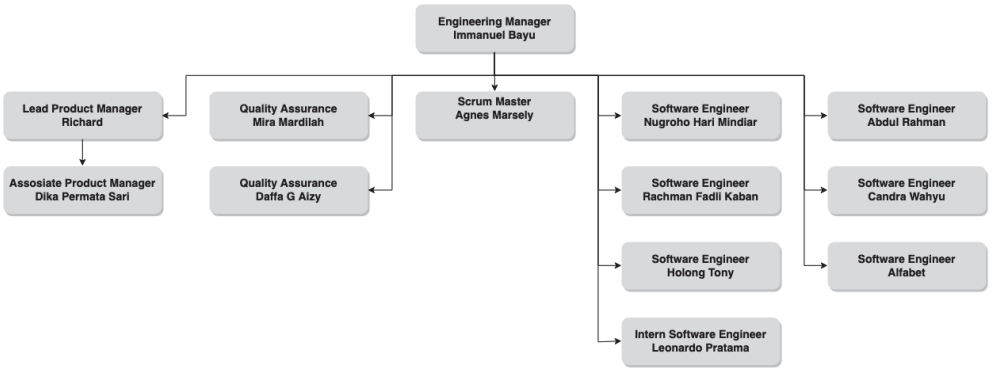
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Kegiatan kerja magang dilaksanakan di PT. Global Tiket Network pada tanggal 10 Juni 2019 hingga 2 Desember 2019 diposisikan sebagai *Backend Engineer* dalam tim MY-ORDER yang kemudian dipindahkan ke dalam tim ORDER. Proses kerja magang di dalam tim ORDER di supervisi oleh Immanuel Bayu sebagai Engineer Manager, dibimbing dan diarahkan oleh Nugroho Hari Mindiar sebagai Senior Backend Engineer dan pemimpin tim.



Gambar 3.1 Struktur Tim
(Sumber: Dokumen Pribadi Penulis)

Proses koordinasi untuk pengembangan selama proses magang terjadi dalam beberapa langkah. Langkah pertama, divisi *Business Intelligence* akan melakukan rapat bersama *Board of Director* untuk membahas masalah yang ingin diselesaikan. Langkah kedua, divisi *Business Intelligence* memberikan rangkaian pekerjaan yang harus diselesaikan kepada divisi *Product* dari masing masing departemen. Langkah ketiga, Divisi *Product* akan merangkai jadwal pekerjaan yang akan dikerjakan oleh *Developer* pada departemennya. Langkah keempat, Divisi

Product mengadakan pertemuan dengan *Developer* untuk membahas pekerjaan yang akan diselesaikan pada sprint selanjutnya.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama proses kerja magang adalah mengembangkan sebuah modul settlement pembayaran Tiket.com. Tugas-tugas yang dilakukan adalah mengembangkan *repository* (seperti *model* pada metode *Model View Controller* di PHP) untuk proses *checkout* dan *logging* dengan bahasa pemrograman Golang. Pekerjaan lainnya yang dilakukan yaitu mengembangkan *service* untuk mengirim notifikasi ke dalam aplikasi Slack, dan melakukan *pair-programming* dengan anggota tim untuk mengembangkan *service sidecar* dan *dead letter queue* dengan menggunakan Golang.

Proses kerja magang menggunakan perangkat lunak Visual Studio Code sebagai *code-editor*, Git Bash sebagai pengontrol versi modul. Setiap anggota tim mengerjakan pekerjaannya di dalam *branch*nya masing-masing yang kemudian dilakukan *pull-request* untuk disatukan dengan *branch* utama. Terdapat juga Jenkins yang digunakan sebagai pembangun kode otomatis, Sonarqube sebagai pendeteksi kode yang tidak sesuai dengan aturan penulisan yang ada pada bahasa pemrograman tersebut dan mendeteksi adanya celah dari kode yang telah dikembangkan dan Atlassian Borobudur sebagai papan untuk melihat pekerjaan yang tersedia dan untuk melihat perkembangan dari pekerjaan yang sedang dilakukan. Realisasi dari proses kerja magang dapat dilihat dari Tabel 3.1.

Tabel 3.1 Tabel Realisasi Kerja Magang

No	Aktivitas	Minggu													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Menyesuaikan *device* dengan *environment* yang ada di dalam tim														
2	Mempelajari *Springboot MVC*														
3	Unit Testing														
4	Mempelajari alur modul MY-ORDER-CORE														
5	Mengaktifkan *refund* penerbangan internasional modul MY-ORDER-CORE														
6	Training bahasa pemrograman Golang														
7	Pengembangan *repository checkout dan logging* untuk modul SETTLEMENT-GO														
8	Pengembangan *service untuk slack notification, sidecar, dan deadletter untuk modul*														
8	Pengembangan *service untuk deadletter untuk modul TRIGGER-PAID-GO*														
9	Pengembangan *order error* dan *order error validator* untuk modul ORDER-GO														
10	Pengembangan *service RabbitMQ dan Logrus* untuk modul ORDER-GO														
11	Perancangan dan pembangunan GENERAL-API														

3.3 Uraian Pelaksanaan Kerja Magang

Proses pelaksanaan kerja magang dibagi menjadi 3 bagian, yaitu proses pelaksanaan, kendala yang ditemukan dan solusi atas kendala yang ditemukan.

