JURNAL INFORMATIKA, Vol.8 No.2 September 2021, Halaman 123~130
ISSN: 2355-6579 | E-ISSN**:** 2528-2247

Implementasi API Payment Gateway Menggunakan Arsitektur Microservice

Erry Julio[1], Magdalena A. Ineke Pakereng[2]

[1,2] Universitas Kristen Satya Wacana
Jl. Diponegoro No.52-60, Salatiga, Kec. Sidorejo, Kota Salatiga, Indonesia

e-mail: [1] 672017022@student.uksw.edu, [2]ineke.pakereng@uksw.edu

Informasi Artikel Diterima: 02-06-2021 Direvisi: 30-06-2021 Disetujui: 01-07-2021

Abstrak

Untuk melakukan integrasi pembayaran *online*, PT. Emporia Digital Raya perlu membuat aplikasi *Payment Gateway* yang dapat meneruskan pembayaran ke bank tujuan. Arsitektur *microservice* dapat diterapkan untuk mengatasi masalah yang ada pada arsitektur *monolithic* dimana pengembangan aplikasi dilakukan dalam bentuk *web service* kecil yang saling berkomunikasi satu dengan yang lain. Dengan menggunakan konsep *routing, payment gateway* memiliki tanggung jawab seperti router dimana *payment gateway* berperan mencari rute ke bank mana data akan dikirimkan. *Payment gateway* akan memproses *request* dan membaca *fields* beneficiaryBankCode untuk menacari rute. Jika bank tujuan tidak ditemukan maka *payment gateway* akan menjalankan *query* dan mencari rute berdasarkan prioritas yang telah ditentukan oleh PT. Emporia Digital Raya.

Kata Kunci: Microservice, Payment Gateway, Bank

Abstract

To integrate online payments, PT. Emporia Digital Raya needs to create a Payment Gateway application that can initiate payments to destination banks. Architectural micro services can be applied to solve problems that exist in monolithic architectures where application development is carried out in the form of small web services that communicate with each other. By using the routing concept, payment gateways have responsibilities such as routers where the payment gateway participates in finding routes to the bank where the data will be sent. The payment gateway will process the request and read the beneficiaryBankCode field to find the route. If the destination bank is not found, the payment gateway will run a query and search for a route based on the priority set by PT. Emporia Digital Raya.

Keywords*: Microservice, Payment Gateway, Bank*

1. Pendahuluan

PT. Emporia Digital Raya atau biasa disebut EDR merupakan anak perusahaan PT. Anabatic Technologies, Tbk yang bergerak di bidang penyedia jasa *E-Business* dan *E-Commerce*. Seiring berkembangnya aplikasi, EDR perlu mengembangkan sistem pembayaran dari tunai menjadi pembayaran *online*.

Untuk melakukan integrasi pembayaran *online*, EDR perlu membuat aplikasi *Payment Gateway* yang dapat meneruskan pembayaran ke bank tujuan. Ada beberapa bank yang bekerja sama dengan EDR yaitu PT Bank Negara Indonesia Tbk, PT Bank Mandiri Tbk, PT Bank Central Asia Tbk.

*Application Programming Inteface (*API) memungkinkan untuk aplikasi saling terintegrasi yang berbeda secara bersamaan. *Representational state transfer* (REST) adalah arsitektur API standar untuk layanan *web* yang memungkinkan meminta sistem untuk mengakses dan memanipulasi sumber daya *web* menggunakan protokol HTTP. REST menggunakan model *client-server*, dimana *client* mengirimkan permintaan HTTP (*GET, POST, PUT, DELETE*), bersama dengan URL. URL berisi parameter variabel di-*encode* yang menjelaskan objek untuk ditindaklanjuti dan *server* merespon dengan *JavaScript Object Notation* (JSON) yang valid (Beni Adi Pranata et al. 2018).

