Jurnal Teknologi Terpadu

https://journal.nurulfikri.ac.id/index.php/JTT
ISSN : 2477-0043 ISSN ONLINE : 2460-7908

IMPLEMENTASI ARSITEKTUR *MICROSERVICE* PADA *BACK END* SISTEM INFORMASI ATLANTAS BERBASIS *WEBSITE*

Calvin Seviro Bima Sakti[1], Indra Hermawan[2]
[1]Program Studi Teknik Informatika
[2]Program Studi Teknik Multimedia dan Jaringan
Jurusan Teknik Informatika dan Komputer
Politeknik Negeri Jakarta
Depok, Jawa Barat, Indonesia
calvin.sakti.tik16@mhsw.pnj.ac.id, indra.hermawan@tik.pnj.ac.id

Abstract

The number of traffic violations that occurred in the Depok area from 2017 to 2019 has fluctuated, from 2017 as many as 33,600 cases, 2018 as many as 52,100 cases and 2019 as many as 42971 cases. In each year an increase in violations occurs and the ability of applications to distribute data will decrease. This happened to the Traffic Unit of the Depok Metro Police, with the increasing number of handling data on accidents and data on violations that occurred in Depok City, of course, it must be balanced with an increase in the system as an online reporting medium. From the above explanation, it can be concluded that to solve this process, it is necessary to have a new architecture, namely the microservice. Microservice allows the development of software functions to be broken down into small, focused service parts, making the service have resource capabilities that can be precisely managed and dividing application functionality into small and interconnected services into a single business application process. This architecture integrates data between units, distributes data related to cases of traffic violations or accidents that occur in real time by local police officers, and provides information on traffic violations and specific accidents.

Keyword: *Offense, Microservice, Traffic, Service*

Abstrak

Jumlah pelanggaran lalu lintas yang terjadi di Wilayah Depok dari tahun 2017 sampai 2019 mengalami fluktuasi, dari tahun 2017 sebanyak 33.600 kasus, 2018 sebanyak 52.100 kasus dan 2019 sebanyak 42971 kasus. Dalam setiap tahunnya peningkatan pelanggaran terjadi dan kemampuan aplikasi dalam pendistribusian data akan menurun. Hal ini terjadi pada Satuan Lalu Lintas Polres Metro Depok, dengan semakin banyaknya penanganan pencatatan data kecelakaan dan data pelanggaran yang terjadi di Kota Depok tentunya harus diimbangi dengan peningkatan sistem sebagai media pelaporan secara *online*. Dari penjelasan diatas dapat disimpulkan untuk memecahkan proses tersebut perlu adanya sebuah arsitektur baru yaitu *microservice*. *Microservice* memungkinkan pengembangan fungsi perangkat lunak dipecah menjadi bagian – bagian *service* yang kecil dan terfokus menjadikan *service* akan memiliki kemampuan *resource* yang bisa diatur dengan tepat dan membagi fungsionalitas aplikasi menjadi layanan yang kecil dan saling berhubungan menjadi satu kesatuan bisnis proses aplikasi. Arsitektur ini mengintegrasikan data antar satuan dan mendistribusikan data terkait adanya kasus pelanggaran lalu lintas atau kecelakaan yang terjadi secara *realtime* oleh petugas polisi setempat serta menyajikan informasi data pelanggaran lalu lintas dan kecelakaan yang spesifik.

Kata kunci: Pelanggaran, *Microservice*, Lalu Lintas, *Service*

1. PENDAHULUAN

Sistem transportasi merupakan hal krusial dan inti dari suatu kota besar yang memiliki penduduk dan aktivitas yang banyak. Pergerakan penduduk dan aktifitas ekonomi sebagian besar dilayani oleh angkutan umum. Sudah sangat banyak kasus pelanggaran lalu lintas yang dilakukan oleh pengguna jalan yang cenderung mengakibatkan timbulnya kecelakaan dan kemacetan lalu lintas yang semakin meningkat.

Berdasarkan data Satlantas Polres Kota Depok banyaknya kasus pelanggaran lalu lintas dalam bentuk bukti pelanggaran atau tilang pada tahun 2017 sebanyak 33600 kasus. Pada tahun 2018 terjadi kenaikan pelanggaran lalu lintas menjadi 52100 kasus dan pada tahun 2019 terjadi penurunan pelanggaran lalu lintas menjadi 42971 kasus.

Diterima 5 Oktober 2020, Direvisi 23 Desember 2020, Diterima untuk publikasi 30 Desember 2020

Pelanggaran tersebut terjadi justru pada jam-jam sibuk dimana aktifitas masyarakat di jalan raya meningkat. Peningkatan tersebut menjadi *challenge* bagi kepolisian agar mampu menerapkan suatu sanksi yang mendidik dan memiliki efek jera. Tetapi hal ini disampingkan oleh anggota kepolisian serta sipil demi tercapainya kepentingan bersama tanpa mengikuti prosedur yang ada [1].