3.3.1 Proses Pelaksanaan

Proses pengembangan modul *settlement* pembayaran Tiket.com ini membutuhkan perangkat lunak maupun perangkat keras. Perangkat lunak yang digunakan untuk mengembangkan modul ini adalah sebagai berikut.

a. Golang versi 1.13 darwin/amd64

b. Visual Studio Code versi 1.38.1 (x64)

c. Google Chrome versi 69.0.497.128 (x64)

d. Git Bash versi 2.23.0

e. Homebrew versi 2.1.11

f. Mysql versi 8.0.15 (Homebrew)

g. *Operating System* macOS versi Darwin 18.7.0 (x64)

Perangkat keras yang digunakan untuk mengembangkan modul ini adalah *notebook* Macbook Pro 2018 dengan spesifikasi sebagai berikut.

a. Processor Intel Core i5 2,3 GHz

b. RAM 8 GB 2133 MHz LPDDR3

c. Intel Iris Plus Graphics 655 1536 MB

Proses pengembangan selama kegiatan kerja magang terdiri menjadi beberapa pekerjaan. Pekerjaan yang dilakukan selama proses kerja magang didapatkan dari papan pekerjaan yang telah disusun pada saat perencanaan *sprint* yang antara lain mengembangkan *repository* (seperti model yang digunakan untuk melakukan *query* ke dalam database) untuk proses *checkout* dan *logging*, mengembangkan *service* untuk mengirimkan notifikasi ke layanan Slack, mengembangkan sistem *sidecar* (sebuah sistem yang digunakan untuk mengecek apakah *request* yang didapatkan berasal dari pengguna yang memiliki izin untuk masuk ke dalam sistem tersebut, dan jika tidak akan dialihkan kepada sistem lain), dan sistem antrian *dead letter* (sebuah sistem yang aktif ketika sistem antrian yang digunakan oleh tiket.com tidak dapat dijangkau, maka sistem ini akan melakukan proses antrian untuk mendaftarkan *request* secara terus menerus hingga sistem antrian tersebut dapat dijangkau dan *request* telah dimasukan ke dalam antrian). Proses pengembangan selama kegiatan kerja magang diuraikan dengan *requirement* dari modul *settlement*, *flowchart* sebagai representasi dari *source code*. Kemudian implementasi dari hasil pekerjaan seperti, struktur JSON yang digunakan untuk melakukan *request* dan memberikan *response* dan bagian terakhir terdapat *documentation* dari *endpoint* yang tersedia.

a. Requirement

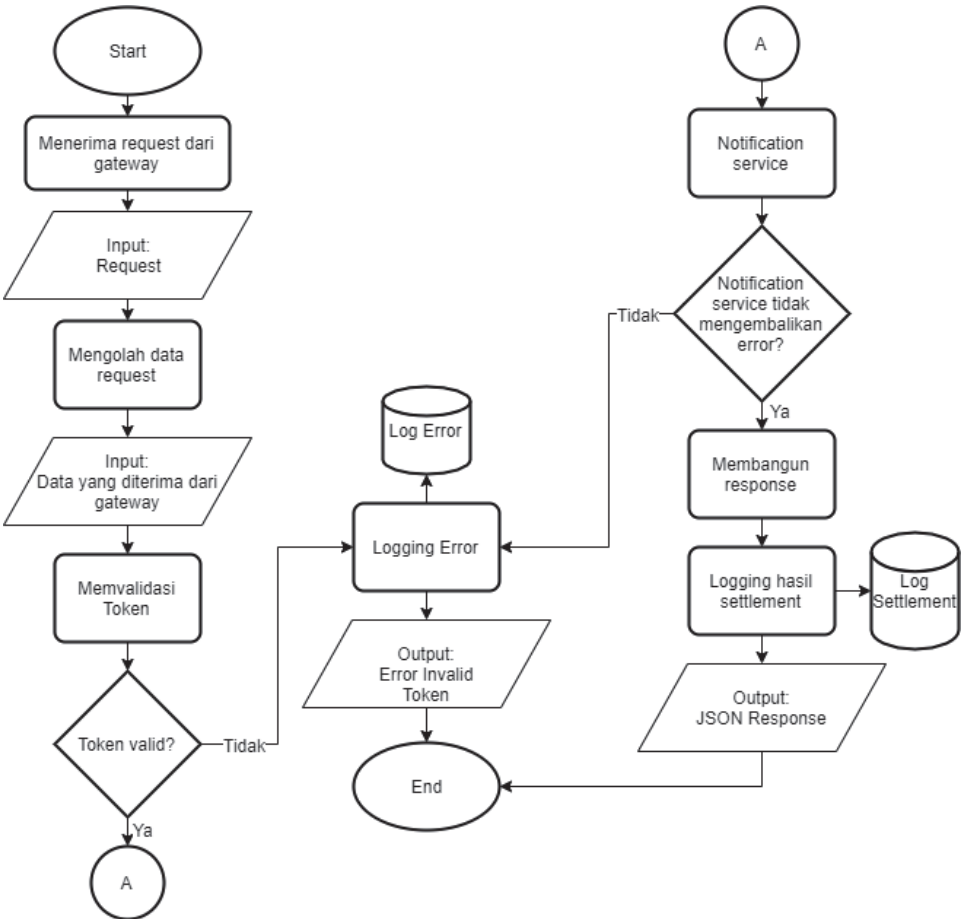
Requirement merupakan suatu batasan yang dibentuk oleh pengguna agar suatu sistem yang dibangun sesuai dengan yang diperlukan. Requirement dari modul ini adalah sebagai berikut. Pengguna yang melakukan pembayaran akan menjalankan proses *settlement* ini ketika pengguna melakukan pembayaran. Ketika pengguna melakukan pembayaran, modul ini akan melakukan pengecekan terhadap proses pembayaran dengan memeriksa *request* pembayaran yang diberikan dan yang diterima oleh modul *Payment-Core*. Modul ini membutuhkan *order* aktif yang telah dipesan oleh pengguna dan berstatus dan berada di *shopping cart*. Setelah pengecekan berhasil, sistem akan melakukan proses *publish* kepada modul *trigger paid* yang berguna sebagai *trigger* ketika *order* telah dibayar dan akan mengubah status *order* dari *shopping cart* menjadi *paid.* Ketika pengguna gagal melakukan proses *settlement* pada modul ini, akan diberikan informasi jelas mengenai kegagalan yang terjadi dan akan memberikan *reference id* terhadap pembayaran dari *order* yang dipesan.

b. Flowchart

Flowchart yang diberikan pada laporan kerja magang ini digunakan untuk merepresentasi *source code* dari modul *settlement* yang telah dikembangkan dengan menggunakan bahasa golang. Adapun *flowchart* yang telah dibuat terdiri dari sebuah *controller* dan berbagai macam *service* yang tersedia di dalam modul tersebut yang antara lain terdiri dari *controller* dan *service settlement* yang terbagi menjadi *service* validasi *request*, *service* validasi *payment, service insert order payment, service process settlement, service update payment order, service process payment detail, service sidecar* dan *service publish trigger paid*.