Arsitektur *monolithic* adalah suatu arsitektur dimana semua logika aplikasi berjalan pada satu *server* aplikasi. Proses pemeliharaan dan pembaruan aplikasi

berukuran besar yang dibangun menggunakan arsitektur *monolithic* akan sulit dikarenakan kompleksitasnya (Mufrizal and Indarti 2019). Seiring berkembangnya aplikasi*,* arsitektur *monolithic* akan menjadi besar disisi lain arsitektur *monolithic* juga akan sulit dikembangkan (Setya Budi and Bachtiar 2018). Untuk mengurangi terjadinya kesalahan saat melakukan perbaikan atau penambahan fitur, EDR perlu membangun *service* yang dapat digunakan secara independen, diskalakan secara mandiri, dan memiliki satu tanggung jawab sesuai kebutuhan (Francesco et al. 2019).

Menurut laporan resmi Amazon Web Services (AWS), arsitektur *monolithic* memiliki berbagai masalah yaitu kompleks nya *code* dan sulit dalam pemeliharaan. Karena *code* berada pada satu *server* maka sulit untuk melakukan distribusi. Sedangkan arsitektur *microservice* memiliki manfaat seperti kecepatan pengembangan, kecepatan dalam penyebaran, *code* yang mudah untuk pelihara, dan pengoptimalan yang lebih baik (Amazon Web Services 2017).

. Dalam aplikasi monolitik, setiap perubahan pada fungsionalitas tertentu dari aplikasi keseluruhan akan melibatkan kompilasi ulang dan, dalam beberapa kasus, pengujian ulang seluruh aplikasi sebelum diterapkan lagi. Namun, dalam kasus layanan mikro, hanya layanan yang relevan yang dimodifikasi dan diterapkan ulang karena sifat independen layanan memastikan bahwa perubahan di satu layanan tidak secara logis memengaruhi fungsionalitas layanan lain. Dalam aplikasi monolitik, Beberapa aplikasi monolitik dapat dibangun secara modular tetapi mungkin tidak memiliki modularitas semantik atau logis. Konstruksi modular mengacu pada bagaimana aplikasi dapat dibangun dari sejumlah besar komponen dan pustaka yang mungkin telah dipasok oleh vendor yang berbeda, dan beberapa komponen (misalnya, database) dapat didistribusikan ke seluruh jaringan (Chandramouli 2019).

Arsitektur *microservice* dapat diterapkan untuk mengatasi masalah yang ada pada arsitektur *monolithic*, aplikasi perlu dipecah menjadi beberapa bagian dimana pengembangan aplikasi dilakukan dalam bentuk *web service* kecil yang saling berkomunikasi satu dengan yang lain. *Microservice* menekankan manajemen mandiri dan ringan (Pahl and Jamshidi 2020).

Pada penelitian yang dilakukan (Chandra et al. 2020), Pembuatan aplikasi dengan arsitektur *microservice* dapat dilakukan dengan menentukan *boundary context* yang bertujuan untuk memisahkan *service* dari arsitektur monolitik menjadi *microservice*.

Gerbang API memainkan peran penting dalam arsitektur layanan mikro. API *gateway*, berfungsi sebagai gateway untuk setiap permintaan yang dimulai oleh aplikasi EDR, menyediakan fungsi publik seperti *load balancing* dan *service blowing*. Ini juga mengintegrasikan berbagai layanan mikro dan melindungi kompleksitas dan keragaman sistem, secara jelas menyederhanakan implementasi komunikasi antara aplikasi EDR dan layanan mikro (Zhao, Jing, and Jiang 2018).

Berdasarkan latar belakang yang telah dijelaskan, didapatkan simpulan bahwa diperlukannya arsitektur *microservice* guna mengatasi masalah kompleksitas sistem untuk memenuhi kebutuhan yang ada.

2. Metode Penelitian

Gambar 1. Metode *Agile*

Pembuatan sistem pada penelitian ini dilakukan dengan membagi ruang lingkup menjadi satu set. Beberapa perencanaan, desain, implementasi, dan proyek penutupan proses manajemen dilakukan di setiap iterasi. Metode ini dikenal dengan istilah *Agile* (Jiménez, Afonso, and Fernandes 2020). *Agile* adalah metode pengembangan perangkat lunak jangka pendek yang memerlukan adaptasi cepat dari pengembang terhadap perubahan dalam bentuk apapun. Tahapan *Agile* dapat dilihat pada Gambar 1.