Berdasarkan data Satlantas Polres Kota Depok pada tahun 2017 sebanyak 253 kasus kecelakaan, pada tahun 2018 sebanyak 217 kasus kecelakaan dan pada tahun 2019 sebanyak 428 kasus kecelakaan. Dari tahun 2017 ke 2018 terjadi penurunan kasus sebanyak 36 tetapi dari 2018 ke 2019 mengalami kenaikan kasus sebanyak 211 kasus. Pesatnya perkembangan teknologi pada zaman ini telah memberikan banyak manfaat bagi manusia. Hal ini terjadi pada Satuan Lalu Lintas Polres Metro Depok, dengan semakin banyaknya penanganan pencatatan data kecelakaan dan data pelanggaran yang terjadi di Kota Depok tentunya harus diimbangi dengan peningkatan sistem sebagai media pelaporan secara *online*. Berdasarkan pernyataan dari Staff Unit Lalu Lintas semakin bertambahnya proses pencatatan data kecelakaan yang direkap tiap bulan dari staff unit laka lantas kepada staff OP satuan lalu lintas dan juga setiap unit pada satuan lalu lintas harus berdiri pada aplikasi atau sistemnya masing-masing.

Dari penjelasan diatas dapat disimpulkan untuk memecahkan proses tersebut perlu adanya sebuah arsitektur baru yaitu *microservice*. *Microservice* memungkinkan pengelolaan *service – service* yang ada pada perangkat lunak secara terpisah, dengan begitu jika dilakukan pengembangan *service* maka *service* lain tidak akan terganggu. Selain itu pengembangan kapasitas antar *service* bisa dibedakan sehingga *resource* yang dipakai bisa tepat.Oleh karena itu, penelitian ini dilakukan pengembangan sistem ini dengan konsep arsitektur *microservice*. Sistem ini diperlukan untuk mengintegrasikan data antar satuan dan mendistribusikan data terkait adanya kasus pelanggaran lalu lintas atau kecelakaan yang terjadi secara *realtime* oleh petugas polisi setempat serta menyajikan informasi data pelanggaran lalu lintas dan kecelakaan yang spesifik.

2. TINJAUAN PUSTAKA

2.1 Sistem Informasi

Sistem Informasi adalah sebuah sistem untuk memproses informasi seperti, mengirimkan, menangkap, memperoleh kembali, mengubah, menyimpan dan menampilkan informasi. Sebuah sistem kerja adalah sebuah sistem di mana partisipasi manusia dan/atau mesin melakukan aktivitas menggunakan teknologi, informasi, dan sumber daya lainnya untuk menghasilkan yang dituju [2].

2.2 *Web Service*

Web Service adalah jembatan antar aplikasi yang berhubungan yang dapat dipanggil dan diakses menggunakan format pertukaran data sebagai pengirim/penerima pesan. *Web Service* digunakan sebagai tempat menyediakan layanan data kepada aplikasi. Format yang digunakan dalam mengirim atau menerima pesan adalah JSON atau XML agar data dapat diakses oleh sistem lain walaupun berbeda platform dan bahasa pemograman [3].

2.3 *Representational State Transfer*

Representational State Transfer (REST) adalah sebuah arsitektur perangkat lunak untuk mengkomunikasikan data dan *website* dalam bentuk protokol HTTP. *Application Programming Interfaces* (API) adalah jembatan komunikasi antar klien dan server untuk mengimplementasikan fitur – fitur yang terdapat pada aplikasi. Sistem yang mengikuti prinsip REST disebut RESTFUL API [4]. RESTFUL API terdiri dari beberapa komponen yaitu :

a. URL *Design*: RESTFUL API diakses menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten sangat diperlukan agar dapat dimengerti penggunanya, URL API biasa disebut juga *end point*. Contoh pemanggilan URL API yang baik adalah seperti berikut: *users, users/1, users/export/1*

b. HTTP *Verbs*: Ketika *user* ingin melakukan *request,* metode ini dijalankan agar server mengerti apa yang *user* ingin dapatkan. Metode dalam *request* terdiri dari : *GET, POST, PUT, DELETE*.

c. *Response Code*: Kode yang telah menjadi standar dalam menginformasikan hasil *request* kepada *user*. Terdapat 3 kelompok kode yang paling sering digunakan di REST API yaitu: Kode 2## yaitu permintaan yang dilakukan berhasil, kode 4## yaitu permintaan mengalami *error* pada klien dan kode 5## yaitu permintaan mengalami kesalahan pada server.

d. *Format Response*: Setiap permintan yang dilakukan oleh *user* akan menerima *output* respon dari server. Respon tersebut berupa data XML atau JSON. Setelah mendapatkan data respon tadi kemudian *user* bisa menggunakannya dengan cara memecah data dan diolah sesuai keperluan.

2.4 *Microservice*

Microservice adalah kumpulan beberapa proses yang berkomunikasi antar *service* lain untuk membentuk sebuah aplikasi yang kompleks terhadap bahasa API apapun. *Microservice* merupakan pengembangan lanjutan dari *Service-oriented Architecture* karena *microservice* merupakan sistem yang terdiri dari komponen servis-servis blok, kecil, terpisah dan fokus pada tugas-tugasnya atau bekerja secara metode modular, autonomous untuk tujuan masing-masing namun terkoneksi satu sama lain secara beraturan. Dengan arsitektur ini maka tercapai satu tujuan utama tertentu dalam pengembangan *software* [5].

