memperbaiki CRUD pada *collection* System Parameter di *microservice* Creditcard dan Payment Core**.

Tugas selanjutnya yang dikerjakan adalah memperbaiki CRUD pada *system parameter* di *Microservice Creditcard* dan *Payment Core*. *System parameter* adalah *collection* yang berisi parameter-parameter utama yang digunakan pada microservice. *Microservice Creditcard* merupakan sebuah *service* yang digunakan untuk menghandle seluruh transaksi yang menggunakan kartu kredit. Sedangkan *Microservice Payment Core* adalah *service* yang bekerja sebagai inti dari sistem pembayaran pada situs maupun aplikasi tiket.com. Struktur tabel pada *System Parameter* dapat dilihat pada Tabel 3.8 berikut.

Tabel 3.8 Struktur tabel atau *collection* System Parameter

No	Kolom	Tipe Data	Keterangan
1	Id	ObjectId	Id di auto generate oleh mongoDB
2	Variable	String	Merupakan variable dari data
3	Value	String	Merupakan nilai dari variable tersebut dan memiliki nilai default
4	Description	String	Berisi penjelasan dari variable tersebut
5	createdBy	String	Menyimpan siapa yang telah membuat data ini.
6	createdDate	Date	Merupakan jenjang waktu saat data dibuat
7	updatedDate	Date	Merupakan jenjang waktu saat ada perubahan pada data
8	isDeleted	Integer	Sebuah flag untuk mengetahui apakah data ini sudah di hapus atau belum

Setelah melihat rancangan tabel pada *collection* System Parameter tersebut, selanjutnya dilakukan perbaikan pada CRUD yang telah ada. Pada sistem CRUD sebelumnya, selalu terjadi *error* saat melakukan CRUD melalui API System Parameter. Error tersebut terjadi karena ada kesalahan pada *code* yang dibuat oleh *developer* sebelumnya. Setelah dilakukan perbaikan, maka dihasilkan API terbaru yang memiliki *method* GET, POST, PUT, dan DELETE.

Tabel 3.9 Parameter body untuk method GET

Name	Type	Required
Page	Integer	True
Limit	Integer	False

Untuk mengambil data pada collection *System Parameter* dapat dilakukan dengan melakukan *request* dengan *method* GET pada API *System Parameter*. Untuk melakukan request tersebut, diperlukan beberapa parameter seperti yang ditunjukkan pada Tabel 3.9.

Tabel 3.10 Parameter body untuk method POST

Name	Type	Required
Variable	String	True
Value	String	True
Description	String	True

Untuk menambahkan data pada *collection* System Parameter dilakukan *request* dengan menggunakan *method* POST pada API System Parameter. Parameter pada bagian body dapat dilihat pada Tabel 3.10.

Tabel 3.11 Parameter body untuk method PUT

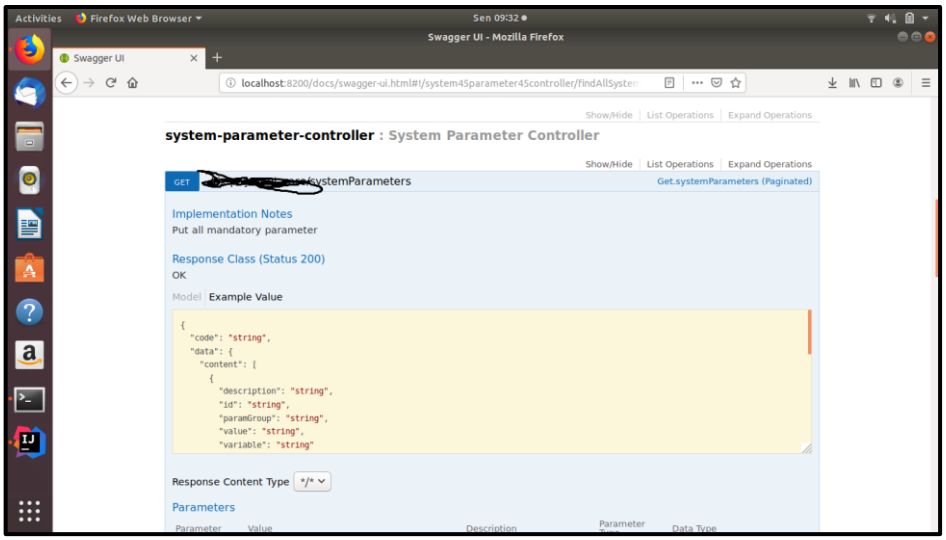
Name	Type	Required
Id	Path/string	True
Variable	String	True
Value	String	True
Description	String	True