1. Perencanaan

Pada tahap perencanaan, dilakukan analisis terhadap spesifikasi proyek serta mengumpulkan informasi yang diperlukan untuk memenuhi kebutuhan pembayaran *online*. EDR membutuhkan sistem *payment gateway* untuk mengintegrasikan beberapa sistem bank yang

berbeda sehingga proses komunikasi antar sistem dapat berjalan dengan aman dan lancar.

EDR membutuhkan *payment gateway* yang dapat melakukan *routing. Routing* adalah proses memilih jalur untuk lalu lintas di jaringan atau di antara atau di beberapa jaringan. Secara luas, perutean dilakukan di banyak jenis jaringan,antara subnet yang berbeda (Aljburi 2016).

Router adalah perangkat lunak dan perangkat keras yang bertanggung jawab untuk mengirimkan paket data antara dua subnet. Pada kasus ini, *payment gateway* memiliki tanggung jawab seperti router dimana *payment gateway* berperan mencari rute kemana data akan dikirimkan.

Tujuan data yang dikirimkan dapat diketahui dengan memproses *request body* yang dikirimkan oleh aplikasi EDR. Salah satu *request body* terdapat *fields* tujuan bank penerima yang berbentuk kode dari masing – masing bank. Kode bank tersebut didefinisikan di *database* sehingga dari *fields* tersebut dapat disimpulkan *service* mana yang akan menerima data transaksi. Jika tujuan bank tidak ditemukan, maka *service* akan ditentukan berdasarkan prioritas yang di definisikan di *database.* Dengan hal tersebut maka EDR akan dikenakan biaya *admin* dikarenakan melakukan transfer atau transaksi ke berbeda bank.

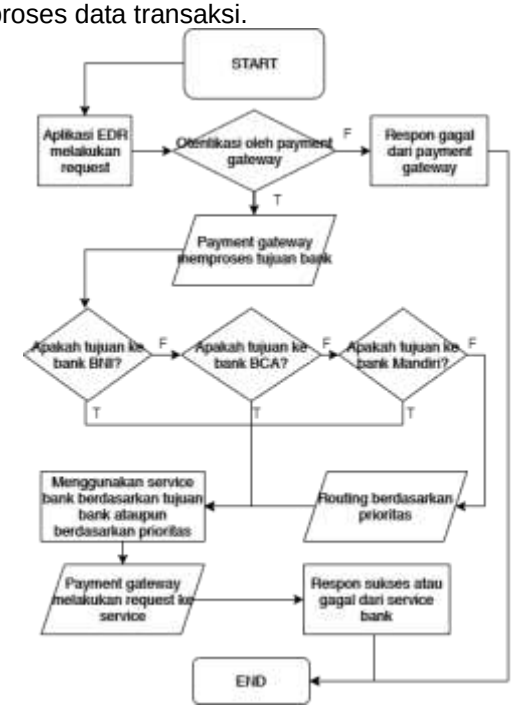
Dalam pertukaran data diperlukan keamanan untuk mengurangi ancaman terhadap *service*. Menurut (Gunawan and Rahmatulloh 2019), JSON Web Token merupakan salah satu keamanan pada *web service* mampu mengatasi permasalahan interoperabilitas, mampu berinteraksi dengan aplikasi lain baik berbasis web, desktop maupun mobile. Jika konten data dirubah secara *illegal*, maka *service* akan memberikan respon bahwa token tersebut tidak valid. Dengan demikian penggunaan JSON Web Token Authentication dianggap aman. Otentikasi jenis ini juga telah dipakai juga oleh peneliti sebelumnya dalam membuat WEB API (Sahoo, Janghel, and Samanta 2017).

Service payment gateway merupakan aplikasi yang dibuat untuk kepentingan perusahaan internal.Untuk menghindari service ini digunakan oleh aplikasi yang tidak dikenal, diperlukan juga *whitelist ip address* untuk memblok aplikasi tidak dikenali yang ingin menggunakan *service* ini.

