InfoTekJar Jurnal Nasional Informatika dan Teknologi Jaringan
ISSN 2540-7600 (Online)
ISSN 2540-7597 (Print)

PENGEMBANGAN REST API LAYANAN PENYIMPANAN MENGGUNAKAN METODE *RAPID APPLICATION DEVELOPMENT* (STUDI KASUS: PT. XYZ)

Muhammad Angga Kawa Perdana
Program Studi Sistem Informasi, Jurusan Matematika dan Teknologi Informasi, Institut Teknologi Kalimantan
anggakawa@gmail.com

***Abstrak*—Dengan semakin berkembangnya teknologi informasi, saat ini banyak perusahaan yang menerapkan konsep arsitektur microservice pada aplikasi mereka. Arsitektur microservice adalah sebuah konsep yang menyusun aplikasi sebagai sekumpulan layanan yang berbeda dan memiliki keterikatan yang renggang. Makalah ini menjelaskan mengenai pengembangan REST API layanan penyimpanan pada PT. XYZ untuk mendukung sistem mereka. Metode yang digunakan adalah Rapid Application Development yang menjadi pedoman dalam pengembangan REST API. API yang telah terimplementasi telah memenuhi kebutuhan yang telah didapatkan.**

***Keywords*— REST API, *rapid application development*, *microservice*.**

Abstract - With the development of information technology, today many companies are applying the concept of microservice architecture in their applications. Microservice architecture is a concept that compiles applications as a set of different services and has a tenuous attachment. This paper describes the development of REST API storage service at PT. XYZ to support their system. The method used is Rapid Application Development which guides in REST API development. Implemented APIs have met the needs that have been obtained.

Keywords: REST API, rapid application development, microservice.

I. PENDAHULUAN

Dengan semakin berkembangnya teknologi informasi, saat ini banyak perusahaan yang menerapkan konsep arsitektur *microservice* pada aplikasi mereka. *microservice* adalah sebuah konsep arsitektur yang menguraikan aplikasi sebagai sekumpulan layanan yang berbeda dan memiliki keterikatan renggang. Dengan menguraikan sebuah aplikasi menjadi beberapa layanan yang berbeda, aplikasi dapat lebih mudah dipahami, dikembangkan dan diuji serta mampu meningkatkan modularitas dari aplikasi tersebut [1].

PT. XYZ merupakan sebuah perusahaan yang bergerak di bidang teknologi informasi dan saat ini sedang mengembangkan aplikasi untuk mendukung kebutuhan bisnis mereka. Arsitektur yang digunakan dalam pengembangan aplikasi ini adalah *microservice*, sehingga aplikasi terbagi menjadi beberapa layanan. Setiap layanan terbagi berdasarkan fungsionalitas yang dimiliki dan dapat berkomunikasi satu sama lain melalui protokol HTTP.

Salah satu layanan yang dibutuhkan adalah layanan penyimpanan, yang bertugas untuk menangani penyimpanan *file* pada aplikasi yang akan dibuat. Dalam penelitian ini dikembangkan REST API untuk mendukung layanan tersebut. REST API adalah implementasi dari API (*application programming interface*) yang metode komunikasinya menggunakan protokol HTTP. REST API digunakan agar memudahkan proses integrasi layanan yang ada.

Metode pengembangan perangkat lunak yang digunakan pada penelitian ini adalah RAD (*rapid application development*). RAD merupakan metode pengembangan software yang diciptakan untuk menekan waktu yang dibutuhkan untuk mendesain serta mengimplementasikan sistem informasi sehingga dihasilkan siklus pengembangan yang sangat pendek. Adapun tujuan dari penelitian ini adalah:

1. Merancang dan mengembangkan REST API layanan penyimpanan untuk PT. XYZ sehingga dapat diintegrasikan dengan layanan lainnya.
2. Menguji fungsionalitas API yang telah dikembangkan.

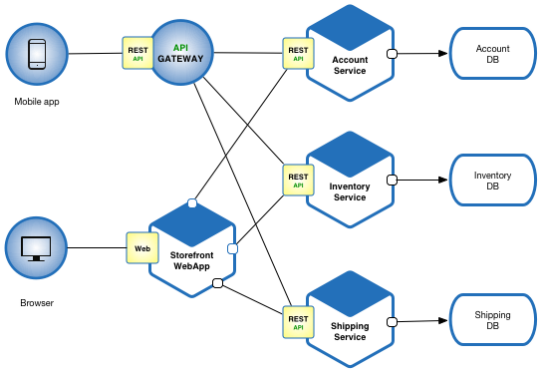
II. TINJAUAN PUSTAKA

A. Microservice

Microservices atau microservice architecture adalah sebuah varian dari service-oriented architecture (SOA) yang menyusun aplikasi sebagai sekumpulan layanan dengan keterikatan yang renggang satu sama lain. Dengan menguraikan sebuah aplikasi menjadi beberapa layanan yang berbeda, aplikasi dapat lebih mudah dipahami, dikembangkan dan diuji serta mampu meningkatkan modularitas dari aplikasi tersebut.