Untuk mengubah data yang sudah ada pada collection *System Parameter*, dapat dilakukan *request* dengan *method* PUT pada API System Parameter. Parameter yang dibutuhkan ditunjukkan oleh Tabel 3.11.

Tabel 3.12 Parameter body untuk method DELETE

Name	Type	Required
Id	Path/string	True

Terakhir, dibuat endpoint untuk melakukan *soft delete* pada data di *collection*. API ini membutuhkan *method* DELETE dan parameter seperti yang ditunjukkan pada Tabel 3.12.



Gambar 3.3 Respon saat melakukan request GET pada System Parameter

Setelah implementasi selesai, API dapat digunakan untuk melakukan CRUD pada *collection System Parameter*. Contoh response saat melakukan request GET pada API dapat dilihat pada Gambar 3.3.

3.3.1.4 Integrasi Metode Pembayaran baru yaitu Pay At Hotel pada Microservice Order Aggregate

Tugas selanjutnya adalah melakukan integrasi metode pembayaran baru Pay At Hotel pada *Microservice Order Aggregate*. Order Aggregate merupakan service yang melakukan *handle* saat pengguna ingin melakukan pembayaran. Saat pengguna ingin melakukan pembayaran, *microservice* Payment Core akan mengirimkan *request* kepada Order Aggregate. Selanjutnya, Order Aggregate akan mengirimkan *response* berupa list dari metode pembayaran apa saja yang bisa digunakan pengguna dalam pemesanan ini. Untuk mengakses API ini, diperlukan parameter seperti yang ditunjukkan oleh Tabel 3.13. Orderhash digunakan untuk memastikan bahwa tidak terjadi *tampering* pada order. OrderId digunakan sebagai referensi terhadap *order* mana yang diproses. hotelPaymentType merupakan sebuah *enum* yang terdiri dari *regular* dan *pay at hotel*. Apabila parameter hotelPaymentType diisi dengan enum *pay at hotel*, maka metode pembayaran selain *pay at hotel* akan di-*set* menjadi *false*.

Tabel 3.13 Parameter *request* pada Order Aggregate

Name	Type	Required
Orderhash	String	True
OrderId	String	True
hotelPaymentType	Enum	False

Contoh *response* saat sukses melakukan *request* pada API ini adalah seperti Gambar 3.4. Data yang dikembalikan berupa list dari metode pembayaran yang dapat digunakan pada *order* tersebut.

Gambar 3.4 Contoh *response*

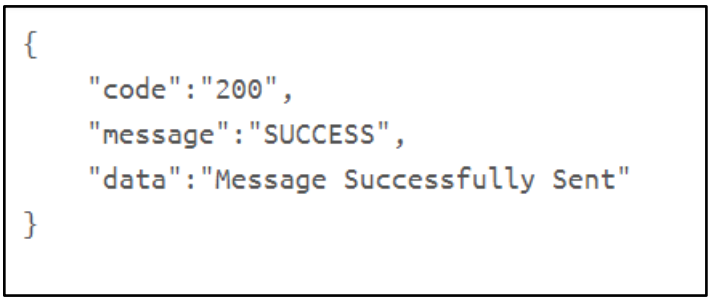
3.3.1.5 Integrasi notifikasi pembayaran melalui *microservice* Whatsapp