2. Desain

Dalam memahami aplikasi, *developer* akan dipermudah dengan adanya desain serta perancangan sistem. Perancangan ini merupakan pondasi awal sebelulm memasuki tahap koding dalam pembuatan aplikasi. Salah satu desain yang paling umum dan mudah dipahami yaitu *flowchart* atau diagram alir.

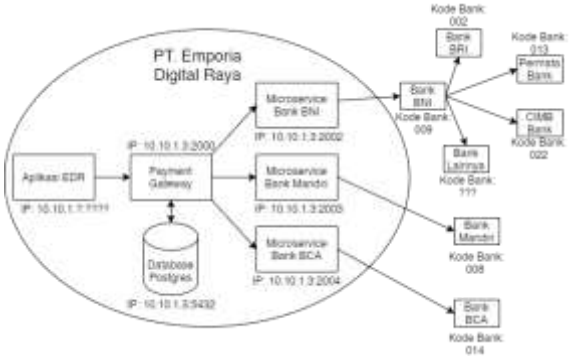
Diagram alir pada gambar 2 mendeskripsikan *payment gateway* dalam memproses data transaksi.



Gambar 2. Diagram Alir Payment Gateway

Standarisasi *request* perlu dilakukan dikarenakan setiap bank memiliki perbedaan spesifikasi dalam melakukan transaksi, maka dari itu konsep *microservice* diterapkan. Aplikasi akan dibagi menjadi beberapa bagian berdasarkan bank. Pembagian ini didasarkan pada perbedaan spesifikasi antar bank. Standarisasi dilakukan di setiap *service* yang akan terintegrasi di bank.

Setiap *service akan* dikenali dengan mengkonfigurasi *ip address local* dan *port* berbeda pada setiap *service. Microservice* bank akan mengidentifikasi *service* bank eksternal melalui bank kode yang telah didefinisikan di *database*. Pada kasus ini bank BNI telah disepakati oleh EDR memiliki prioritas utama yang berarti jika aplikasi EDR ingin melakukan transaksi diluar bank yang bekerja sama dengan EDR maka transaksi akan menggunakan bank BNI untuk diteruskan ke bank lain. Arsitektur *microservice* dapat dilihat pada gambar 3.



Gambar 3. Arsitektur *Microservice Payment Gateway*

Tabel 1. Tabel Mitra Di *Database*

Bank Code (PK)	Bank Name	Priority	Key Token	Identifier Local Service
009	PT Bank Negara Indonesia Tbk	1	e3c05ef5-8f9d-43d4-bd3e-8c0fecd50715	http://10.10.1.3:2002
014	PT Bank Central Asia Tbk	2	4923e7a5-1e20-4a0d-a684-ec74ed00baa3	http://10.10.1.3:2004
008	PT Bank Mandiri Tbk	3	8ceacc82-c957-417a-8311-6c5254431746	http://10.10.1.3:2003

Tabel 2. Tabel Routing Di *Database*

Bank Code (FK)	Usage	Method	Endpoint Local	Endpoint External
009	TRANSFER	POST	/api/bni/transfer	https://{{hostBNI}}/p2pl/payment/transfer
009	TRANSFER INTERBANK	POST	/api/bni/transferInterbank	https://{{hostBNI}}/p2pl/payment/interbank
014	TRANSFER	POST	/api/bca/transfer	https://{{hostBCA}}/banking/corporates/transfers
008	TRANSFER	POST	/api/mandiri/transfer	https://{{hostMandiri}}/gateway/TbwPayment/1.0/doPayment

Tabel 3. Tabel Client Di *Database*

ID (PK)	Service Name	Allowed IP	Key Token
1	IKI PPOB	139.99.60.117	09ec79db-a383-48c9-aee6-1ea516b64f50
2	IKI Modal	103.74.212.111	7c51d6b6-6b99-4896-9fc5-4425522ed2ce

Semua *service* bank yang telah melakukan kerja sama harus di definisikan di *database*. Tabel 1 merupakan isi dari *service* bank yang bekerja sama dengan EDR. Kolom Bank Code merupakan *primary key* dan bersifat unik. Nama dari setiap bank di simpan di kolom Bank Name. Saat tujuan bank transaksi tidak ditemukan maka *payment gateway* akan melakukan *routing* berdasarkan prioritas. Key Token merupakan key yang digunakan untuk membuat otentikasi JWT. Kolom Identifier Local merupakan alamat *host* agar *payment gateway* dapat melakukan HTTP *request* ke *microservice* lainnya.