2.5 Lumen Laravel

Laravel adalah *framework* MVC yang mampu meningkatkan kualitas dan produktivitas perangkat lunak serta mengurangi biaya pengembangan dan perbaikan. [6] Lumen adalah bagian dari laravel yang biasa disebut *micro-*

framework yang dikhususkan untuk pembuatan API. Berbeda dengan laravel yang digunakan untuk pembuatan *full-stack website*.

2.6 Metode *Rapid Application Development*

Metode *Rapid Application Development* (RAD) adalah proses pengembangan perangkat lunak yang proses pembuatan aplikasinya dalam waktu yang singkat. Metode *iterative* yang digunakan pada metode ini yaitu *working model* (model bekerja) yang dibangun pada tahap awal pengembangan dengan tujuan menetapkan *requirement analysis* (analisis kebutuhan) [7]. Berikut pada gambar 1 merupakan tahapan pengembangan aplikasi dari metode *rapid application development*:

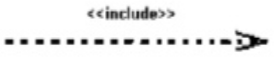
Gambar 1. Metode *Rapid Application Development* [8]

a. *Requirement Planning*: Mengidentifikasikan kebutuhan informasi dan masalah yang dihadapi untuk menentukan tujuan dibuatnya sistem, kendala dan juga alternatif pemecahan masalah.
b. *Design Workshop*: Mengidentifikasi solusi alternatif dan memilih solusi terbaik adalah langkah dari kebutuhan desain. Pada bagian ini diperlukan membuat desain pemograman dan proses bisnis dalam bentuk pemodelan dalam arsitektur informasi yang telah didapatkan dari data-data.
c. *Implementation*: Tahap ini proses pembuatan aplikasi menggunakan bahasa pemograman yang diwujudkan dalam bentuk program atau unit program.

2.7 *Use Case Diagram*

Use Case atau diagram *use case* adalah pemodelan untuk menggambarkan hubungan antara sistem dengan aktor. Diagram ini menggambarkan himpunan *use case* dan aktor-aktor. Diagram ini sangat penting dalam mengorganisasi dan memodelkan perilaku sistem yang diharapkan oleh pengguna [9]. Berikut pada Tabel 1 merupakan arti simbol pada *use case* diagram.

Tabel 1. Simbol *Use Case* Diagram [10]

Simbol	Deskripsi
	Yang berinteraksi dengan sistem informasi yang akan dibuat disebut dengan aktor.
	Fungsionalitas yang disediakan sistem sebagai unit yang saling bertukar pesan antar unit atau aktor yang disebut *Use Case*.
	Asosiasi adalah komunikasi antara aktor dan *use case* yang berpartisipasi pada *use case* yang memiliki interaksi dengan aktor.
	*Include* adalah relasi *use case* yang bergantung dengan *use case yang* lain untuk menjalankan fungsinya.
	Ekstensi adalah relasi *use case* yang dapat berdiri sendiri walaupun tambahan *use case* lain.
	Generalisasi adalah hubungan umum-khusus antar dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umumnya dari lainnya.

2.8 *Black Box Testing*

Black Box Testing adalah salah satu metode pengujian pada pengembangan sistem atau aplikasi yang dilakukan untuk memeriksa apakah aplikasi berjalan pada fungsional dengan baik dan benar tanpa harus mengetahui bisnis proses yang terjadi. Pengujian *blackbox* dilakukan dengan membuat *test case* yang berupa form test *input* dan *ouput* yang diharapkan pada fungsional aplikasi. Pengujian bisa dilakukan pada aplikasi yang tingkat granulitas rendah sehingga memungkinkan memakan waktu yang relatif tidak lama [11].

3. METODE PENELITIAN

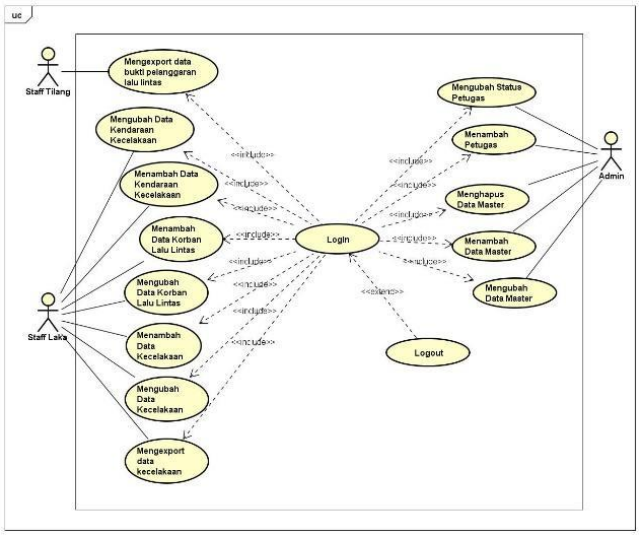
Metode penelitian digunakan sebagai panduan dalam mengerjakan penelitian agar lebih sistematis, teratur dan terarah. Pembuatan metode penelitian berdasarkan model pengembangan *Rapid Application Development.* Tahapan-tahapan dalam penelitian ini dapat dilihat sebagai berikut:

3.1 Analisis Kebutuhan

Tahap ini dilakukan analisis kebutuhan mengenai sistem informasi yang akan dibuat. Hasil dari analisis tersebut akan menjadi kebutuhan data, fungsional, dan non-fungsional.