1. Controller

Controller mengatur jalannya suatu proses utama dari modul, dan dipanggil ketika *endpoint* menerima sebuah *request*. *Request* yang diterima akan di-*map* dengan struktur *request* yang *controller* harapkan. *Request* yang telah di-*map* akan divalidasi *token*nya apakah sesuai dengan *token* yang diketahui oleh *controller*. Jika *controller* tidak mengetahui *token* yang diberikan pada *request, controller* akan menuliskan *log* yang disimpan didalam database. Setelah *token* divalidasi, *service* utama dari modul ini yang bernama *notification service* dipanggil dan diberikan *request* yang diterima oleh *controller*. *Flowchart* untuk dapat dilihat pada gambar 3.2.

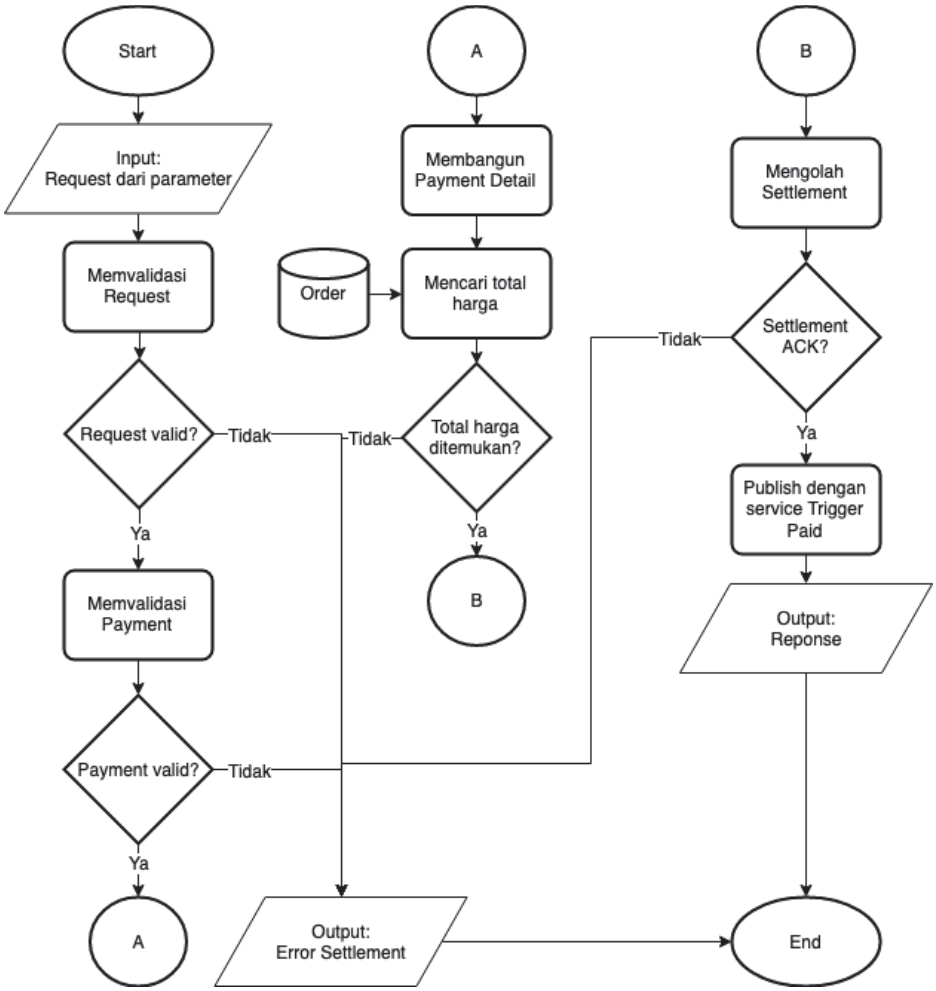


Gambar 3.2 Flowchart Controller

2. Service Settlement

Service merupakan layanan yang tersedia pada suatu modul yang terpisah berdasarkan fungsinya masing-masing seperti untuk memproses data, mutasi data, mengecek data, menghasilkan *response* dan *logging* aktivitas sistem maupun error.

Service settlement ini merupakan service utama yang dipanggil langsung oleh *controller* ketika menerima *request. Service* ini terbagi menjadi beberapa *sub-service* yang mempunyai beberapa fungsi *private* yang hanya dapat dipanggil oleh *service*. Service *settlement* ini menerima masukan berupa *request context* yang membawa nilai, *deadline*, dan *error* dari fungsi atau API lain, dan sebuah *request* khusus yang dibentuk dari seluruh informasi yang diperlukan untuk menyelesaikan proses *settlement* pembayaran ini. *Flowchart* dari *service settlement* digambarkan pada gambar 3.3.