Tugas selanjutnya yang dikerjakan adalah melakukan integrasi notifikasi pembayaran melalui *microservice* Whatsapp. Sebelumnya, *microservice* Payment Notification memanggil langsung *third-party* yang selanjutnya memproses pesan notifikasi tersebut. Namun, tim Whatsapp telah membuat *microservice* sendiri yang mengatur segala hal yang berurusan dengan Whatsapp. Oleh karena itu, dilakukan pengintegrasian notifikasi pembayaran yang sebelumnya langsung memanggil *third-party* menjadi melalui *microservice* Whatsapp dengan membuat sebuah *endpoint*. Endpoint tersebut akan menerima *request* dan mengolah *request* tersebut agar dapat melakukan *request* ke *microservice* Whatsapp. *Endpoint* yang dibuat memiliki parameter seperti yang tertera pada Tabel 3.14.

Tabel 3.14 Parameter *body* pada *endpoint* Whatsapp

Name	Type	Required
Param	Array	True
phoneNumber	String	True
ReferenceId	Integer	True
templateCode	String	True

Parameter param merupakan sebuah *array* dimana *array* tersebut dapat dimasukkan beberapa *parameter* lain seperti OrderId. *Parameter* phoneNumber merupakan nomor whatsapp dari pengguna yang ingin dikirimkan pesan. templateCode berisi kode dari *template* pesan yang akan dikirimkan. Contoh *response* saat *request* berhasil dapat dilihat pada Gambar 3.5.

