E-ISSN:2540-9719
ISSN:2302-8149

IMPLEMENTASI API *MASTER STORE* MENGGUNAKAN FLASK, REST DAN ORM DI PT XYZ

Brian Pratama Putra, Yerymia Alfa Susetyo
Program Studi Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Satya Wacana,
Jl. Dr. O. Notohamidjojo, Salatiga 50714, Indonesia
Email: 672016228@student.uksw.edu, yeremia.alfa@uksw.edu

(Diterima: 15 Juni 2020 ,direvisi: 7 Juli 2020, disetujui: 25 Juli 2020)

ABSTRACT

Master Store is one of the main systems that governs store data contained in PT XYZ, the master store system is still running and works with Monolithic architecture which requires that every new and old application PT XYZ must duplicate the master store database, this can cause weakening and database performance decreases over time. Making Master Store Application Programming Interface aims to change the architecture currently used into Microservices where the Master Store system can then be used continuously by other applications and make it easier for developers to build new applications using the master store database without fear of a decline in the master store database. . The design of this API uses the Python programming language because it is a safe and simple programming language, Flask framework, REST communication standards, and ORM methods. ORM method was chosen because the security system is better compared to native queries in general, especially because of the process and development of ORM methods based on previous research. The results this study are the dynamic Master Store API that provides store data response, based on what PT XYZ developers need for applications and programs that are built.

Keywords: API, flask, *microservices*, *monolithic*, ORM, REST

ABSTRAK

Master Store merupakan salah satu sistem utama yang mengatur tentang data toko yang terdapat pada PT XYZ, sistem *master store* ini masih berjalan dan bekerja dengan arsitektur *Monolithic* yang mengharuskan setiap aplikasi baru maupun lama PT XYZ harus melakukan duplikasi *database master store* hal ini dapat menyebabkan melemah dan menurunnya performa *database* seiring berjalannya waktu. Pembuatan *Application Programming Interface Master Store* bertujuan untuk mengubah arsitektur yang saat ini digunakan menjadi *Microservices* dimana sistem *Master Store* ini kemudian dapat digunakan secara terus menerus oleh aplikasi lain dan mempermudah para *developer* untuk membuat aplikasi baru menggunakan *database master store* tanpa takut terjadi penurunan terhadap *database master store*. Perancangan API ini menggunakan bahasa pemrograman Python karena merupakan bahasa pemrograman yang aman dan sederhana, *framework* Flask, standar komunikasi REST, serta metode ORM. Metode ORM dipilih karena sistem keamanan yang lebih baik dibandingkan dengan *query native* pada umumnya, terlebih karena proses dan perkembangan metode ORM berdasarkan penelitian terdahulu. Hasil dari penelitian ini adalah *API Master Store* dinamis yang memberikan *response* data toko, berdasarkan apa yang *developer* PT XYZ butuhkan untuk aplikasi dan program yang dibangun.

Kata Kunci: API , flask, *microservices*, *monolithic*, ORM, REST

1 PENDAHULUAN

PT XYZ termasuk dalam salah satu perusahaan terbesar di Indonesia yang bergerak dalam bidang retail, juga menggunakan sistem *Application Programming Interface* pada beberapa aplikasi yang dimilikinya. Sebagai perusahaan retail yang memiliki beribu-ribu transaksi perhari, PT XYZ khususnya PT XYZ harus memiliki berbagai sistem untuk menunjang transaksi yang dilakukan serta

berbagai aplikasi untuk memenuhi dan mempermudah kebutuhan yang diperlukan, baik untuk konsumen, mitra kerja maupun PT XYZ sendiri.

Salah satu sistem yang memiliki peran penting dalam setiap transaksi dan proses bisnis yang terjadi di PT XYZ adalah *Master Store*. *Master Store* adalah sebuah sistem atau aplikasi yang menyimpan berbagai data serta informasi mengenai semua toko yang dimiliki oleh PT XYZ. *Master Store* memiliki *database master* atau utama yang digunakan aplikasi *Master Store* maupun aplikasi lain yang membutuhkan data mengenai toko-toko PT XYZ, namun seiring berjalanannya waktu dan berkembangnya PT XYZ, aplikasi-aplikasi yang membutuhkan data mengenai toko pun semakin bertambah, disisi lain hingga penelitian ini dilakukan, PT XYZ masih menggunakan arsitektur *monolithic* pada sistem *Master Store* dimana aplikasi - aplikasi yang membutuhkan data toko PT XYZ harus membuat atau menduplikasi *database master store* dan melakukan sinkronisasi dengan *database Master Store*, karena arsitektur *monolithic* dibangun sebagai sistem tunggal dan mandiri, meskipun arsitektur *monolithic* dibagi menjadi modul dan layanan yang terpisah tetapi semuanya berjalan dalam proses yang sama, oleh karena itu arsitektur *monolithic* memiliki kecepatan yang lebih unggul jika dibandingkan arsitektur yang lain tetapi memiliki kelemahan yaitu bahwa layanan atau sistem yang dibangun akan saling tergantung[1].

Karena masalah tersebut PT XYZ melakukan perubahan pada arsitektur teknologi yang telah ada yaitu dari arsitektur *monolithic* menjadi arsitektur *microservices*. *microservices* adalah arsitektur teknologi yang membagi layanan atau modul menjadi bagian lebih kecil dan terpisah sehingga modul atau layanan yang terbentuk dapat dijalankan dan digunakan oleh aplikasi atau program lain tanpa perlu membuat ulang atau menduplikasinya, yang terpenting adalah layanan atau modul yang berjalan menggunakan mekanisme komunikasi yang ringan[2], sebagai contoh memisahkan antara *back-end* dan *front-end*, *back-end* yang telah dibangun dapat digunakan oleh aplikasi lain yang membutuhkan data dan fungsi yang telah dibuat tersebut, sehingga hal ini dapat mengurangi waktu pembuatan dan penurunan performa *server* yang saat ini masih sering terjadi.

API atau *Application Programming Interface* adalah suatu pengolahan maupun kumpulan perintah dan data yang terdiri dari beberapa fungsi, kelas, antar muka (*interface*), struktur serta berbagai protokol untuk merancang, membangun, serta implementasi perangkat lunak. API bisa dikatakan sebagai kode program atau sistem yang berfungsi untuk menghubungkan antar *web* atau aplikasi (*front-end*) dan berbagai fungsi yang ada (*back-end*)[3]. Karena itu API *Master Store* dirancang untuk mengubah arsitektur teknologi *monolithic* menjadi microservices dengan cara menjadi *back-end* berupa API untuk aplikasi - aplikasi yang membutuhkan akses atau data toko sehingga tidak perlu membuat *database* sendiri atau menduplikasi *database Master Store* sehingga API yang dibuat ini akan meminimalkan akses *database Master Store* oleh banyak aplikasi menjadi berkurang, dan secara otomatis performa *database server* maupun aplikasi yang saling terhubung tidak akan menurun.