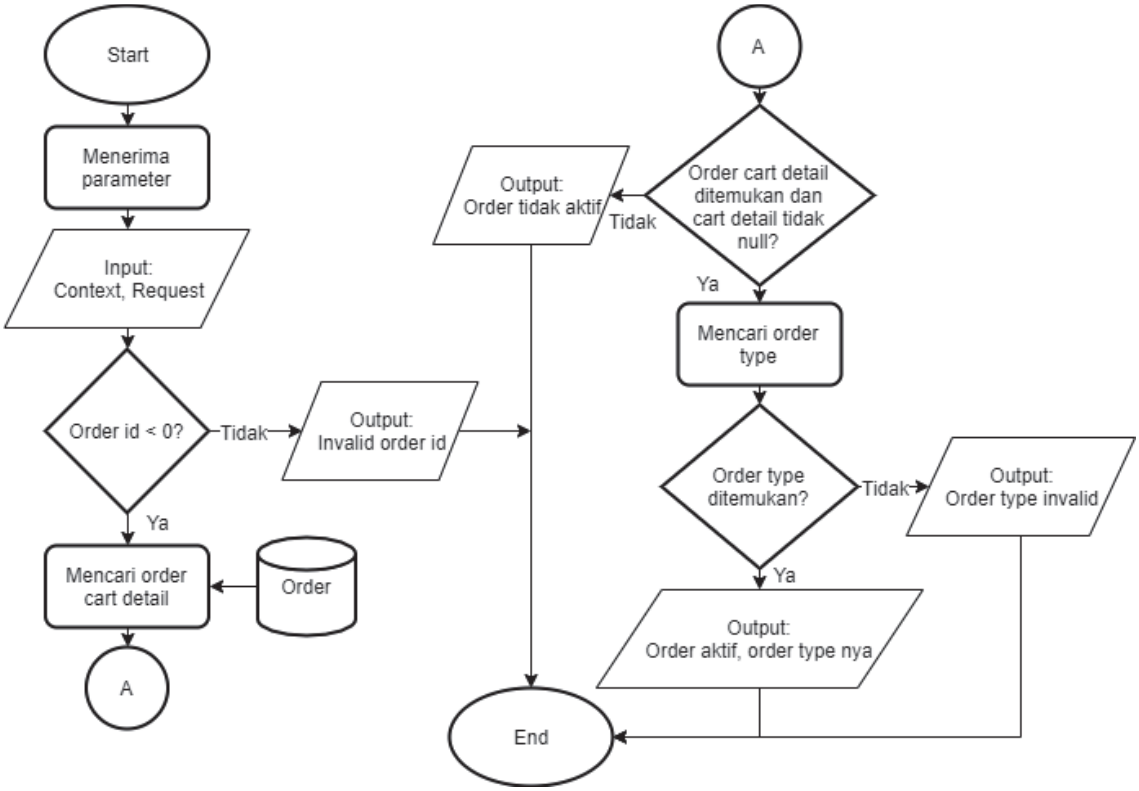


Gambar 3.3 Flowchart Service Settlement

3. Service Validasi Request

Service validasi *request*, dipanggil oleh *service settlement* dengan input context dan request. *service* ini berfungsi untuk memvalidasi apakah *request* yang diberikan memiliki *order id* yang valid, detail *order cart*-nya tidak kosong dan *order type*-nya sesuai dengan yang ada. Proses jalannya *service* ini dimulai dengan pemeriksaan terhadap order id yang diterima di dalam *request*, jika order idnya lebih kecil dari 0 maka akan mengembalikan error "Invalid Order ID". Setelah diperiksa, dicari order *detail*nya dari database. Jika order *detailnya* tidak ditemukan maka akan mengembalikan error "Order Tidak Aktif". Setelah order *detail*

ditemukan maka akan dicari order *type* dari *request* ke dalam database untuk memastikan apakah order *type* tersebut valid, dan jika tidak ditemukan maka akan dikembalikan error "Tipe order invalid" sedangkan jika ditemukan maka akan dikembalikan status order aktifnya dan order *type*-nya. *Flowchart* dari *service* validasi *request* digambarkan pada gambar 3.4.

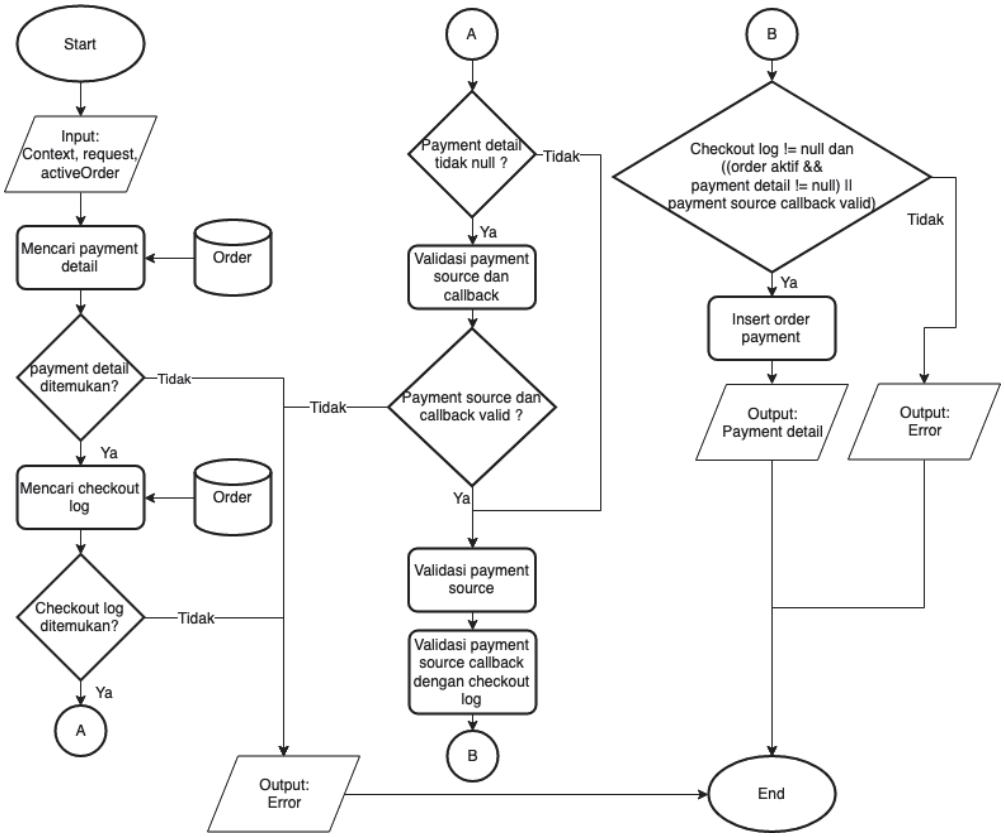


Gambar 3.4 Flowchart Service Validasi Request

4. Service Process Settlement

Service process settlement, dipanggil oleh *service settlement* untuk memvalidasi apakah data yang diberikan oleh *request* valid untuk diproses pembayarannya. Pertama, dicari *payment detail* di dalam *database*. Jika tidak ditemukan, akan dikembalikan sebuah error, sebaliknya jika ditemukan akan mencari log *checkout* di dalam *database*. Jika log *checkout*-nya tidak ditemukan,