Setelah *payment gateway* melakukan *routing* dan meneruskan data dari aplikasi EDR ke *microservice*, selanjutnya *microservice* akan membaca Tabel 2 untuk melakukan *request* ke bank tujuan dengan membaca kolom Endpoint External. Setelah menerima *response* dari bank maka *response* tersebut akan dikembalikan ke *payment gateway* yang selanjutnya akan diteruskan kembali ke aplikasi EDR.

Tabel 3 merupakan definisi dari aplikasi EDR yang akan menggunakan *payment gateway*. *Whitelist ip* dilakukan dengan membaca kolom Allowed IP pada tabel 3. Sedangkan kolom Key Token digunakan

untuk membuat otentikasi JWT oleh aplikasi EDR.

3. Implementasi / Koding

Implementasi menggunakan teknologi NodeJS dengan *framework* ExpressJS. NodeJS merupakan *runtime environment* javascript yang digunakan untuk membuat server. *Payment* gateway akan membaca rute yang diakses oleh aplikasi EDR. Selanjutnya *payment gateway* akan memproses *request* dan membaca *fields* beneficiaryBankCode untuk mengetahui bank mana yang akan dituju.

Setelah mendapatkan kode bank, *payment gateway* akan menjalankan *query berdasarkan kode bank* untuk mencari bank tujuan. Jika bank tujuan tidak ditemukan maka *payment gateway* akan menjalankan *query* berdasarkan prioritas. Prioritas yang terkecil yang akan diutamakan untuk melakukan transfer. Sintaks dapat dilihat pada gambar 4.

```
// NodeJS Syntax
router.post("/api/transfer", (req, res, next) => {
    let bankCode = req.body.beneficiaryBankCode; //800
    db.query(`
        SELECT
            TM.identifier_local,
            TR.endpoint_local
        FROM table_mitra AS TM
        INNER JOIN table_routing AS TR
        ON TM.bank_code = TR.bank_code
        WHERE TR.usage = 'TRANSFER'
        AND
        TM.bank_code = ${bankCode}
    `).then((result)=>{
      if(result == false){
        // if result not found, query by priority
        findByPriority();
      }else{
        // result.identifier_local = http://10.10.1.3:2002
        // result.endpoint_local = /api/bni/transfer
        // do request to microservice, url =
        //  result.identifier_local + result.endpoint_local
      }
    })
})
```

Gambar 4. Arsitektur *Microservice Payment Gateway*

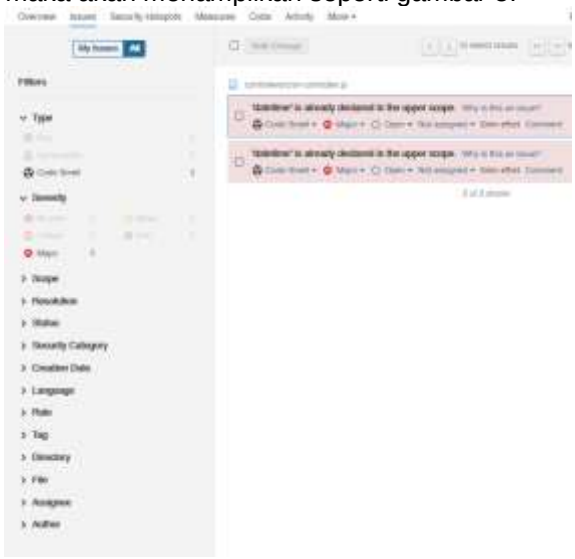
4. *Testing*

Pengujian merupakan bagian penting dari setiap proses pengembangan perangkat lunak di mana perusahaan mencurahkan banyak waktu dan tenaga.