2 TINJAUAN PUSTAKA

Dalam penelitian terdahulu yang berjudul "Perancangan *Application Programming Interface* (API) Berbasis Web Menggunakan Gaya Arsitektur *Representational State Transfer* (REST) Untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit", dijelaskan cara perancangan API untuk sebuah sistem informasi dalam klinik, yang berfokus pada administrasi pasian klinik perawatan kulit dalam penelitian ini sangat berkaitan dengan pembuatan API Master Store, tujuan dari penelitian tersebut adalah untuk kegiatan administrasi pasien yang akan berobat atau berkonsultasi pada klik tersebut. API yang dirancang menggunakan arsitektur REST, namun pada penelitian ini masih menggunakan *query native* sehingga *output* API yang terbentuk masih statis, yang dimaksud adalah data *response* yang tercipta adalah tetap dan tidak bisa diubah oleh pengguna atau programmer yang ingin menggunakan API ini[4].

Pada penelitian yang berjudul "*Application Programming Interfaces* pada Aplikasi *Geo Social Commerce*", dijabarkan tentang perancangan API menggunakan *Geographic Information System* (GIS) yaitu sebuah sistem informasi khusus yang biasanya digunakan untuk mengelola informasi spasial (bentuk ruang) biasanya digunakan dalam pembuatan peta, tujuan penelitian ini adalah untuk

membangun sebuah website Marketspot atau biasa disebut *Geo Social Commerce* yang berfungsi untuk membantu penjual barang atau jasa bertemu dengan *customer* di sekelilingnya dengan menampilkan *customer* di sebuah peta yang dibuat menggunakan GIS, dan *customer* dapat langsung berinteraksi begitu pula penjual maupun penyedia jasa[5].

Penelitian yang berjudul "Perancangan dan Pembuatan *Application Programming Interface Server* untuk *Arduino*", membahas mengenai perancangan serta pembangunan *Application Programming Interface* untuk menghubungkan atau menjadi jembatan antara aplikasi dan *arduino* pada jaringan *computer* atau *internet* sehingga dapat mempermudah *programmer* atau *developer* dalam membuat aplikasi dan meringankan kerja *arduino*[6].

Pada penelitian yang berjudul "Penerapan *Model View Controller* dan *Object Relational Mapping* Pada Pengembangan Sistem Informasi Keanekaragaman Hayati di Taman Nasional Kutai Bontang". Dijabarkan tentang penggunaan dan penerapan MVC serta ORM untuk mengembangkan sistem informasi pariwisata di Taman Nasional Kutai Bontang, terlihat dan bisa disimpulkan pada penelitian tersebut ORM menjadi fitur keamanan tambahan karena data-data yang ada diubah menjadi *object*. Data yang digunakan pada penelitian tersbut adalah data test karyawan pada Hotel Grand Victoria Samarinda[7].

Pada penelitian yang berjudul "Implementasi Konsep *Object Relational Mapping* dan *Model View Controller* pada Manajemen Pembelian, Penjualan dan Inventory", membahas mengenai penerapan serta implementasi konsep model *Object Relational Mapping* serta *Model View Controller* membantu dalam mengelola data pada sistem pembelian dan penjualan *Inventory* TOP Distribusi Salatiga, sedikit berbeda dengan penelitian yang sebelumnya dibahas pada penelitian ini implementasi ORM lebih ditonjolkan sehingga terlihat pembelian dan penjualan yang terjadi pada TOP Distribusi Salatiga berjalan dan aman[8].

Penelitian yang berjudul "Implementasi *Application Programming Interface* (API) *Google Calendar* Sebagai *Reminder* Informasi Kegiatan Pondok Pesantren", membahas mengenai pemanfaat dari API yang disediakan oleh Google yaitu API *Google Calendar* yang dikombinasikan dengan sebuah metodologi pengembangan sistem yaitu *Personal Extreme Programming* (PXP), hasil dari penelitian ini adalah membuat sebuah aplikasi pesantren *reminder* yang berfungsi untuk pengaturan jadwal kegiatan dan menyampaikan informasi kegiatan kepada seluruh anggota pondok pesantren secara manual menggunakan API *Google Calendar*[9].

Berdasarkan beberapa penelitian yang telah dilakukan sebelumnya, maka akan dilakukan penelitian terkait pembuatan *Application Programming Interface* yang menggunakan *framework* Flask, metode *Object Relational Mapping*, dan arsitektur REST dengan judul penelitian "Implementasi API Master Store Menggunakan Flask, REST dan ORM DI PT XYZ". Perbedaan yang paling mendasar antara penelitian terdahulu dengan penelitian sekarang adalah kasus yang diambil, kasus yang diambil pada penelitian ini adalah kasus nyata yang dialami oleh PT XYZ, selain perbedaan mendasar dari kasus yang diambil, perbedaan lain dari penelitian sekarang adalah penggunaan *framework* Flask, standar arsitektur komunikasi REST dan metode ORM yang dikombinasikan sehingga dapat menghasilkan API yang dinamis dan dapat digunakan untuk semua kebutuhan *developer*.

Penggunaan *framework* Flask dipilih karena Flask merupakan *microframework* dan berbeda dengan kebanyakan *framework* karena Flask jauh lebih ringan dan cepat serta mudah diimplementasikan, hal ini karena *framework* Flask dibuat dengan tujuan untuk menyerdehanakan inti *framework* seminimal mungkin[10]. Flask digunakan sebagai kerangka kerja suatu aplikasi dan tampilan dari *website* yang dibangun. Dengan menggunakan Flask dan bahasa program Python, pengembang atau *programmer* dapat membuat sebuah *website* ringan yang pastinya tetap terstruktur dan dapat mengatur perubahan suatu *website* serta *maintenance* dengan mudah, karena sebagian besar fungsi dan komponen umum seperti *database*, validasi *form* dan sebagainya tidak terpasang secara *default* di Flask[11].