akan dikembalikan sebuah error dan akan melakukan validasi *payment source* dan *callback.* Ditemukan atau tidak prosesnya akan berlanjut untuk melakukan validasi payment source dan validasi *payment source* berdasarkan *checkout* log. Jika *payment source callback* valid atau payment detailnya tidak null dan ordernya aktif dan checkout lognya tidak null akan dijalankan *service insert order payment,* mengembalikan payment detail. Jika tidak, akan dikembalikan error. *Flowchart service* validasi *payment* digambarkan pada gambar 3.5.

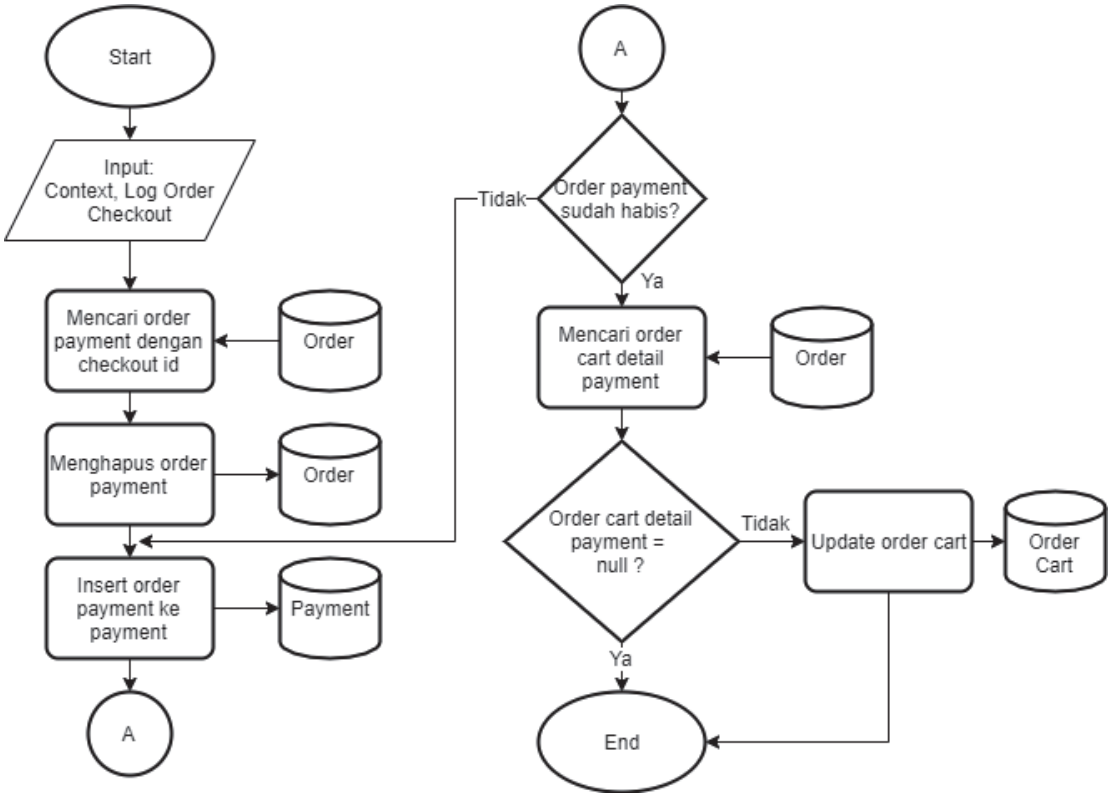


Gambar 3.5 Flowchart Service Validasi Payment

5. Service Insert Order Payment

Service insert order payment, dipanggil oleh *service* validasi *payment* ketika *payment valid*. Pertama akan dilakukan pencarian terhadap order payment dengan menggunakan checkout id yang dapat mengembalikan sebuah *array*, dan order

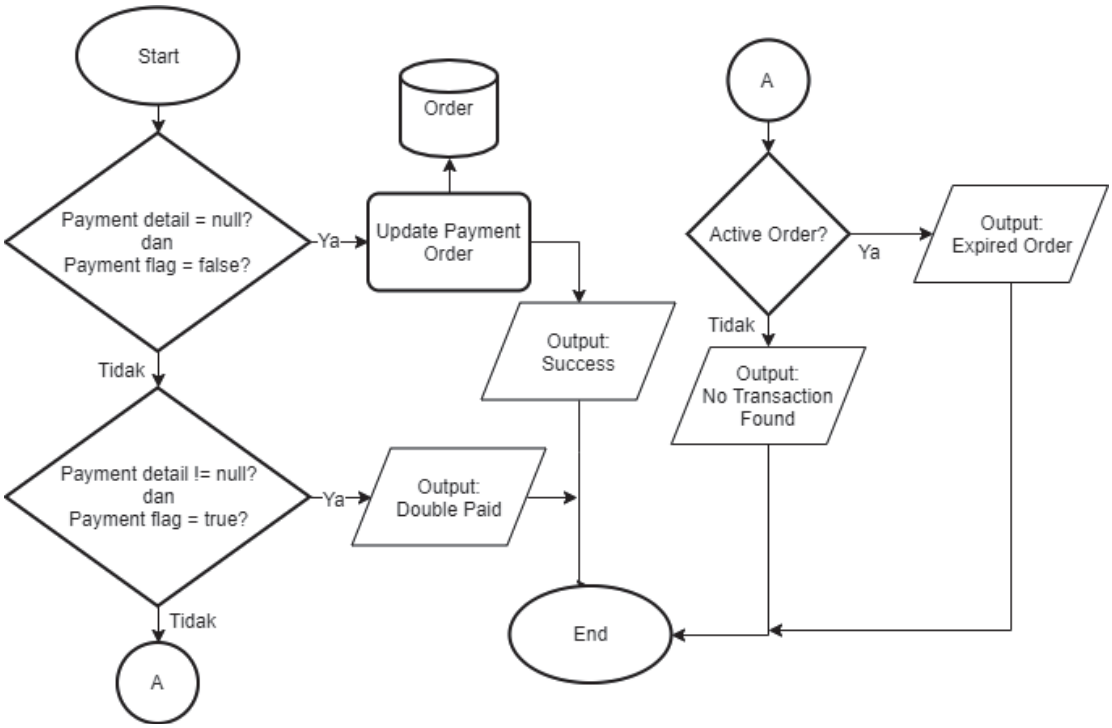
paymentnya dihapus. Kemudian order payment yang didapatkan dengan checkout id akan dimasukan kedalam *database* payment dan akan dilakukan terus menerus hingga order payment yang terakhir. Setelah looping jika *order cart detail*-nya tidak null akan diperbaharui dengan menggunakan checkout log dan order detail yang sudah didapatkan sebelumnya. *Flowchart service* validasi *payment* digambarkan pada gambar 3.6.