3.2 Desain Sistem

Pada tahap ini dilakukan perancangan dan desain aplikasi sistem informasi atlantas. Perancangan dan desain menggunakan *Use Case* Diagram dan alur model. Berikut pada Gambar 2. menunjukkan diagram *use case* yang berguna sebagai informasi kebutuhan pengguna:



Gambar 2. *Use Case* Diagram *microservice*

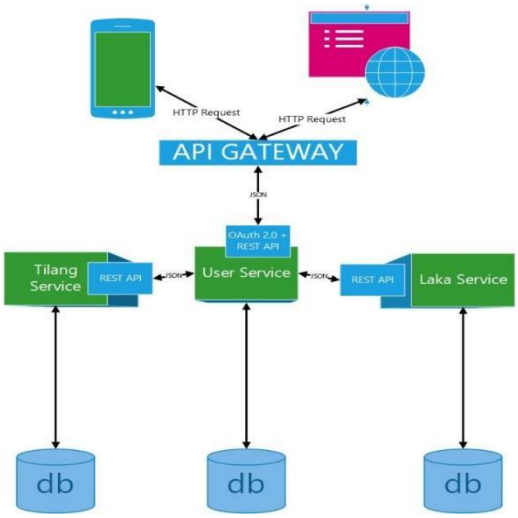
Dari hasil analisis kebutuhan dan kebutuhan aktor pada *use case* diagram lalu dikembangan menjadi model arsitektur sebagai rancangan awal dari sebuah sistem. Berikut pada Gambar 3. adalah model arsitektur sistem yang dibuat.

Pada hasil Gambar 3 menghasilkan 3 *microservices* yang terdiri dari :

a. *User Service*: *service* yang menangani bagian akun seperti petugas lapangan, staff dan admin, sekaligus bagian peroutingan *request* masuk dan keluar dari aplikasi ke *microservice*. Validasi akses ke *microservice* tilang dan laka dilakukan disini dengan menggunakan *OAuth 2.0*.

b. Tilang *Service* : *service* yang bertugas pada proses bisnis satuan tilang seperti *store* surat tilang, kendaraan yang melanggar, orang yang dilanggar serta pasal pelanggaran lalu lintas.

c. Laka *Service* : *service* yang bertugas pada proses bisnis satuan kecelakaan seperti mencatat kejadian kecelakaan, surat laporan kecelakaan, kendaraan yang terlibat serta mencatat korban atau pelaku kecelakaan.



Gambar 3. Arsitektur m*icroservice* sistem

3.3 Pembuatan Sistem

Pembuatan sistem merupakan tahapan pembuatan berdasarkan desain yang telah disepakati. Kode aplikasi dibuat dengan menggunakan *framework* lumen. Untuk implementasi microservice yang dibuat dapat dilihat pada beberapa penjelasan tiap *service* dibawah ini berdasarkan dari penjelasan Gambar 3.

3.3.1 Kecelakaan *Service*

Pada Tabel 2. merupakan *controller master* kendaraan yang berisi beberapa *method* dalam kelola data yang berhubungan dengan *master* data kendaraan. *Controller* tersebut akan digunakan dalam proses pencatatan jenis data kendaraan kecelakaan.

Tabel 2. Master kendaraan *controller*

***Controller Master* Kendaraan**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data jenis kendaraan	*GET*	/MKendaraan
Menampilkan detail jenis kendaraan	*GET*	/MKendaraan/{id}
Menambah data jenis kendaraan	*POST*	/MKendaraan
Menghapus data jenis kendaraan	*DELETE*	/MKendaraan/{id}
Mengubah data jenis kendaraan	*PUT*	/MKendaraan/{id}

Pada Tabel 3. merupakan *controller master* tabrakan yang berisi *method-method* dalam kelola data yang berhubungan dengan *master* data tabrakan. *Controller* tersebut akan digunakan dalam dalam proses pencatatan jenis data tabrakan kecelakaan.

Tabel 3. *Master* tabrakan *controller*

***Controller Master* Tabrakan**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data jenis tabrakan	*GET*	/MTabrakan
Menampilkan detail jenis tabrakan	*GET*	/MTabrakan/{mtabrakan}
Menambah data jenis tabrakan	*POST*	/MTabrakan
Menghapus data jenis tabrakan	*DELETE*	/MTabrakan/{mtabrakan}
Mengubah data jenis tabrakan	*PUT*	/MTabrakan/{mtabrakan}

Pada Tabel 4. merupakan *controller master* kondisi yang berisi *method-method* dalam kelola data yang berhubungan dengan *master* data kondisi korban. *Controller* tersebut akan digunakan untuk proses pencatatan data kondisi korban kecelakaan.

Tabel 4. Master kondisi *controller*

***Controller Master* Kondisi**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data kondisi korban	*GET*	/MKondisi
Menampilkan detail kondisi korban	*GET*	/MKondisi/{mkondisi}
Menambah data kondisi korban	*POST*	/MKondisi
Menghapus data kondisi korban	*DELETE*	/MKondisi/{mkondisi}
Mengubah data kondisi korban	*PUT*	/MKondisi/{mkondisi}

Pada Tabel 5. merupakan *controller master* jalan yang berisi *method-method* dalam kelola data yang berhubungan dengan *master* data nama jalan. *Controller* tersebut digunakan dalam proses pencatatan data kecelakaan khususnya posisi kecelakaan berada dengan mengambil nama jalan.