Pemilihan REST untuk perancangan *API Master Store* karena REST ialah salah satu teknologi *web service* yang cukup populer di masa sekarang ini terlebih dalam industri teknologi 4.0. Teknologi ini bekerja berdasarkan sumber daya data untuk membuat sistem menjadi terdistribusi. REST (yang sering juga disebut sebagai *RESTful services*) adalah *software* atau perangkat lunak yang didesain dengan sederhana pada penekanan skalabilitas dan kegunaan[12]. REST sering dijalankan melalui

HTTP (*Hypertext Transfer Protocol*) oleh karena itu REST dapat dimanfaatkan untuk membaca halaman *website* tertentu yang memuat sebuah atau beberapa file data yang berbentuk XML ataupun JSON[13].

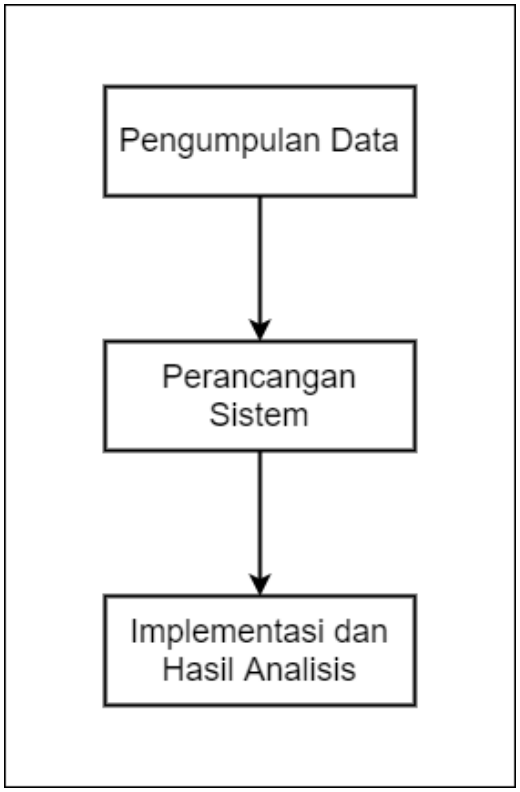
Penggunaan ORM dipilih karena kode database dipetakan menjadi *object* sehingga keamanan, pemeliharaan dan portabilitas akan lebih baik, pemetaan yang dilakukan juga mengurangi terjadinya jumlah masalah yang dihadapi *developer* maupun *user*[14]. ORM atau *Object Relational Mapping*, adalah suatu teknik yang menggambarkan *persistent object* pada suatu perangkat lunak atau sistem menjadi suatu *table* pada *database* serta memungkinkan untuk melakukan pengelolaan tipe sistem yang berbeda pada basis data relasional dengan bahasa pemrograman berorientasi objek, atau dengan kata lain ORM dapat juga disebut sebuah teknologi pemrograman yang menghubungkan SQL dengan konsep pemrograman berorientasi objek. ORM mampu menjadi jembatan untuk menghubungkan perbedaan tipe data pada konsep pemrograman yang berorientasi objek dengan konsep pemrograman *Relational Database Management System* (RDBMS). Dengan mengimplementasikan ORM pada sistem yang dibangun, *developer* dapat lebih berpikir secara objek dibanding dengan berpikir terhadap kolom dan tabel, yang menjadi ciri dari relasional model. ORM dibangun untuk menjaga kemurnian pola pikir *developer* atas pemrograman berorientasi objek[15].

Hasil akhir dari penelitian ini adalah API *Master Store* yang diharapkan dapat membantu para *developer* dan *programmer* dalam membuat ataupun melakukan migrasi aplikasi – aplikasi yang ada di PT XYZ dengan mudah tanpa harus melakukan duplikasi *database master store* serta tidak memberikan efek negatif pada *server database master store*, seperti menurunnya performa *database* yang sering terjadi saat transaksi data yang dilakukan dalam skala besar. API Master Store yang dibuat akan diuji dengan dua tahapan yaitu pengujian token dengan JWT (JSON Web Token) pada *back-end server*,dan pengujian performa API menggunakan metode *Equivalence Partitioning*.

3 METODE PENELITIAN

Dalam pembuatan API *Master Store* dilakukan beberapa tahapan diantaranya identifikasi masalah, pengumpulan data, perancangan sistem, implementasi dan hasil analisis, penulisan laporan penelitian, seperti yang terlihat pada Gambar 1.

Berdasarkan Gambar 1 dapat dijelaskan bahwa proses tiap tahapan penelitian yang dilakukan adalah Tahapan pertama adalah (1) dilakukan pengumpulan data serta analisis kebutuhan dalam menyelesaikan permasalahan. Pengumpulan dan analisis data dilakukan ketika bertemu dengan *programmer* yang bersangkutan untuk mendapatkan informasi agar dapat menemukan solusi yang tepat dan terbaik. Contoh dari informasi yang didapatkan saat bertemu dengan *programmer* adalah ketika suatu aplikasi A dan B membutuhkan data yang berbeda dari *database master store,* dengan demikian *programmer* akan melakukan duplikasi lebih dari satu hit ke *database* hanya untuk 2 aplikasi. Kemudian, berdasarkan informasi yang telah di dapatkan, pada tahap (2) perancangan sistem dibuatlah sebuah rancangan sistem *Application Programming Interface Master Store* meliputi, pembuatan UML *(Unified Modeling Language)* berupa *use case diagram, class diagram, activity diagram, sequence diagram*. Berikutnya, setelah rancangan sistem berupa diagram telah selesai dilakukan tahap (3), implementasi dan hasil analisis, pada tahapan ini dilakukan pembuatan sistem API yang telah dirancang menggunakan bahasa pemrograman Python dengan *framework* Flask, arsitektur REST serta metode ORM. Setelah implementasi sistem selesai.



Gambar 1. Metode Penelitian

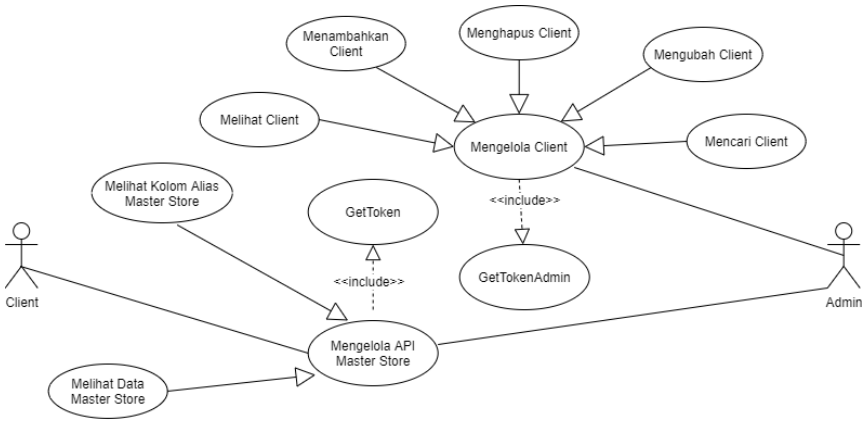
4 HASIL DAN PEMBAHASAN

Pada bab hasil dan pembahasan dibagi menjadi 3 sub bab, dimulai dari perancangan sistem yang menjelaskan tentang diagram UML serta atribut-atribut yang dimiliki dalam API *Master Store*, kemudian implementasi, pembuatan *code* program menggunakan bahasa pemrograman python, berdasarkan rancangan yang telah dibuat, setelah pembuatan code program selesai, akan dilakukan pengujian API yang telah dibuat menggunakan JWT dan *Blackbox*.