Gambar 3.6 Flowchart Insert Order Payment

6. Service Process Settlement

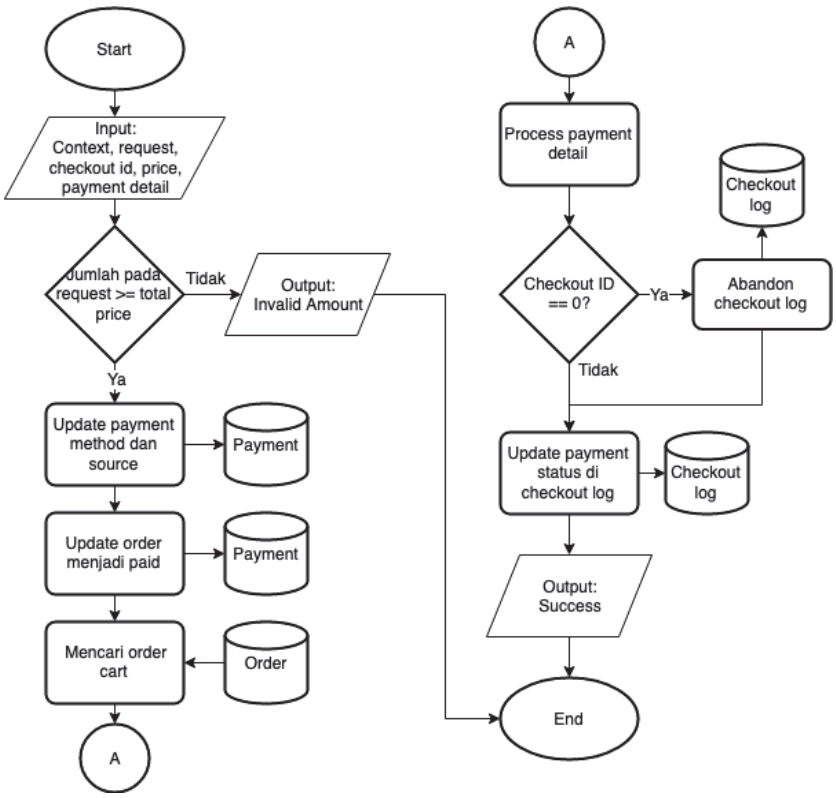
Service process settlement, dipanggil oleh *service settlement* untuk mengecek apakah proses *settlement* sudah dilakukan sebelumnya. Pertama, apakah *payment detail*-nya null dan *payment flag*-nya false. Jika *payment detail*-nya null dan *payment flag*-nya false akan diperbaharui terhadap *payment order* dan mengembalikan *message* "Success". Jika tidak terpenuhi akan diperiksa kembali apakah *payment flag*-nya *true*. Jika *payment flag*-nya *true* makan akan dikembalikan *message* "Double Paid", dan jika ordernya *payment detail*-nya tidak null, *payment flag*-nya false tetapi ordernya aktif kembalikan *message* "Expired Order". Selain itu kembalikan *message* "No Transaction Found". *Flowchart service* process *settlement* digambarkan pada gambar 3.7.