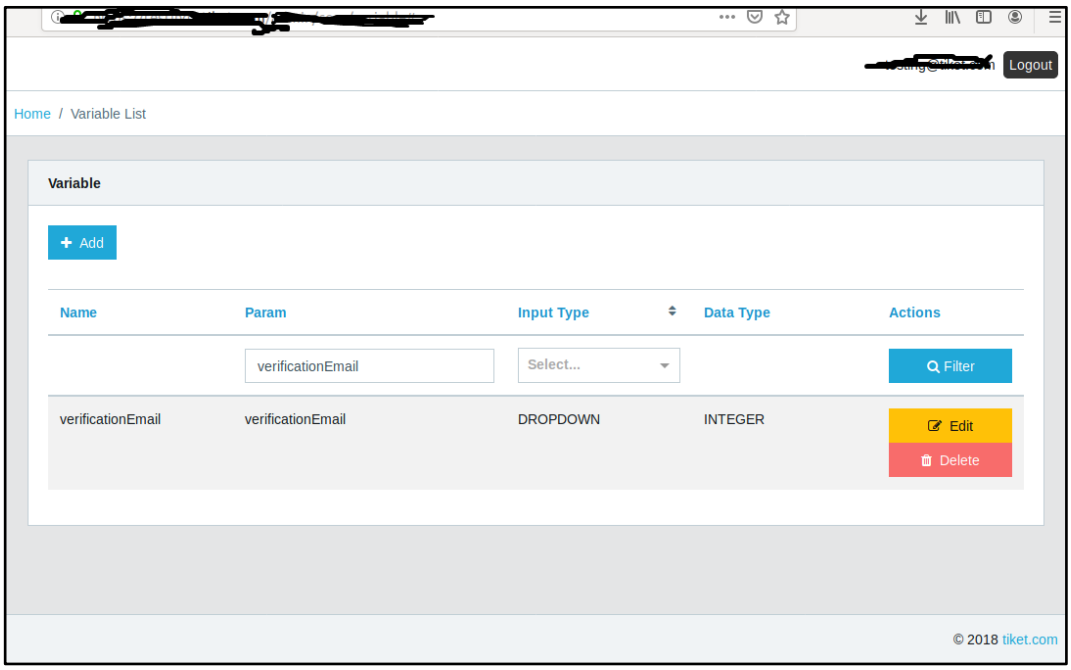


Gambar 3.5 *Response* saat *request* berhasil

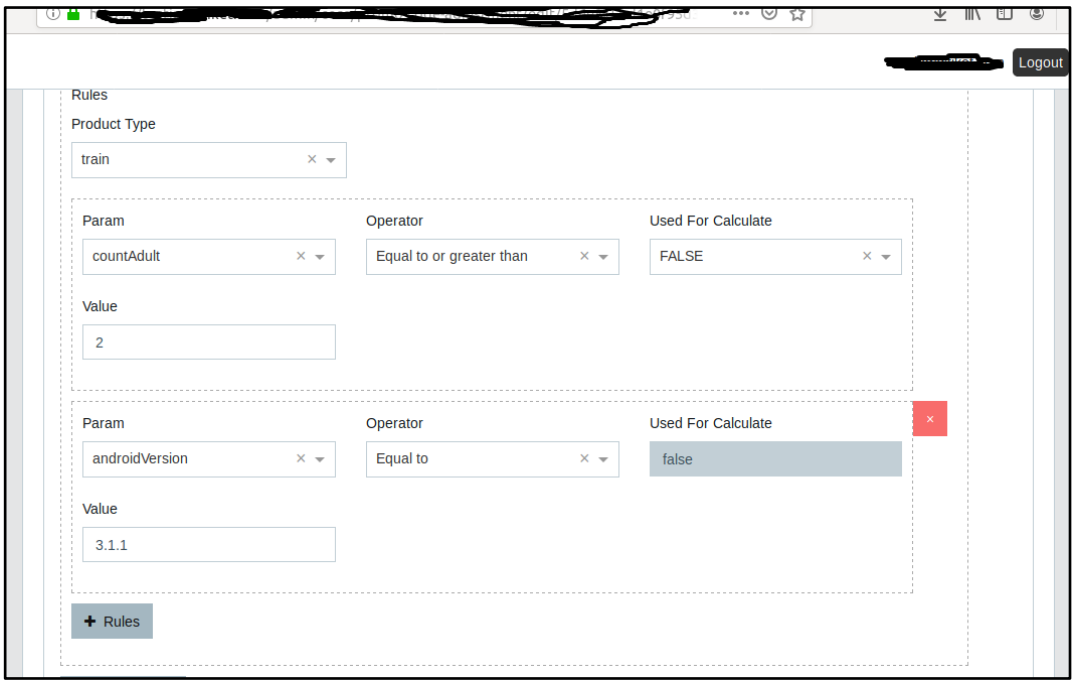
3.3.1.6 Menambahkan *variable* validasi email, nomor *handphone*, dan *apps version* pada microservice Promocode

Pada minggu ke-7 hingga ke-8, tugas yang diberikan adalah untuk menambahkan *variable* validasi email, nomor hp, dan *apps version* pada *microservice* Promocode. *Microservice* Payment Core akan mengirimkan data berupa apakah email dan nomor hp pengguna sudah divalidasi, dan *apps version* dari pengguna. Variable tersebut dibuat dengan tujuan agar saat membuat kode promo baru, dapat dibuat aturan apabila pengguna belum melakukan validasi email,

ataupun belum melakukan validasi nomor hp sehingga tidak bisa menggunakan kode promo tersebut. Setelah itu dibuat kode error baru untuk meng-*handle* apabila pengguna mencoba kode promo tersebut walaupun belum melakukan validasi email maupun nomor hp. Sedangkan untuk *apps version* perlu dilakukan pengecekan terlebih dahulu apakah yang pengguna pakai adalah android atau iOS. Implementasi *variable* yang telah dibuat dapat dilihat pada Gambar 3.6 dan 3.7.