4.1 Perancangan Sistem

Pada perancangan sistem telah dibuat diagram UML yang terdiri dari :

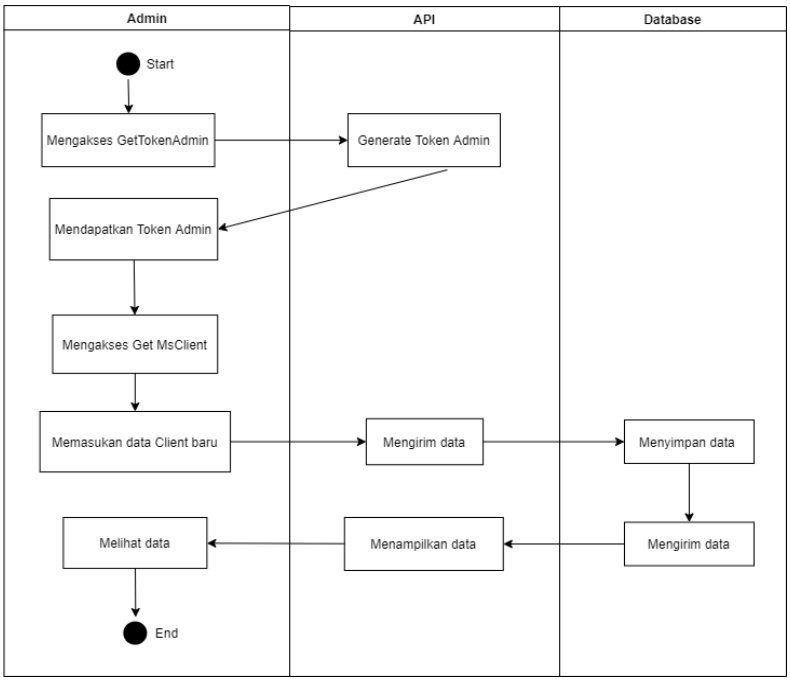
1) Use Case Diagram



Gambar 2. *Use Case API Master Store*

Gambar 2 adalah *use case diagram* API *Master Store* yang dibuat memiliki dua aktor utama yaitu *client* dan *admin*. *Client* berperan sebagai *user* yang ingin mengakses API *Master Store* untuk kebutuhan aplikasi atau laporannya. Admin merupakan super *client* atau dapat dikatakan *client* yang memiliki hak lebih tinggi karena selain dapat mengakses API *Master Store*, admin juga yang mengatur pembuatan, pengubahan dan penghapusan *client* lain yang ingin mengakses API *Master Store*.

2) Activity Diagram



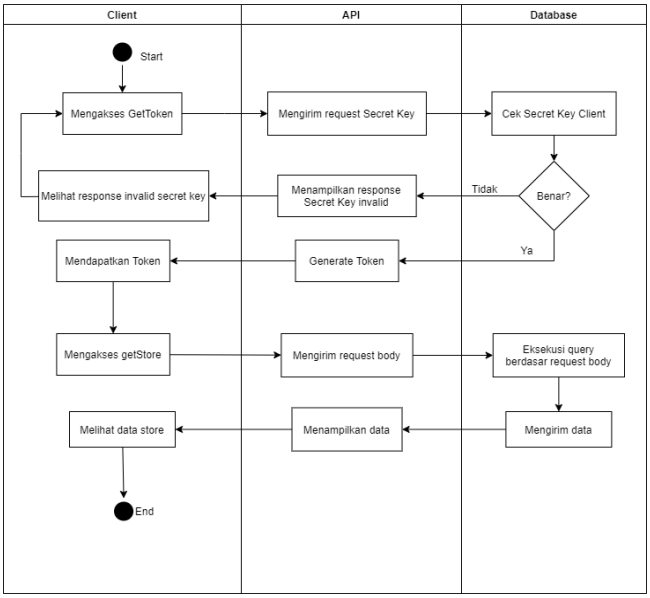
Gambar 3. ***Activity Diagram Admin Insert Client - Api Master Store***

Pada Gambar 3 dijelaskan mengenai perancangan *activity diagram insert client* menunjukkan diagram alir suatu aktivitas. *Activity diagram* yang terjadi adalah admin mengakses API untuk mendapatkan token admin, hal ini dilakukan untuk memvalidasi apakah yang akan melakukan penambahan *client* adalah benar *admin*, setelah mendapatkan *token*, *admin* akan mengakses *function MsClient* dan memasukan *client* baru, API akan mengirimkan *request insert* kemudian *database* akan menyimpan data *client* baru, setelah berhasil, akan muncul data *client* yang telah ditambahkan.

Activity Diagram pada Gambar 4 menjelaskan bagaimana proses aktivitas *client* untuk dapat mengakses data toko pada API *Master Store*, langkah pertama yang dilakukan oleh *client* adalah memastikan apakah *client* tersebut sudah memiliki hak akses ke API *Master Store*, dengan cara mengakses *token* dengan *secret key* yang dia miliki melalui *getToken*, jika *secret key* yang dimiliki benar maka API akan megenerate token untuk mengakses data toko, jika *secret key* yang dimilikinya salah, maka akan muncul response "*Invalid Secret Key*". Setelah *client* mendapatkan *token*, *client* dapat mengakses data toko yang ada melalui *getStore* menggunakan *request body* data sesuai dengan kebutuhan *client* tersebut, API akan melakukan pengecekan terhadap *request body*, setelah membaca *request body* maka akan dilakukan eksekusi perintah pada *database* kemudian API akan menampilkan data sesuai dengan permintaan *client* pada *request* yang dikirim.