Tabel 5. Master jalan *controller*

***Controller Master* Jalan**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data nama jalan	*GET*	/jalan
Menampilkan detail nama jalan	*GET*	/jalan/{jalan}
Menambah data nama jalan	*POST*	/jalan
Menghapus data nama jalan	*DELETE*	/jalan/{jalan}
Mengubah data nama jalan	*PUT*	/jalan/{jalan}

Pada Tabel 6. merupakan *class* kecelakaan *controller*. *Controller* tersebut berisi *method-method* yang digunakan untuk melakukan proses pencatatan data kecelakaan yang terjadi.

Tabel 6. Kecelakaan *controller*

***Controller* Kecelakaan**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data kecelakaan	*GET*	/kcl
Menampilkan detail kecelakaan	*GET*	/kcl/{kcl}
Menambah data kecelakaan	*POST*	/kcl
Mengubah data kecelakaan	*PUT*	/kcl/{kcl}
Export data kecelakaan	*PUT*	/kcl/export-{id}

Pada Tabel 7. merupakan *class* kendaraan *controller*. *Controller tersebut* berisi *method-method* yang digunakan untuk melakukan proses pencatatan data kendaraan yang terlibat kecelakaan. Pada Tabel 8. merupakan *class* korban *controller*. *Controller* tersebut berisi *method-method* yang digunakan untuk melakukan proses pencatatan data korban yang terlibat kecelakaan.

Tabel 7. Kendaraan *controller*

***Controller* Kendaraan**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data kendaraan	*GET*	/kstnk
Menampilkan detail kendaraan	*GET*	/kstnk/{kstnk}
Menambah data kendaraan	*POST*	/kstnk
Mengubah data kendaraan	*PUT*	/kstnk/{kstnk}

Tabel 8. Korban controller

***Controller* Korban**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data korban	*GET*	/orang
Menampilkan detail korban	*GET*	/orang/{orang}
Menambah data korban	*POST*	/orang
Mengubah data korban	*PUT*	/orang/{orang}

3.3.2 Tilang *Service*

Pada Tabel 9. merupakan *controller* surat operasi yang berisi *method-method* yang berhubungan dengan proses pendataan surat operasi. *Controller* tersebut akan digunakan untuk tindakan pelanggaran lalu lintas secara represif, preventif dan preemtif.

Tabel 9. Surat Operasi *Controller*

***Controller* Surat Operasi**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data surat operasi	*GET*	/soperasi
Menampilkan detail surat operasi	*GET*	/soperasi/{soperasi}
Menambah data surat operasi	*POST*	/soperasi
Mengubah data surat operasi	*PUT*	/soperasi/{soperasi}

Pada Tabel 10. merupakan *controller* tilang. *Controller* tersebut berisi *method* untuk menampilkan informasi semua dan detail pada pelanggar lalu lintas.

Tabel 10. Tilang *Controller*

***Controller* Tilang**		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Menampilkan data tilang	*GET*	/tilang
Menampilkan detail tilang	*GET*	/tilang /{tilang}

3.3.3 *User Service*

User service mempunyai fungsi dalam proses *login*, menambah dan mengubah status aktor. *Service* ini juga mempunyai peran sebagai API *Gateway* yang mengatur management API dari *service* kecelakaan dan tilang, *merge*

beberapa API serta autentikasi API . Dikirimkan berbentuk REST API menggunakan protokol HTTP *Request*. Berikut Tabel 11. merupakan *controller user service*.

Tabel 11. *User controller*

Controller User		
Kebutuhan *Microservice*	***Method Rest***	**URL REST**
Login	*POST*	*/login*
Mengambil data petugas	*GET*	*/users*
Mengambil detail petugas	*GET*	*/users/{id}*
Menambah data petugas	*POST*	*/users*
Mengubah status petugas	*PUT*	*/users/{id}*
Generate Token	*POST*	*/oauth/token*

4. HASIL DAN PEMBAHASAN

Pada bagian ini akan dijelaskan mengenai pengujian *black box* dan pengujian integrasi. Pengujian ini ditujukan kepada pengguna aplikasi khususnya bagian *back end* sistem informasi atlantas.

4.1 Pengujian *Black Box*

Pengujian *black box* digunakan agar dapat mengetahui fungsional sistem dan kesesuaian dari data masukan hingga keluaran yang dihasilkan dari sistem yang telah dirancang. Dari 69 skenario yang dijalankan dengan presentasi keberhasilan sebesar 100%. Berikut Tabel 12. merupakan ringkasan table sebagai informasi item apa saja yang dilakukan pengujian :