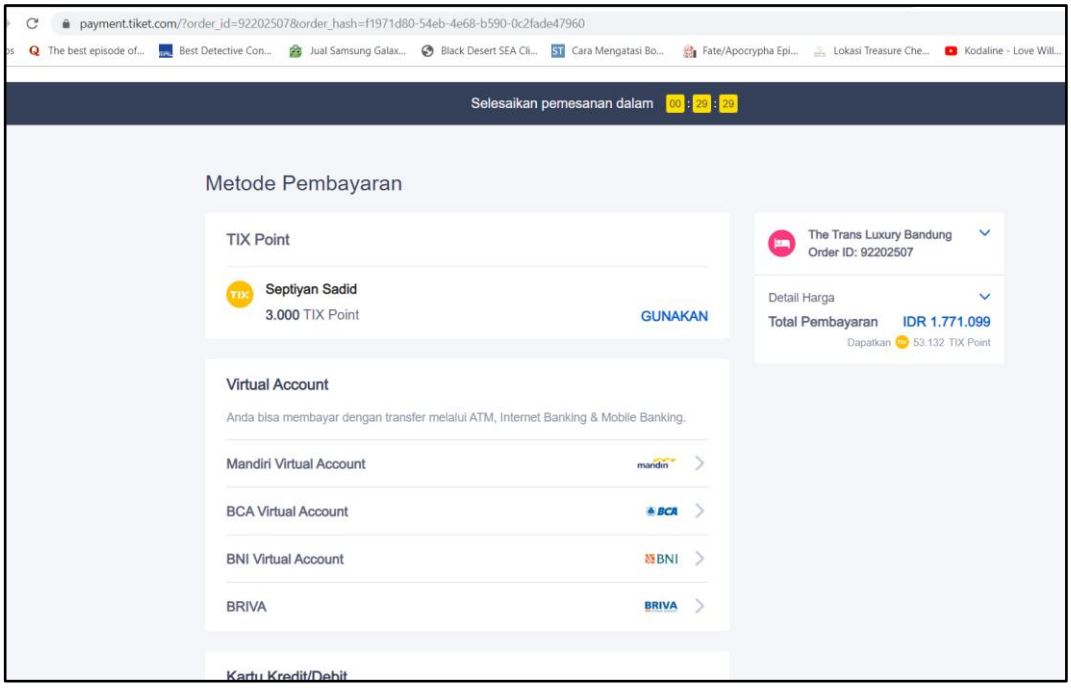
Gambar 3.6 Variable verificationEmail yang digunakan untuk validasi email.



Gambar 3.7 Penggunaan variable androidVersion pada promocode rules

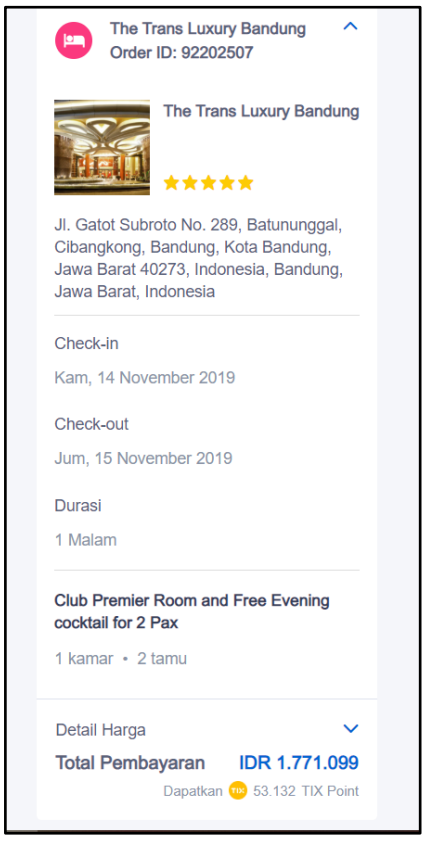
3.3.1.7 Implementasi Metode Pembayaran baru Akulaku pada *microservice* Payment Core

Tugas terakhir yang dilakukan dalam proses kerja magang di PT Global Tiket Network adalah membuat implementasi metode pembayaran baru yaitu Akulaku pada *microservice* Payment Core. Payment Core merupakan inti dari sistem pembayaran pada tiket.com. Payment Core meng-*handle* beberapa bagian seperti saat pengguna ditunjukkan metode pembayaran yang dapat dilakukan seperti yang ditunjukkan oleh Gambar 3.8.