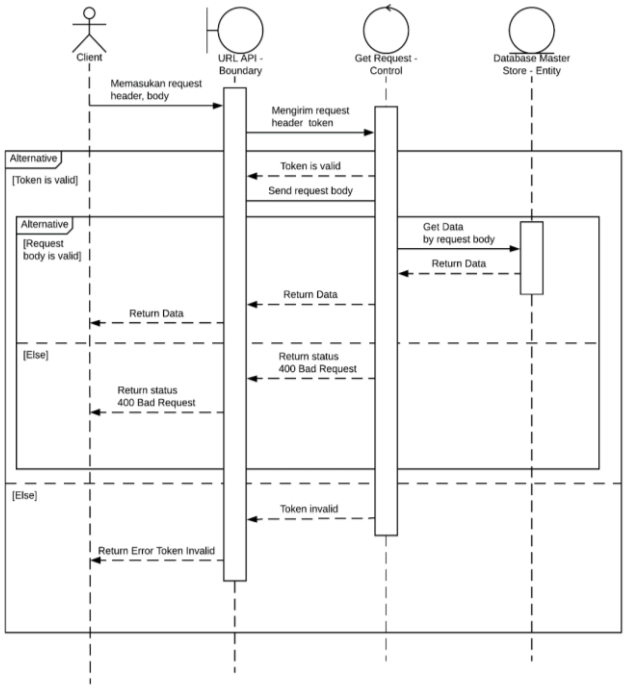
Gambar 5 menunjukan alur yang terjadi saat *client* melakukan *getStore (Get Data Store)* data master store, terlihat pada *sequence diagram* tersebut saat *client* akan melakukan akses pada *master store*, *client* akan memberi *request input* pada *header* dan *body*, *request header* berisi *token* yang telah didapatkan setelah *client* ditambahkan oleh *admin*, jika token salah maka *controller api* akan mengirim *response token invalid*, jika *token* benar, maka *contoller api* akan melakukan cek *request*

body yang dikirim *client*, jika *request* pada *body* salah maka *api* akan memberi *response error* status 400 *Bad Request* jika *request body* yang dikirimkan telah benar, maka *controller api* akan mengirimkan request tersebut ke database master store dan *database* akan mengembalikan *response* status 200 dan data akan ditampilkan.



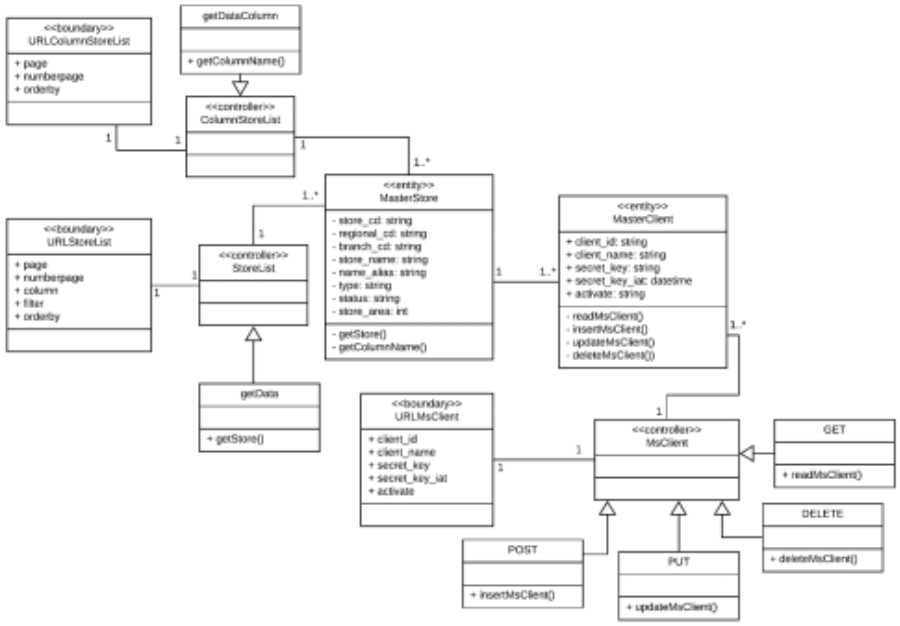
Gambar 4. ***Activity Diagram Get Data Store - Api Master Store***

3) Sequence Diagram



Gambar 5. ***Sequence Diagramet Data Store - Api Master Store***

E-ISSN:2540-9719
ISSN:2302-8149

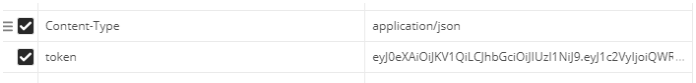


Gambar 6. *Class Diagram* Api *Master Store*

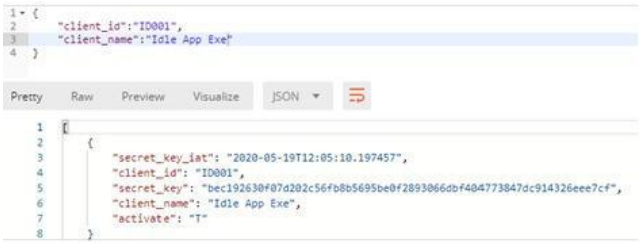
Gambar 6 merupakan *Class Diagram* dalam API *Master Store*. API tersebut terdiri dari 2 kelas yang saling berelasi satu dengan yang lainnya. *Class diagram* juga memiliki *attribute* serta *behaviour*-nya masing-masing. *Kardinalitas* di dalam API *Master Store* adalah penentuan berapa banyak hubungan suatu *class* antara *class* satu dengan yang lainnya. *Kardinalitas* dalam Gambar 7 yaitu setiap *client* dapat melakukan lebih dari satu *request* ke *Master Store*.

4.2 Implementasi

Pada penjelasan perancangan sistem API *Master Store* yang dibuat. *Admin* dapat melakukan pengelolaan terhadap *client* yang ingin mengakses API *Master Store*, namun disisi lain admin sendiri juga dapat mengakses data API *Master Store*, pada Gambar 7 terlihat *request header* untuk melakukan *insert client* dan pada Gambar 9 terlihat *request body* dan contoh data *client* baru yang telah berhasil tersimpan.