Tabel 12. Rencana Pengujian *Black Box Testing*

Item Uji	**Butir Uji**	**Jenis Pengujian**
Autentfikasi	Melakukan *Login*	*Black Box*
	Melakukan *Logout*	*Black Box*
Menu Data *Master*	Admin Melihat Data *Master*	*Black Box*
	Admin Menambah Data *Master*	*Black Box*
	Admin Mengubah Data *Master*	*Black Box*
	Admin Menghapus Data *Master*	*Black Box*
Menu Data Petugas	Admin Dapat Melihat Data Petugas	*Black Box*
	Admin Dapat Menambah Data Petugas	*Black Box*
	Admin Dapat Mengubah Data Petugas	*Black Box*
Menu Data Kecelakaan	Staff Kecelakaan Dapat Melihat Data Kecelakaan	*Black Box*
	Staff Kecelakaan Dapat Menambah Data Kecelakaan	*Black Box*
	Staff Kecelakaan Dapat Mengubah Data Kecelakaan	*Black Box*
	Staff Kecelakaan Dapat Mengexport Data Kecelakaan	*Black Box*
Menu Data Kendaraan	Staff Kecelakaan Dapat Melihat Data Kendaraan yang terlibat kecelakaan	*Black Box*
	Staff Kecelakaan Dapat Menambah Data Kendaraan yang terlibat kecelakaan	*Black Box*
	Staff Kecelakaan Dapat Mengubah Data Kendaraan yang terlibat kecelakaan	*Black Box*
Menu Data Korban	Staff Kecelakaan Dapat Melihat Data Korban yang terlibat kecelakaan	*Black Box*
	Staff Kecelakaan Dapat Menambah Data Korban yang terlibat kecelakaan	*Black Box*
	Staff Kecelakaan Dapat Mengubah Data Korban yang terlibat kecelakaan	*Black Box*
Menu Data Surat Operasi	Staff Tilang Dapat Melihat Data Surat Operasi	*Black Box*
	Staff Tilang Dapat Menambah Surat Operasi	*Black Box*
	Staff Tilang Dapat Mengubah Data Surat Operasi	*Black Box*

4.2 Pengujian Integrasi

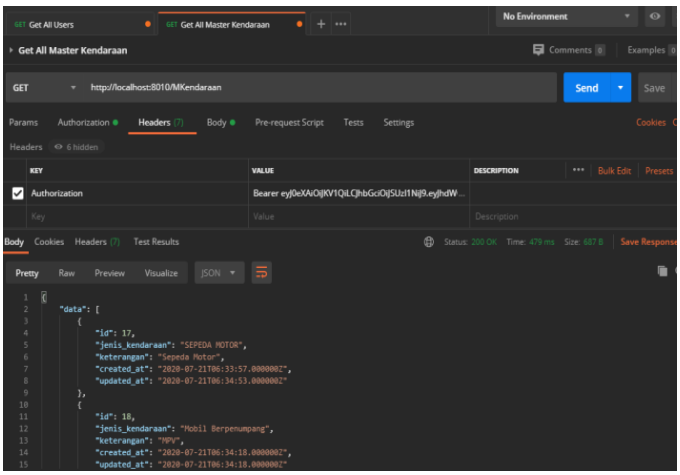
Pengujian integrasi digunakan untuk mengetahui status *log* melalui layanan API pada saat pengujian dan menampilkan *response* yang didapat pada saat melakukan *request*. Prosedur pengujian ini melibatkan 3 *service* yaitu API *Gateway*, Kecelakaan *Service* dan Tilang *Service* dengan menggunakan 2 jenis pengujian yaitu kedua *service* dinyalakan dan salah satu *service* dimatikan / *offline*. Dari 7 skenario pengujian, 2 skenario menunjukkan hasil *response code 500*. Hasil dari presentase keberhasilan sebesar 71%. Berikut hasil pengujian integrasi yang dilakukan :

4.2.1 Integrasi API *Gateway* ke Kecelakaan *Service*

Berikut pengujian dengan kondisi 2 *service* yang *online* pada Gambar 4. Pada Gambar 4. menunjukkan bahwa *API gateway* melakukan request ke kecelakaan *service* untuk mendapatkan data jenis kendaraan. Berikut rincian REST API dari gambar diatas :

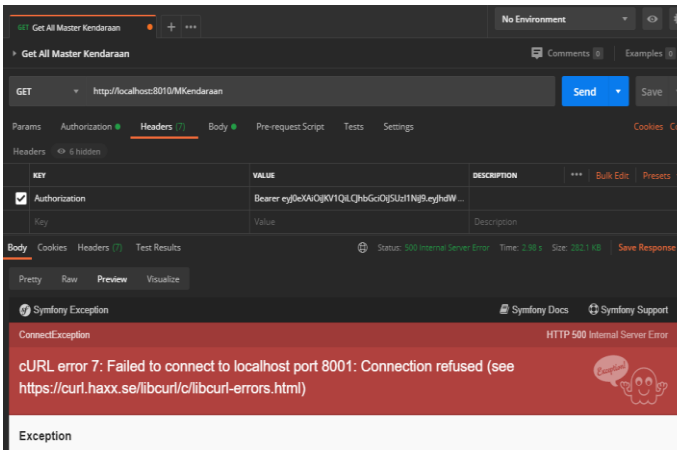
HTTP Request :
a. *Method : GET*
b. *URI : http://localhost:8010/MKendaraan*
c. *Header : client_credentials*

Response :
a. *Response Code :* 200
b. *Time : 479ms*
c. *Response Body* : data berupa id, jenis_kendaraan dan keterangan



Gambar 4. Integrasi API *Gateway* ke Kecelakaan *Service*

Berikut pengujian dengan kondisi kecelakaan *service* yang *offline* pada Gambar 5.