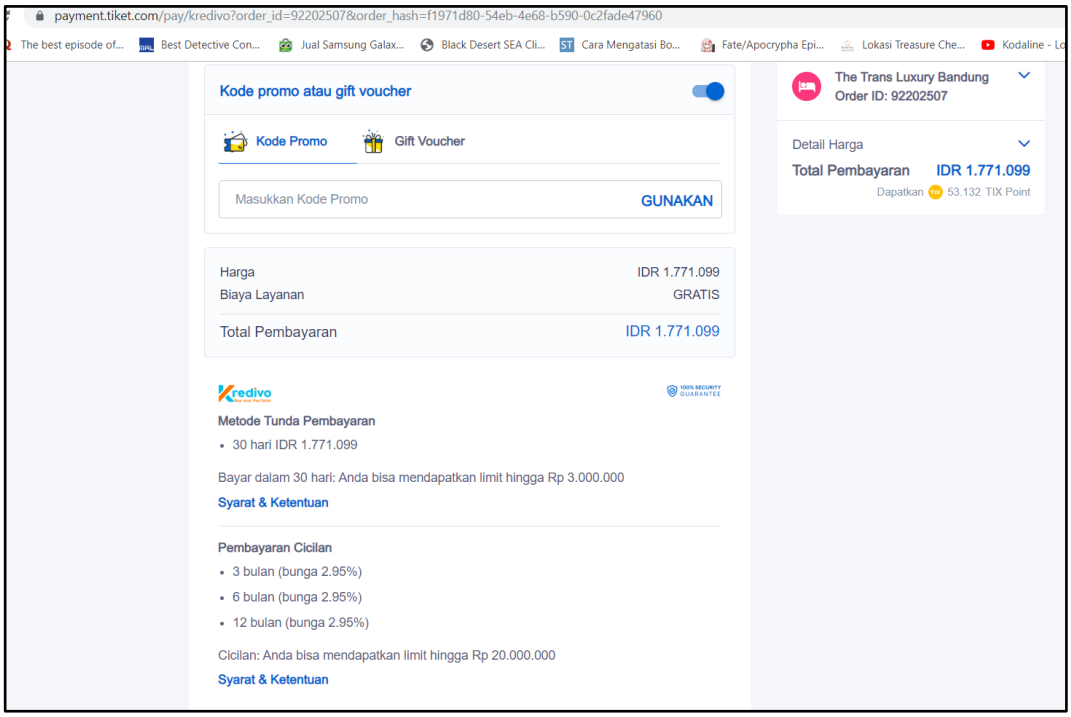
Gambar 3.8 Tampilan saat pengguna memilih metode pembayaran

Lalu mengirimkan detail dari pemesanan seperti Gambar 3.9. Detail tersebut meliputi order id, detail harga, total pembayaran, dan lain-lain.



Gambar 3.9 Detail pemesanan

Terdapat juga informasi penting terkait metode pembayaran yang dipilih pengguna seperti pada Gambar 3.10 hingga pengguna dapat menyelesaikan pemesanan dengan baik dan benar.



Gambar 3.10 Informasi penting terkait metode pembayaran yang dipilih

Untuk menambahkan metode pembayaran baru, maka terdapat 3 *collection* yang akan terdampak, yaitu *collection* payment_config, payment_order, dan important_info.

Tabel 3.15 Struktur tabel atau *collection* payment_config

No	Kolom	Tipe Data	Keterangan
1	Id	ObjectId	Id yang di-*auto generate* oleh mongoDB.
2	paymentName	String	Nama dari metode pembayaran.
3	paymentStatus	Boolean	Status metode pembayaran tersebut saat ini.
4	label	String	Label dari suatu metode pembayaran.