Berbagai jenis teknik pengujian seperti pengujian unit, pengujian cakupan, pengujian struktural, pengujian statistik, pengujian interaksi kombinatorial, pengujian penetrasi, pengujian rekayasa perangkat lunak berbasis pencarian, pengujian hierarki, pengujian *bypass*, pengujian kompatibilitas lintas browser, memanfaatkan pengujian data sesi pengguna telah disajikan oleh berbagai peneliti dalam konteks pengujian aplikasi web (Lakshmi and Mallika 2017).

Pengujian unit adalah metode pengujian perangkat lunak di mana setiap unit kumpulan *source code* satu atau lebih modul program komputer bersama dengan data kontrol terkait, prosedur penggunaan, dan prosedur operasi diuji untuk menentukan apakah cocok untuk digunakan.

Popularitas alat untuk analisis kualitas perangkat lunak telah meningkat selama bertahun – tahun. SonarQube adalah salah satu alat dan penyedia yang paling banyak digunakan (Saarimaki et al. 2019). SonarQube merupakan alat otomatis untuk menganalisis kode, mendeteksi *bugs,* kode yang tidak sesuai dan kerentanan terhadap keamanan. Jika SonarQube menemukan isu dalam kode maka akan menampilkan seperti gambar 5.

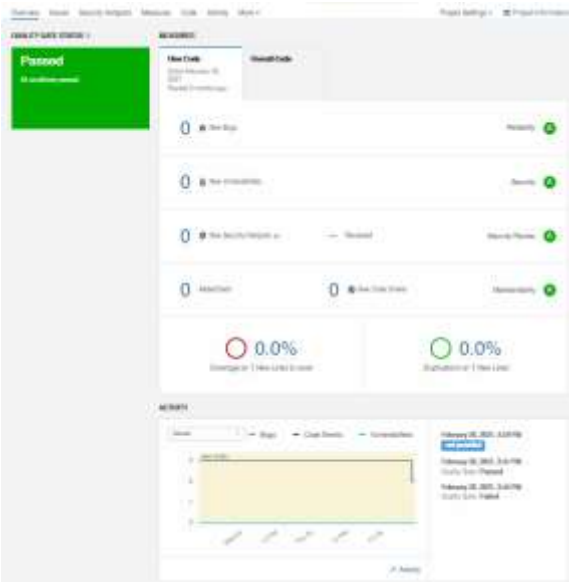


Gambar 5. Isu Kode *Payment Gateway*

5. Evaluasi

Setelah melakukan testing ada baiknya *developer* melakukan evaluasi dan perbaikan *bugs* jika ada. *Pada* hasil report SonarQube terhadap *payment gateway*, terdapat beberapa *code smells.* Dengan hal ini *developer* bertanggung jawab untuk memperbaiki *bugs.*

SonarQube juga memberikan rating A untuk *maintainability.* Selain itu, SonarQube memberikan saran untuk perbaikan kode. Hal ini berarti para *developer* tidak butuh banyak usaha ataupun waktu untuk mengatasi permasalahan *code smells.* Jika permasalahan sudah teratasi semua maka SonarQube akan memberikan *report* seperti gambar 6.



Gambar 6. Hasil *Report* SonarQube

3. Hasil dan Pembahasan

Dari penelitian yang telah dilakukan dapat diketahui bahwa pembuatan *payment gateway* dengan *microservice* dilakukan dengan membagi *microservice* berdasarkan bank yang terintegrasi. Dengan memanfaatkan metode *Agile*, setiap pembagian *microservice* akan dikelompokkan menjadi satu set ruang lingkup. Selanjutnya *service* yang telah dibagi, didaftarkan ke database agar *payment gateway* dapat mengenali semua *microservice* yang terintegrasi.

Pada setiap *microservice* diberikan pengamanan agar hanya *payment gateway* yang bisa melakukan transaksi. Selain pengamanan, setiap *endpoint* pada *microservice* didaftarkan sehingga *payment gateway* dapat mengenali kemana transaksi akan diteruskan.

Agar dikenali, aplikasi yang ingin menggunakan *payment* gateway didaftarkan ke database. IKI Modal merupakan salah satu aplikasi EDR yang terintegrasi dengan *payment gateway*. Gambar 7 merupakan fitur yang terdapat di aplikasi IKI Modal.