Gambar 7. *Header Request Insert Client*

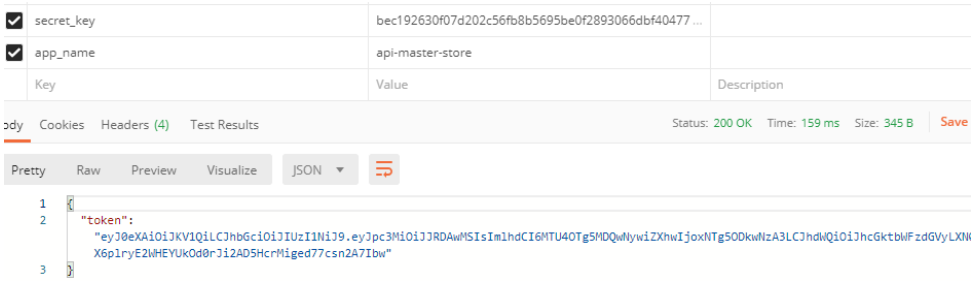


Gambar 8. *Body Request Insert Client*

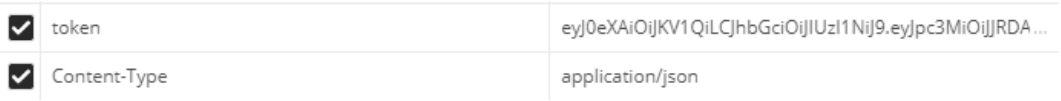
Setelah *client* sudah terdaftar pada API *Master Store* dapat dilihat bahwa *client* memiliki *secret key*, *secret key* tersebut yang akan digunakan untuk *generate token* yang akan digunakan untuk mengakses data toko pada API *Master Store*. Dapat dilihat pada Gambar 9, *client Idle App Exe* melakukan *request generate* token. Pada fungsi *getToken* hanya dibutuhkan *headers parameter* yang

E-ISSN:2540-9719
ISSN:2302-8149

berupa *secret key* yaitu *secret key* masing-masing *client* dan *app name* yaitu API yang dituju yaitu api-*master-store*.

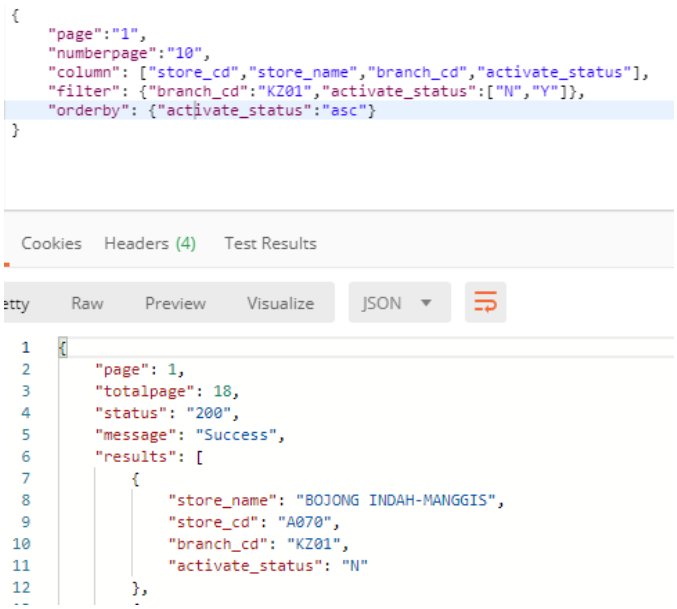


Gambar 9. *Request Token*



Gambar 10. *Request Headers Getstor*

Setelah mendapatkan *token*, *client* akan mengakses *getStore* untuk mendapatkan data toko yang diperlukan, *token* yang terbentuk memiliki masa aktif dengan begitu *client* harus melakukan *generate token* setiap beberapa menit, hal ini dilakukan untuk meningkatkan keamanan pada API *Master Store*, sehingga token yang terbentuk tidak akan disalah gunakan oleh orang yang tidak bertanggung jawab, dapat dilihat pada Gambar 10. *request header* yang diperlukan *client* untuk dapat mengakses data toko API *Master Store*, yaitu *token* yang didapatkan saat *generate token* dengan *secret key* masing-masing *client* dan *Content-Type* yang berisikan *application/json* untuk memastikan data yang dihasilkan berbentuk json, dan pada Gambar 11 terlihat *request body* yang dibutuhkan untuk mendapatkan data sesuai kebutuhan setiap *client*, *request page* berfungsi untuk menentukan *pagination* mengatur data pada halaman berapa yang akan diambil. *numberpage* adalah total data yang akan ditampilkan dalam 1 halaman, *column* adalah *request* yang berfungsi untuk menentukan data kolom apa saja yang akan diambil, *filter* adalah *request parameter* kondisi tertentu yang diperlukan sebagai contoh adalah dimana *branch_cd* bernilai KZ01, jika terdapat kondisi *or* dapat dilakukan dengan membuat *list* atau *array*, seperti yang terlihat pada Gambar 11, *orderby* adalah *request parameter* yang berfungsi untuk mengurutkan data yang ditampilkan.



Gambar 11. *Request Body* Dan *Response Getstore*

API *Master Store* dirancang menggunakan bahasa pemrograman Python, *framework* Flask serta metode ORM yang mengubah semua perintah SQL atau query menjadi *object*, sehingga metode ORM ini dapat mengatasi masalah yang sering ditemukan pada API, seperti SQL *Injection* atau perubahan perintah SQL melalui *request header* dan *body*, berikut adalah contoh kode program dari API *Master Store.*

```
class getToken(Resource):
        def post(self):
        app_name = request.headers.get('app_name')
        secret_key = request.headers.get('secret_key')
        strfilter={}
        aktif = 'T'
        cek_secret_key = msClient(strfilter,aktif)
        for cek in cek_secret_key:
   if secret_key == cek_secret_key[cek_secret_key.index(cek)]['secret_key']:
       strfilter = {"secret_key":secret_key}
       hasil = msClient(strfilter,aktif)
       for hsl in hasil:
         token = jwt.encode({'iss':hasil[hasil.index(hsl)]['client_id'],
         'iat':datetime.utcnow(),
         'exp' : datetime.utcnow() + timedelta(minutes=5),
         'aud':apl_name}, secret_key)
       return jsonify({'token' : token.decode('UTF-8') })
     else:
       token = 'None'
       if token == 'None':
       return jsonify({'description':'Invalid Key','status_code':'401'})
```