Gambar 3.7 Flowchart Process Settlement

7. Service Update Payment Order

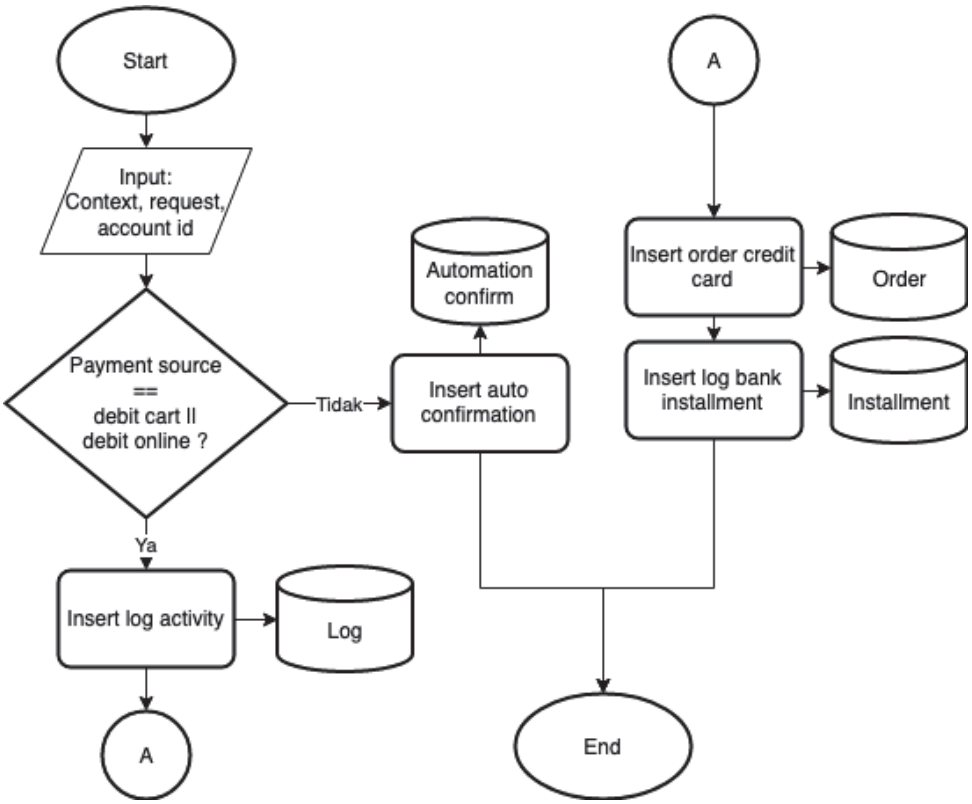
Service update order payment, dipanggil oleh *service* process *settlement* untuk memperbaharui *payment status*. Pertama, jika jumlah total yang harus dibayar di dalam *request* lebih kecil dari total harga dari order yang dilakukan maka akan dikembalikan error "Invalid Amount". Jika lebih besar maka akan diperbaharui *payment method* dan *payment source*-nya dan diperbaharui juga ordernya menjadi *paid*. Jika checkout id yang diterima sebagai *input* nilainya nol, maka akan diperbaharui checkout lognya agar statusnya berubah menjadi "*Abandoned*". Setelah diperbaharui checkout lognya jika checkout idnya nol, maupun yang checkout idnya tidak nol, akan diperbaharui payment statusnya di dalam *checkout log*-nya dan mengembalikan message "*Success*". *Flowchart service* process *settlement* digambarkan pada gambar 3.8.



Gambar 3.8 Flowchart Update Payment Order

8. Service Process Payment Detail

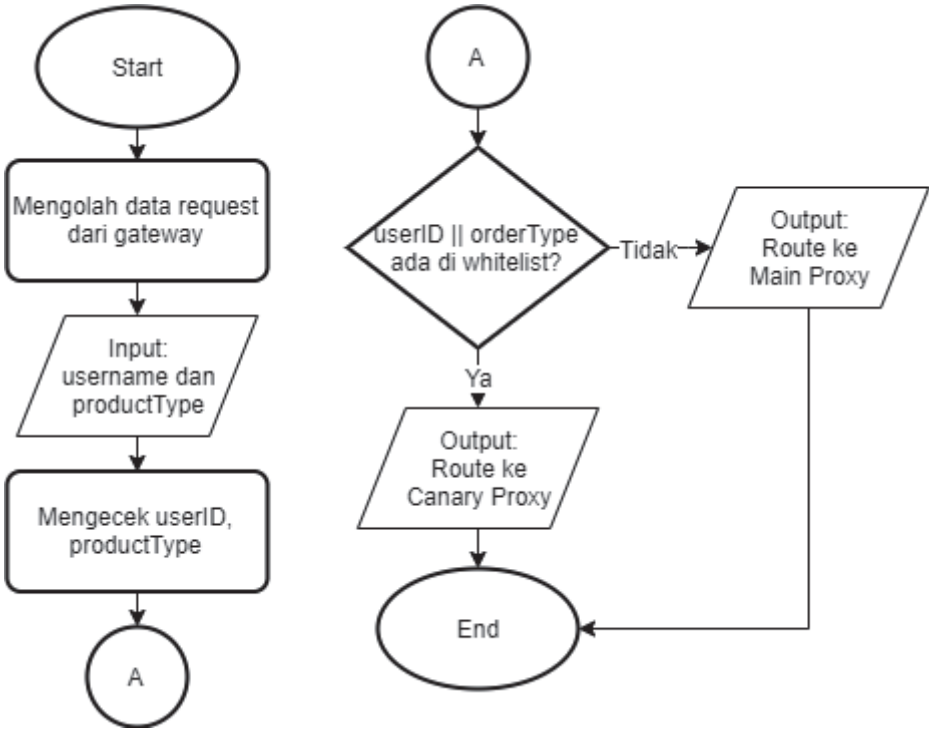
Service process payment detail, dipanggil oleh *service update order payment* untuk memproses detail *payment*. Pertama, jika payment source yang didapatkan dari request nilainya sama dengan "debit card" atau "debit online". Jika tidak, akan langsung melakukan *insert* ke dalam *database* "*automation confirm*". Jika sesuai akan melakukan *insert* aktivitas yang dilakukan ke dalam *log* yang lalu melakukan *insert* order kartu kredit ke dalam *database* dan *insert log*-nya ke dalam *database "installment"*. *Flowchart service* process *payment detail* digambarkan pada gambar 3.9.