Gambar 7. IKI Modal Paylater Ajukan Talangin

Saat melakukan transaksi, sistem IKI Modal akan melakukan HTTP *request* ke *payment gateway.* Gambar 8 merupakan contoh *request body* dalam bentuk JSON yang dilakukan oleh sistem IKI Modal.

```
{
    "transactionId": "FDP29771029211",
    "accountNumber": "0847652590",
    "beneficiaryBankCode": "009",
    "beneficiaryAccountNumber": "0115471119",
    "amount": "2475000",
    "remark": "Paylater - FDP29771029211"
}
```

Gambar 8. *Request Body Payment Gateway*

Pada saat melakukan *request*, *payment gateway* akan melakukan otentikasi yang selanjut nya akan diteruskan ke *service* bank. Uang akan ditransfer melalui *escrow account* milik PT. Emporia Digital Raya ke rekening pengguna. *escrow account* adalah rekening bersama yang dikelola pihak ketiga (EDR). Setelah berhasil maka *payment gateway* akan merespon JSON seperti gambar 9.

```
{
    "statusCode": "000",
    "message": "Success transfer!",
    "data": {
        "referenceNumber": "FDP29771029211",
        "accountNumber": "084765290",
        "accountName": "IKIMODAL",
        "beneficiaryAccountNumber": "0115476119",
        "beneficiaryAccountName": "ERRY JULIO",
        "amount": "2475000"
    }
}
```


Gambar 9. *Response Body Payment Gateway*

Setelah *payment gateway* mersepon sukses ke IKI Modal, selanjutnya sistem IKI Modal akan memberikan notifikasi bahwa saldo telah masuk ke rekening pengguna seperti pada gambar 10.

Gambar 10. Saldo IKI Talangin

4. Kesimpulan

Penelitian ini dilakukan dengan membangun aplikasi *payment gateway* menggunakan arsitektur *microservice*, dimana peneliti dapat memelihara aplikasi berdasarkan pembagian bank. Hal ini dapat meminimalisir kesalahan dalam melakukan pengembangan maupun pemeliharaan aplikasi. Selain itu, jika *payment* gateway ingin melakukan integrasi dengan bank baru maka, peneliti hanya perlu membuat *microservice* baru.

Berdasarkan *report* pengujian unit yang telah dilakukan dengan SonarQube sebagaimana telah tampilkan pada Gambar, ditemukan beberapa isu pada kode berupa kesalahan penulisan sintaks. Setelah melakukan perbaikan dan pengujian ulang berdasarkan Gambar 6, tidak ditemukan adanya *bugs* maupun kerentanan pada keamanan. Dengan menggunakan konsep *routing* membuat aplikasi sangat fleksibel dalam melakukan transaksi ke sesama bank maupun berbeda bank. Pada Gambar 9 didapatkan respon sukses dari *microservice* yang selanjutnya akan diteruskan ke *payment gateway* dan akan diteruskan lagi ke aplikasi IKI Modal. Berdasarkan cek saldo Gambar 10, dapat disimpulkan bahwa transaksi yang dilakukan telah berhasil.

Selain *payment gateway* arsitektur ini dapat digunakan oleh proyek yang berskala besar untuk mengurangi kompleksitas dan mempermudah pemeliharaan pada suatu aplikasi.

Saat aplikasi sudah sangat banyak transaksi diperlukan optimalisasi terhadap performa agar aplikasi tidak menunggu respon terlalu lama. Dikarenakan *payment gateway* berkomunikasi dengan microservice menggunakan REST API maka protokol yang digunakan adalah HTTP.

Pada penelitian yang dilakukan (Permatasari and Dhika 2018), pertukaran data dengan HTTP dapat dioptimasi dengan menggunakan protokol *Message Queuing Telemetry Transportasi* (MQTT). Menurut penelitian tersebut, MQTT lebih cepat 9 kali dibandingkan dengan HTTP. MQTT menggunakan pola *publish* atau *subscribe* yang dirancang terbuka, ringan, dan mudah untuk diimplementasikan. Pola publish/subscribe memiliki kelebihan yang disebut *decouple*, maksudnya baik pengirim maupun penerima pesan tidak berkomunikasi secara langsung. Salah satu contoh aplikasi yang mengimplementasikan MQTT adalah RabbitMQ.