Kode Program 1 *Function Encode Token*

Kode program 1 pada baris 3 dan 4 merupakan proses pembacaan *app_name* dan *secret_key* dari *request headers*, setelah itu akan dilakukan pengecekan *secret_key* apakah dimiliki oleh *client* yang telah terdaftar pada *Master Client*, jika *secret key* benar dimiliki oleh *client* maka akan dilakukan *encode token* menggunakan jwt proses ini ditunjukan pada baris ke 13, hasil dari proses *encode secret key* menjadi *token* dapat dilihat pada Gambar 10, jika *secret key* pada *request headers* salah maka API akan menampilkan *response error status code* 401 dengan *description Invalid Key*.

```
def tkn_required(f):
  def decodeted(*args, **kwargs):
    tkn = request.headers.get('tkn')
    if not tkn:
      return jsonify({'message' : 'Token Hilang, Generate Token Ulang!!'})
    try:
      token_jwt = jwt.decode(token,verify=False)
      strfilter = {"client_id":token_jwt['iss']}
      aktif = 'T'
      cek_secret_key = msClient(strfilter,aktif)
      for cek in cek_secret_key:
                          data=jwt.decode(token,
                          cek_secret_key[cek_secret_key.index(cek)]['secret_key'],
                          audience=token_jwt['aud'])
    except Exception as e:
      print(str(e))
      return jsonify({"page": "None",
                              "totalpage": "None",
                    "status": "401",
                    "message": "Token Invalid",
                    "results": "None"
                   })
    return f(*args, **kwargs)
  return decodeted
```

Kode Program 2 *Function Decode Token*

Kode program 2 pada baris ke 7 merupakan kode proses pembacaan *token* yang telah dibuat, pada fungsi baris ke 11 isi dalam *token* akan dicek sehingga dapat diketahui siapa yang membuat dan

E-ISSN:2540-9719
ISSN:2302-8149

akan menggunakan *token* tersebut untuk mengakses data toko API *Master Store*, jika pembuat dan pengguna *token* adalah *client* yang berbeda maka akan terdapat *response Token Invalid* dan tidak dapat digunakan.

```
result = AmuStoresTab.query.limit(page_size).offset(page*page_size)
storeSchemaMult=StoreSchema(many=True,only=column)
return storeSchemaMult.dump(result)
```

Kode Program 3 Kode ORM *Getstore*

Kode program 3 pada baris ke 1 merupakan *query* kode untuk mendapatkan data dari *database* menggunakan metode ORM, terlihat perbedaan yang cukup jelas saat menggunakan kode program ORM, *query* yang digunakan berbentuk *object* sehingga keamanan saat melakukan eksekusi *query* menjadi lebih aman dan fleksibel.

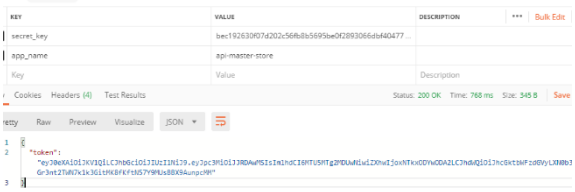
4.3 Pengujian

Pengujian API dilakukan dengan 2 metode yaitu menguji token yang dibuat API dengan menggunakan jwt untuk melakukan pengujian terhadap token yang terbentuk dan pengujian menggunakan metode *Equivalence Partitioning* dengan *Black Box testing*:

1) JWT *(JSON Web Tokens)*

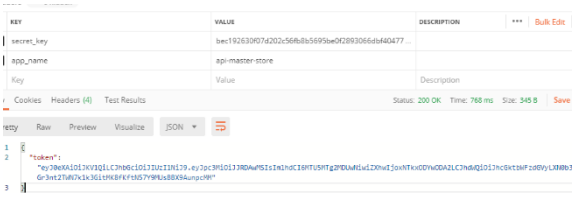
Tabel 1. Pengujian Menggunakan JWT

No	Pengujian	Hasil
1	Pengujian pertama dilakukan pengecekan apakah app_name tujuan telah benar yaitu api-master-store dan siapa pembuatnya.	Hasil dari pengujian pertama adalah keakuratan token yang dibuat dan sesuai dengan yang diinginkan.
2	Pengujian kedua dilakukan pengecekan masa aktif token, yang seharusnya sudah tidak bisa digunakan setelah 5 menit dihitung saat token digenerate.	Berdasarkan yang terlihat, hasil yang keluar pada pengujian kedua menyatakan sistem *expired* telah berhasil, token tidak bisa digunakan setelah 5 menit.

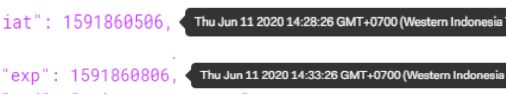


Gambar 12. Pengujian 1

Gambar 13. Hasil Pengujian 1



Gambar 14. Pengujian 2



Gambar 15. Hasil Pengujian 2

Berdasarkan hasil uji JWT yang telah dilakukan, API Master Store sudah dikatakan aman karena setiap token yang dibuat sudah sesuai, token yang terbentuk memiliki jangka waktu sehingga jika token diakses saat jangka waktu habis token tersebut sudah tidak berlaku, token yang terbentuk juga memiliki informasi seperti *issuer* (pembuat token) dan iat (waktu pembuatan token), untuk melakukan pengujian lebih lanjut, akan dilakukan pengujian *blackbox*, berikut merupakan hasil dari pengujian yang telah dilakukan.

2) Black Box Testing

Tabel 2. Tabel *Black Box Testing* Api Getstore

No	Skenario Pengujian	Hasil yang diharapkan	Hasil Pengujian	Kesimpulan
1	Tidak memasukan token saat mengakses API getStore.	Sistem akan menolak menampilkan data toko dan akan memberi response status 401 dengan *message Token* Hilang, *Generate Token* ulang.	Sesuai yang diharapkan.	*Valid*
2	Memasukan token yang salah atau *expired*	Sistem akan menolak menampilkan data toko dan akan memberi response status 401 dengan *message Invalid Token*	Sesuai yang diharapkan.	*Valid*
3	Tidak memasukan Content-Type pada *request Header*.	Sistem tidak akan menampilkan data dengan bentuk JSON.	Data tetap didapatkan dengan bentuk JSON.	*Invalid*
4	Memasukan *request body* dengan tipe data yang salah.	Sistem akan menolak dan mengirimkan *response* 401 dengan *message request body* tidak sesuai.	Sesuai yang diharapkan.	*Valid*
5	Melakukan *sql injection* pada *request body*.	Sistem akan membaca karakter ilegal kemudian menghapus karakter ilegal tersebut, mengirimkan *response* data jika *request body* sesuai, jika tidak akan mengimkan *response* 401.	Sesuai yang diharapkan.	*Valid*
6	Tidak mengisi *request body* atau mengosongi parameter data.	Sistem akan tetap menampilkan semua data karena tidak mengimkan kondisi parameter data.	Sesuai yang diharapkan.	*Valid*
7	Menghapus semua parameter atau *request body*.	Sistem akan menolak dan mengirimkan *error* yang menyatakan bahwa parameter tetap harus ada meskipun datanya kosong.	Sesuai yang diharapkan.	*Valid*
8	Memasukan *request body* pada parameter *column* dengan data salah atau tidak sesuai dengan alias *column* yang dibuat.	Sistem akan menolak menampilkan data toko dan akan memberi response status 401 dengan *message Failed – Column does not exist.*	Sesuai yang diharapkan.	*Valid*
9	Memasukan *request body* pada parameter *filter* dengan data salah atau asal.	Sistem tetap akan mengirimkan *response* dengan status 200 dengan *message Success* namun data yang ditampilkan tidak ada.	Sesuai yang diharapkan.	*Valid*