Gambar 3.9 Flowchart Process Payment Detail

9. Service Sidecar

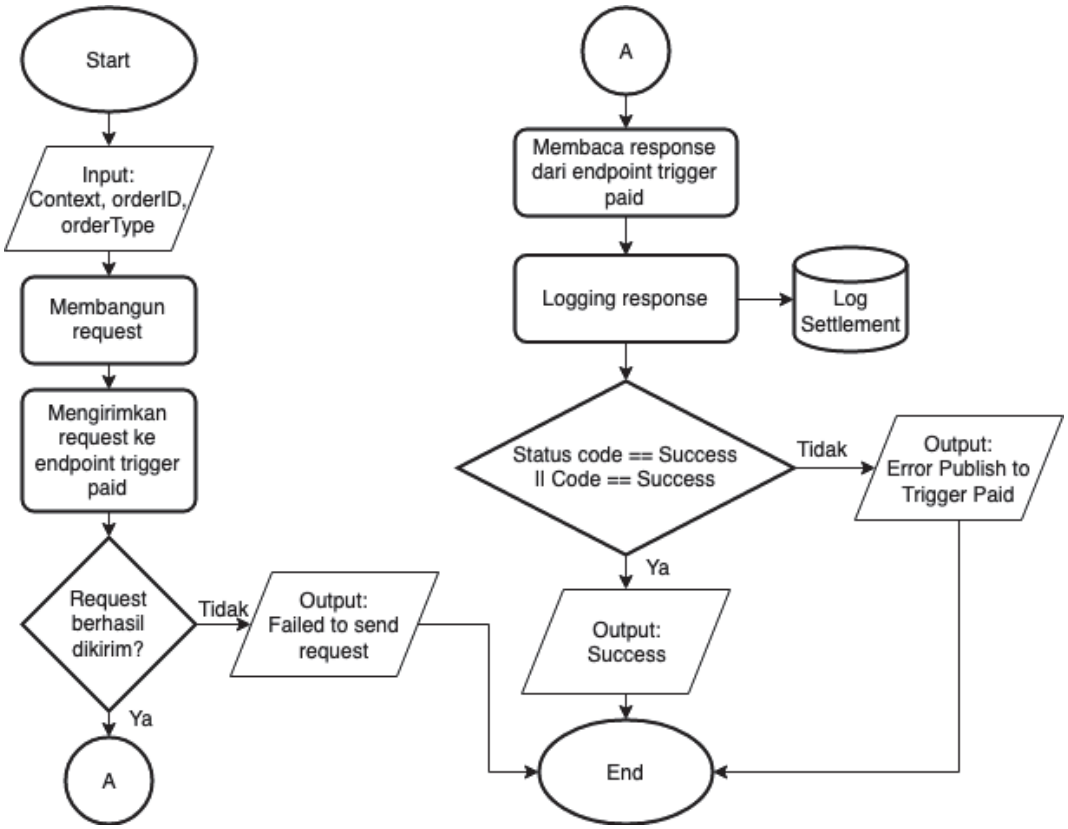
Service sidecar, dipanggil oleh *service* modul *canary*, yang berfungsi untuk memeriksa apakah *request* yang diberikan oleh *endpoint* masuk di dalam sebuah *whitelist*. Berguna sebagai *middleware* yang baik untuk mengetes fitur baru yang telah dikembangkan oleh *Quality Assurance* tanpa mengganggu jalannya tes yang sedang dilakukan oleh tim lain maupun ketika sudah di-*release*, sehingga disaat terjadi error dapat dengan cepat dialihkan ke sistem yang lama. *Flowchart service sidecar* adalah digambarkan pada gambar 3.10.



Gambar 3.10 Flowchart Sidecar

10. Service Publish Trigger Paid

Service publish trigger paid, dipanggil oleh *service settlement* yang berfungsi untuk mengirimkan *request* kepada modul *trigger paid* untuk men-*trigger* notifikasi selesainya proses *settlement*. *Flowchart service publish trigger paid* digambarkan pada gambar 3.11.



Gambar 3.11 Flowchart Publish Trigger Paid

c. Implementasi

Endpoint dari modul *settlement* ini menerima *request* dengan menggunakan format JSON di dalam *body* dari *request*nya. Format JSON dari *request* yang diperlukan digambarkan pada gambar 3.12.

Name	Description
request **object** *(body)*	**Example Value** | Model

```
{
  "abtcParameters": {},
  "amount": "float",
  "authorizationCode": "string",
  "conf_id": "string",
  "customAccount": "string",
  "paymentGateway": "string",
  "paymentSource": "string",
  "reconciliationId": "string",
  "ref_id": 0,
  "timestamp": 0,
  "token": "string"
}
```

Parameter content type

application/json

Gambar 3.12 Format JSON Request

Amount sebagai jumlah yang dibayarkan, *authorization code* sebagai *digital signature*, *custom account* sebagai *account id* dari pengguna, *payment gateway* sebagai gerbang pembayaran yang digunakan oleh pengguna dan *payment source* sebagai merchant pembayaran, *reconciliation id* digunakan sebagai penghubung antara pembayaran dengan yang dibayar, *timestamp* sebagai waktu terjadinya suatu *request* dan *token* merupakan kunci rahasia yang dimiliki oleh sistem. Jika *request* yang diberikan berhasil diproses, maka modul *settlement* akan mengembalikan *response* dengan format JSON yang digambarkan pada gambar 3.13.

Notification Settlement Success

Example Value | Model

```
{
  "ack": "string",
  "conf_id": "string",
  "message": "string",
  "ref_id": 0,
  "timestamp": "string"
}
```

Gambar 3.13 Format JSON Response Success

Ack sebagai *field* yang memberitahu apabila *settlement* yang terjadi sukses, *double paid*, *expired order*, atau transaksi yang diberikan dalam *request* tidak ditemukan didalam *database*. Dan ketika sistem *settlement* tidak berhasil memproses *request* yang diberikan, modul *settlement* akan mengembalikan *response* dengan format JSON yang digambarkan pada gambar 3.14.

Notification Settlement Failed

Example Value | Model

```
{
  "code": "string",
  "message": "string"
}
```

Gambar 3.14 Format JSON Response Failed

Code sebagai *field* yang memberikan kode error yang dialami ketika memproses *request* dan message sebagai *field* yang memberikan detail error yang diberikan.

3.4.1 Kendala yang Ditemukan

a. Pengembangan sistem *sidecar* ditemukan sebuah hambatan untuk mengembangkan sistem yang dapat melakukan pengecekan *request* terhadap aturan yang diberikan, tetapi aturan yang diberikan dapat berubah-ubah dan tidak statik.

3.4.2 Solusi atas Kendala yang Ditemukan

a. Menggunakan *library* Regexp dan YQL, dengan Regexp sebagai *library* yang digunakan untuk mengkonversi aturan yang ditulis sebagai string, dan YQL sebagai *library* yang digunakan untuk mencocokan nilai yang ada di dalam aturan dengan *request* yang didapatkan.