Collection payment_config seperti yang ditunjukkan pada Tabel 3.15 digunakan untuk menyimpan nama metode pembayaran, label, dan statusnya saat ini.

Tabel 3.16 Struktur tabel atau *collection* payment_order

No	Kolom	Tipe data	Keterangan
1	Id	ObjectId	Id yang di-*auto generate* oleh MongoDB
2	groupName	String	Group dari beberapa metode pembayaran yang terhubung dengan suatu kesamaan
3	Order	Integer	Urutan *priority* dari grup yang akan ditampilkan pada saat memilih list pembayaran
4	paymentMethod	Array	Kumpulan dari metode pembayaran yang termasuk dalam grup

Sedangkan *collection* payment_order digunakan untuk melakukan *grouping* terhadap metode pembayaran yang tersedia, lalu diurutkan berdasarkan *order*-nya.

Tabel 3.17 Struktur tabel atau *collection* important_info

No	Kolom	Tipe data	Keterangan
1	Id	ObjectId	Id yang di-*auto generate* oleh mongoDB
2	paymentName	String	Nama dari metode pembayaran
3	infoStatus	Boolean	Status dari metode pembayaran tersebut
4	Message	Array	Pesan dari metode pembayaran tersebut

Collection important_info berguna untuk menyimpan informasi penting terkait suatu metode pembayaran. Informasi dapat berupa rincian cicilan, batas waktu pembayaran, dan lain-lain.

Setelah mengimplementasikan metode pembayaran baru kedalam *service*, selanjutnya dibuat *data migration* untuk memasukkan data-data dari metode pembayaran baru yaitu akulaku. *Data migration* memiliki 2 fungsi utama, yaitu migrate dan rollback. Apabila *data migration* dijalankan, maka fungsi migrate akan dilakukan, dimana dalam konteks ini fungsi tersebut akan menambahkan data kedalam 3 *collection* yang telah disebutkan diatas. Apabila terjadi masalah atau *error*, maka fungsi kedua yaitu rollback akan dijalankan agar apapun perubahan yang terjadi setelah fungsi migrate dijalankan akan dihapus. Fungsi rollback juga dapat digunakan apabila *developer* ingin melakukan *fresh install.*

3.3.2 Kendala yang ditemukan

Dalam proses kerja magang yang dijalani selama 3 bulan, terdapat beberapa kendala-kendala yang ditemukan antara lain:

a) Kendala Teknis

   Kendala yang dihadapi pertama kali adalah belum adanya pengetahuan akan *framework* dan *tools* yang akan digunakan selama proses kerja magang. Perbedaan versi dan masalah konfigurasi juga kerap terjadi yang menyebabkan terganggunya proses kerja magang.

b) Kendala Non Teknis

   Kendala lainnya yang dihadapi adalah kendala non teknis seperti malu bertanya pada awal proses kerja magang. Terdapat juga kendala yaitu meja kerja yang ditempatkan sangat jauh dari meja tim dikarenakan tidak adanya tempat lagi. Hal tersebut menyebabkan kurangnya komunikasi kepada anggota tim lainnya.

3.3.3 Solusi atas Kendala yang Ditemukan

Untuk menghadapi masalah teknis, dilakukan pembelajaran lebih intens mengenai *framework* dan *tools* yang digunakan, baik itu melalui pertanyaan-pertanyaan kepada anggota tim yang lebih mengetahui maupun dengan mempelajari menggunakan media internet.

Sedangkan untuk masalah non teknis, dilakukan pendekatan lebih baik dengan seluruh anggota tim, baik itu melalui *chat* ataupun bertemu langsung. Solusi dari masalah meja yang berjauhan dengan anggota tim lainnya adalah dengan rutin bertanya kepada bagian *General Affairs* apabila ada meja yang tersedia disekitar anggota tim lainnya.