Dalam pengembangan atau penelitian berikutnya diharapkan untuk menggunakan aplikasi *message broker* seperti RabbitMQ atau Apache Kafka.

Referensi

Adi Pranata, Beni, Astria Hijriani, and Akmal Junaidi. 2018. “Perancangan Application Programming Interface (Api) Berbasis Web Menggunakan Gaya Arsitektur Representational State Transfer (Rest) Untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit.” *Jurnal Komputasi* 6(1): 33–42.

Aljburi, Dalya. 2016. “RoutingProtocols.” : 10.13140/RG.2.1.4310.4249.

Amazon Web Services. 2017. “Microservices on AWS.” *AWS Whitepaper* (September):

39.

Chandra, Yuri et al. 2020. "Implementasi Arsitektur Microservice Pada Aplikasi Web Pengajaran Agama Islam Home Pesantren." 1(November): 88–97.

Chandramouli, Ramaswamy. 2019. "NIST Special Publication 800-204 - Security Strategies for Microservices-Based Application Systems." *NIST Special Publication*. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf.

Francesco, Paolo Di, Paolo Di, Patricia Lago, and Ivano Malavolta. 2019. "Architecting with Microservices : A Systematic Mapping Study The Journal of Systems and Software Architecting with Microservices : A Systematic Mapping Study." *The Journal of Systems & Software* 150(April): 77–97. https://doi.org/10.1016/j.jss.2019.01.001.

Gunawan, Rohmat, and Alam Rahmatulloh. 2019. "JSON Web Token (JWT) Untuk Authentication Pada Interoperabilitas Arsitektur Berbasis RESTful Web Service." *Jurnal Edukasi dan Penelitian Informatika (JEPIN)* 5(1): 74.

Jiménez, Victor, Paulo Afonso, and Gabriela Fernandes. 2020. "Using Agile Project Management in the Design and Implementation of Activity-Based Costing Systems." *Sustainability (Switzerland)* 12(24): 1–23.

Lakshmi, D. Rajya, and S. Suguna Mallika. 2017. "A Review on Web Application Testing and Its Current Research Directions." *International Journal of Electrical and Computer Engineering* 7(4): 2132–41.

Mufrizal, Rizki, and Dina Indarti. 2019. "Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik." *Jurnal Nasional Teknologi dan Sistem Informasi* 5(1): 57–68.

Pahl, Claus, and Pooyan Jamshidi. 2020. "CLOSER 2020 - Proceedings of the 10th International Conference on Cloud Computing and Services Science." *CLOSER 2020 - Proceedings of the 10th International Conference on Cloud Computing and Services Science* 1(Closer): 137–46.

Permatasari, Claudia Felicia, and Harry Dhika. 2018. "Optimasi Jalur Transfer Data Dari HTTP Menjadi MQTT Pada IoT Menggunakan Cloud Services." *JISA(Jurnal Informatika dan Sains)* 1(2): 67–72.

Saarimaki, Nyyti, Maria Teresa Baldassarre, Valentina Lenarduzzi, and Simone Romano. 2019. "On the Accuracy of SonarQube Technical Debt Remediation Time." *Proceedings - 45th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2019* (August): 317–24.

Sahoo, Pravat, Nilesh Kumar Janghel, and Debabrata Samanta. 2017. "Securing WEB API Based on Token Authentication." : 4–7.

Setya Budi, Cahyanto, and Adam Mukharil Bachtiar. 2018. "Implementasi Arsitektur Microservices Pada Backend Comrades." *Program Studi Teknik Informatika, Universitas Komputer Indonesia*: 1–6.

Zhao, J. T., S. Y. Jing, and L. Z. Jiang. 2018. "Management of API Gateway Based on Micro-Service Architecture." *Journal of Physics: Conference Series* 1087(3).