5 KESIMPULAN

Dari penelitian dan pengujian API yang telah dilakukan bisa disimpulkan bahwa API *Master Store* telah dapat memenuhi kebutuhan perusahaan untuk data toko yang akan digunakan dalam pengembangan aplikasi lain yang ada dalam perusahaan, data yang dihasilkan dari *request* telah sesuai yang dibutuhkan *programmer*. Perpindahan arsitektur sistem dari *Monolithic* menjadi *Microservices*, sudah dapat dilakukan dengan API *Master Store* yang dibuat sehingga mempermudah para *developer* dan *programmer* saat membuat aplikasi baru maupun melakukan migrasi aplikasi, terlebih API yang dibuat adalah API dinamis yang dapat diatur data yang dibutuhkan untuk membangun aplikasi. Fitur pada alias kolom *database master store* juga dapat menjamin keamanan API *Master Store* karena *developer* tidak akan tahu nama kolom sebenarnya. Penggunaan metode ORM juga mendukung keamanan sistem API *Master Store* yang dibuat karena ORM merupakan suatu teknologi yang menggambarkan *persistent object* pada aplikasi maupun sistem menjadi tabel pada *database* sehingga memungkinkan *developer* untuk melakukan pengelolaan tipe sistem pada basis data relasional dengan bahasa pemrograman berorientasi objek yang berbeda, atau dengan kata lain ORM dapat disebut sebuah teknologi pemrograman yang menjadi jembatan untuk menghubungkan sistem *database* SQL dengan konsep pemrograman berorientasi objek, sehingga meningkatkan keamanan dalam peretasan yang mungkin akan dilakukan oleh orang yang tidak bertanggung jawab. *Framework* Flask juga menjadi keunggulan dari API *Master Store* ini karena flask menyediakan semua *libraries* yang dibutuhkan dan penggunaan flask lebih sederhana jika dibandingkan oleh *framework* lain, sehingga flask akan terasa lebih ringan.

Pengembangan atau penambahan fitur pada API *Master Store* kedepannya mungkin dapat ditambahkan sebuah *database* baru untuk mencatat *log* siapa saja yang mengakses API *Master Store* agar keamanan dan jejak *client* dapat dilacak dengan mudah, untuk saat ini *log* hanya bisa dilihat dari *token* yang mengakses API *Master Store*. Terlepas dari pembuatan sistem API *Master Store* ini, hal terpenting adalah dapat mengurangi resiko menurunnya performa *database master store* sehingga semua proses transaksi di PT XYZ tidak terganggu, terlebih lagi dapat mempermudah para *developer* untuk membuat sebuah aplikasi baru.

REFERENSI

[1] M. C. Architecture and D. Zöchbauer, "Monolithic Architecture Based on an Existing . NET Application," 2019.
[2] J. Hansen and G. Mike, "Monolith til Microservices," no. 201703726, pp. 1–40, 2019.
[3] M. S. Amri, "Membangun Sistem Navigasi Di Surabaya Menggunakan Google Maps Api," *Pens Its*, vol. 1, no. Proposal 2013, pp. 1–5, 2010.
[4] B. Adi Pranata, A. Hijriani, and A. Junaidi, "Perancangan Application Programming Interface (Api) Berbasis Web Menggunakan Gaya Arsitektur Representational State Transfer (Rest) Untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit," *J. Komputasi*, vol. 6, no. 1, pp. 33–42, 2018.
[5] A. F. Pambudy, S. Fajar, S. Gumilang, and M. A. Hasibuan, "Application programming interfaces," vol. 2, pp. 25–32, 2015.
[6] S. A. Sena, A. Muttaqin, and A. Setyawan, "Perancangan dan Pembuatan Application Interface Server untuk Arduino," *J. Tek. Elektro, Fak. Tek. Univ. Brawijaya*, vol. 1, no. 4, pp. 1–6, 2013.
[7] H. Muhammad Romadinu Al Mukabir, Edy Budiman, "Penerapan Model View Controller Dan Object Relational Mapping Pada Pengembangan Sistem Informasi Keanekaragaman Hayati Di Taman Nasional Kutai, Bontang," *Midwifery*, vol. 2, no. 2, pp. 227–249, 2018.
[8] K. D. Hartomo, R. Latuperissa, and R. W. Djunanto, "Implementasi Konsep Object Relational Mapping dan Model View Controller pada Manajemen Pembelian , Penjualan dan Inventory (Studi Kasus : TOP Distributor Salatiga)," pp. 133–149.
[9] A. Kurniawan, A. Rahmatulloh, and H. Sulastri, "Calendar Sebagai Reminder Informasi Kegiatan Pondok Pesantren," vol. 8, no. 1, 2019.
[10] M. G. L. Putra and M. I. A. Putera, "Analisis Perbandingan Metode Soap Dan Rest Yang Digunakan Pada Framework Flask Untuk," vol. XIV, pp. 1–7, 2019.

[11] R. Irsyad, "Penggunaan Python Web Framework Flask Untuk Pemula," *Lab. Telemat. Sekol. Tek. Elektro Inform.*, pp. 1–4, 2018.
[12] Y. Fauziah, "Aplikasi Iklan Baris Online menggunakan Arsitektur REST Web Service," *Telematika*, vol. 9, no. 2, 2014.
[13] D. S. Wiyono and A. Wijayanto, "Implementasi Rest Web Service Dengan Menggunakan Json Pada Aplikasi Mobile Enterprise Resource Planning," *PERFORMA Media Ilm. Tek. Ind.*, vol. 11, no. 2, pp. 143–152, 2012.
[14] T. H. Kim *et al.*, "Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7105 LNCS, no. December, 2011.
[15] U. D. Praditya, R. Saputra, and B. Noranita, "Implementasi Object Relational Mapping Pada Pengembangan E-Commerce Menggunakan Framework Yii," *J. Informatics Technol.*, vol. 2, no. 3, pp. 113–124, 2013.