Gambar 5 Integrasi API *Gateway* ke Kecelakaan *Service* (*Offline*)

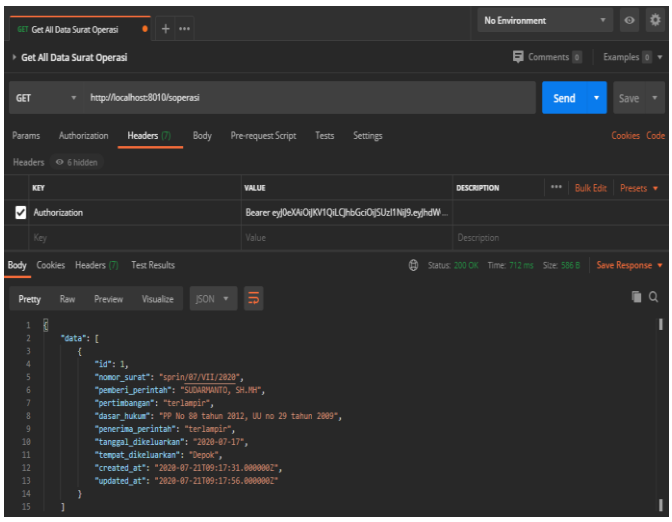
Pada Gambar 5. menunjukkan bahwa *API gateway* melakukan request ke kecelakaan *service* untuk mendapatkan data jenis kendaraan jika kondisi kecelakaan service dalam keadaan *offline*. Berikut rincian *output* REST API pada gambar diatas :

HTTP Request :
a. *Method : GET*
b. *URI : http://localhost:8010/MKendaraan*
c. *Header : client_credentials*

Response :
a. *Response Code :* 500
b. *Time : 2.98 s*
c. *Response Body* : *Error* menampilkan pesan *cURL error 7 failed connect to localhost:8001*.

4.2.2 Integrasi API *Gateway* ke Tilang *Service*

Berikut pengujian dengan kondisi 2 *service* yang *online* pada Gambar 6.



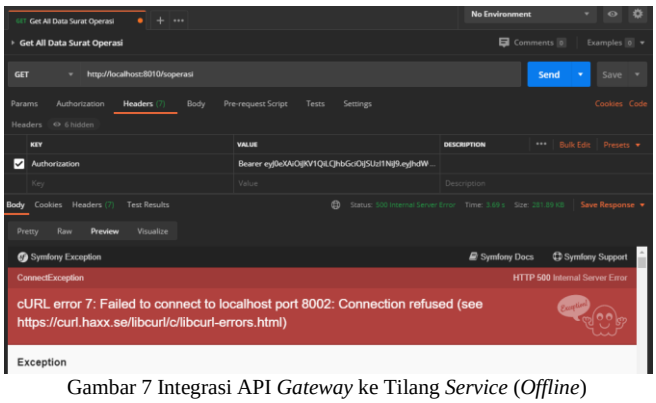
Gambar 6. Integrasi API *Gateway* ke Tilang *Service*

Pada Gambar 6. menunjukkan bahwa *API gateway* melakukan request ke tilang *service* untuk mendapatkan data surat operasi tilang. Berikut rincian *output* REST API dari gambar diatas :

HTTP Request :
a. *Method : GET*
b. *URI : http://localhost:8010/soperasi*
c. *Header : client_credentials*

Response :
a. *Response Code :* 200
b. *Time : 712ms*
c. *Response Body* : data berupa id, nomor_surat, pemberi perintah, pertimbangan, dasar_hukum, penerima_perintah, tanggal_dikeluarkan dan tempat_dikeluarkan.

Berikut pengujian dengan kondisi tilang *service* yang *offline* pada Gambar 7.



Gambar 7 Integrasi API *Gateway* ke Tilang *Service* (*Offline*)

Pada Gambar 7. menunjukkan bahwa *API gateway* melakukan request ke kecelakaan *service* untuk mendapatkan data jenis kendaraan jika kondisi kecelakaan service dalam keadaan *offline*. Berikut rincian *output* REST API gamabr diatas :

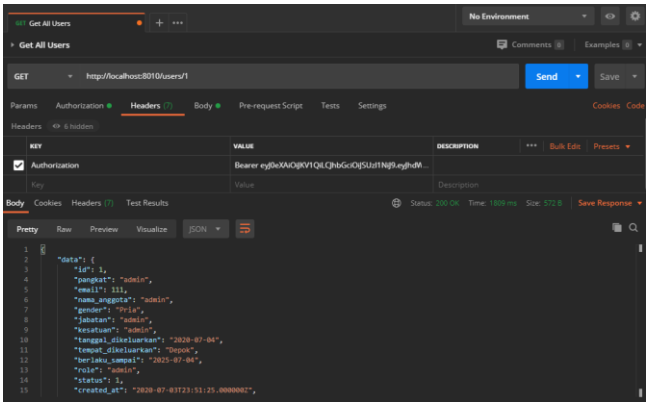
HTTP Request :
a. *Method : GET*
b. *URI : http://localhost:8010/soperasi*
c. *Header : client_credentials*

Response :

a. *Response Code :* 500
b. *Time : 3.69 s*
c. *Response Body* : *Error* menampilkan pesan *cURL error 7 failed connect to localhost:8002.*

4.2.3 Integrasi API *Gateway* ke *User Service*

Berikut pengujian dengan menggunakan 2 *service online* pada gambar 8.



Gambar 8 Integrasi API *Gateway* ke *User Service*

Pada Gambar 8. menunjukkan bahwa API *gateway* melakukan request ke *user service* untuk mendapatkan detail data petugas. Berikut rincian *output* REST API gambar diatas :

HTTP Request :
a. *Method : GET*
b. *URI : http://localhost:8010/users/1*
c. *Header : client_credentials*
Response :
a. *Response Code :* 200
b. *Time : 1809ms*
c. *Response Body* : data petugas

5. KESIMPULAN

Berdasarkan hasil implementasi arsitektur *microservice* pada *back end* sistem informasi atlantas berbasis *website*, maka dapat ditarik kesimpulan :

1. Pembuatan sistem *back end* pada sistem informasi atlantas berhasil dibuat dengan konsep arsitektur *microservice* yang memberikan informasi kasus pelanggaran dan kebutuhan setiap unit akan saling terintegrasi.

2. Dalam membangun sistem *back end* menggunakan arsitektur *microservice* maka kebutuhan fungsional harus dipecah menjadi beberapa *microservice* yang dipanggil melalui API *Gateway*.

3. Berdasarkan hasil pengujian *black box* didapatkan presentase sebesar 100% bahwa sistem *backend* menggunakan arsitektur *microservice* sudah berjalan sesuai fungsionalitasnya.

4. Berdasarkan hasil pengujian integrasi dengan presentase sebesar 71% terhadap antar *microservice* yang terintegrasi satu sama lain.

Adapun saran yang diberikan untuk pengembangan selanjutnya dari impementasi arsitektur *microservice* agar selalu memenuhi kebutuhan pengguna:

1. Ada baiknya pada pengembangan sistem ini pada tahap selanjutnya dilakukan pengujian dengan menggunakan metode *white box testing* karena memungkinkan kedepannya pengembangannya yang akan dilakukan akan lebih mendalam.

2. Untuk mendapatkan skalabilitas dengan integrasi yang lebih tinggi perlu diterapkan *command query separation* dan *multiple server* pada penelitian selanjutnya.

DAFTAR PUSTAKA

[1] L. Z. Apriliana, “Efektivitas Penggunaan E–Tilang Terhadap Pelanggaran Lalu Lintas Di Polres Magelang,” *J. Komun. Huk.*, vol. 5, no. 2, p. 1, 2019, doi: 10.23887/jkh.v5i2.17595.

[2] A. L. Yudanto, H. Tolle, and A. H. Brata, “Rancang Bangun Aplikasi Sistem Informasi Manajemen Laboratorium Biomedik Fakultas Kedokteran Universitas Brawijaya,” *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 1, no. 8, pp. 628–634, 2017.

[3] G. Arsya, P. Zaman, And P. N. Jakarta, “Perancangan Dan Implementasi Web Servicesebagai Media Pertukaran Data Pada Aplikasi Permainan,” *J. Inform.*, Vol. 11, No. 2, Pp. 22–30, 2017.

[4] I. A. Faruqi, S. F. S. Gumilang, and M. A. Hasibuan, “Perancangan Back-End Aplikasi Rumantara Dengan Gaya Arsitektur Rest Menggunakan Metode Iterative Incremental,” *eProceedings Eng.*, vol. 5, no. 1, pp. 1411–1417, 2018.

[5] R. A. Putra, “Analisa Implementasi Arsitektur Microservices Berbasis Kontainer Pada Komunitas Pengembang Perangkat Lunak Sumber Terbuka (Opendaylight Devops Community),” vol. 9, pp. 150–162, 2019.

[6] M. Sendiang, S. Kasenda, and J. Purnama, “Implementasi Teknologi Mikroservice pada Pengembangan Mobile Learning,” *J. Appl. Informatics Comput.*, vol. 2, no. 2, pp. 63–66, 2018, doi: 10.30871/jaic.v2i2.1046.

[7] M. Prabowo and A. Suprapto, “Implementasi Metode Profile Matching Dalam Aplikasi Penerimaan Siswa Baru pada SMK Ma’arif NU 2 Boyolali,” *Jusifo (Jurnal Sist. Informasi)*, vol. 5, no. 2, pp. 71–80, 2019.

[8] M. P. Putri and H. Effendi, “Implementasi Metode Rapid Application Development Pada Website Service Guide ‘ Waterfall Tour South Sumatera ,’” vol. 07, no. September, pp. 130–136, 2018.

[9] F. Luthfi, “Penggunaan Framework Laravel Dalam Rancang Bangun Modul Back-End Artikel Website Bisnisbisnis.ID,” *JISKA (Jurnal Inform. Sunan Kalijaga)*, vol. 2, no. 1, p. 34, 2017, doi: 10.14421/jiska.2017.21-05.

[10] Y. Heriyanto, “Perancangan Sistem Informasi

Rental Mobil Berbasis Web Pada PT.APM Rent Car," Vol. 2, No. 2, Pp. 64–77, 2018.

[11] D. P. Jayanto, "Informasi Web Application Backend Development Of ' Siap ': ' Sistem Informasi Aspirasi Dan Pengaduan MasyarakatApplication Based On Web Using Microservice Springboot," P. 156, 2017