Konsep ini juga memparalelkan pengembangan aplikasi dengan memungkinkan setiap layanan dikembangkan, disebar, dan diukur skala-nya secara independen oleh tim kecil yang otonom. Sehingga setiap layanan tidak terlalu bergantung satu sama lain dan proses pengembangan aplikasi dapat

berlangsung lebih cepat [2]. Komunikasi antar *microservices* biasanya dilakukan melalui protokol HTTP dengan API ataupun *messaging*.



Gbr 1 Contoh arsitektur *microservice*

Gambar 1 memperlihatkan contoh dari arsitektur *microservice*, aplikasi terbagi menjadi beberapa layanan dengan tugas dan basis data masing-masing. Penguraian atau pembagian aplikasi menjadi beberapa layanan idealnya harus benar-benar terfokus pada satu fungsi ataupun sekumpulan kecil fungsi saja. Penguraian dapat berdasarkan kemampuan ataupun kapabilitas bisnis dari aplikasi. Penguraian juga bisa didasarkan pada kata kerja ataupun kata benda yang menggambarkan tanggung jawab apa yang diemban oleh layanan tersebut. Contohnya adalah shipping service yang bertanggung jawab terhadap pengiriman permintaan barang dan account service yang bertanggung jawab untuk mengatur akun pengguna.

B. REST API

REST API adalah sebuah implementasi dari API (*Application Programming Interface*). REST (*Representional State Transfer*) adalah suatu arsitektur metode komunikasi yang menggunakan protokol HTTP untuk pertukaran data [4].

REST API terdiri dari beberapa komponen yaitu:

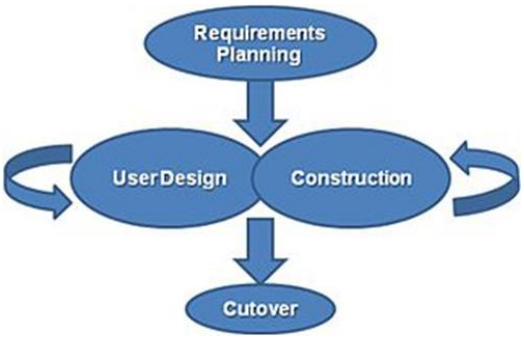
1. *URL design*
   REST API diakses menggunakan protokol HTTP, oleh karena itu diperlukan penamaan dan struktur URL yang baik dan mudah dimengerti dalam penggunannya. URL API biasa disebut sebagai *endpoint* dalam pemanggilannya. Contoh pemanggilan URL yang baik adalah seperti berikut: *users*, *users/123*, *users/123/photos* dan seterusnya.
2. *HTTP verbs*
   Adalah sebutan metode yang dilakukan ketika melakukan *request* sehingga server mengetahui apa yang ingin *client* dapatkan. Metode ini ada beberapa macam namun yang paling sering dipakai adalah: GET, POST, PUT, DELETE.
3. *HTTP response code*
   Adalah kode yang telah menjadi standar dalam menginformasikan hasil *request* kepada klien. Secara umum terdapat 3 kelompok kode yang paling sering digunakan di REST API yaitu:
   a. 2XX, yang menandakan request yang dilakukan berhasil.
   b. 4XX, yang menandakan bahwa request mengalami kesalahan pada sisi klien.
   c. 5XX, yang menandakan bahwa request mengalami kesalahan pada sisi server.
4. *Forrmat response*
   Setiap *request* yang dilakukan klien akan menerima data *response* dari *server*, *response* tersebut biasanya berupa data XML ataupun JSON. Setelah mendapatkan data *response* tersebut barulah client bisa menggunakannya dengan cara *parsing* (mengurai) data tersebut dan diolah sesuai kebutuhan.

C. Rapid Application Development

Rapid Aplication Development (RAD) adalah sebuah metode pengembangan software yang diciptakan untuk menekan waktu yang dibutuhkan untuk mendesain serta mengimplementasikan sistem, informasi sehingga dihasilkan siklus pengembangan yang sangat pendek.

Model RAD ini merupakan adaptasi dari model sekuensial linier dimana perkembangan yang cepat dicapai dengan menggunakan pendekatan kontruksi berbasis komponen. Sehingga, jika kebutuhan sistem dipahami dengan baik, proses RAD memungkinkan developer menciptakan sistem fungsional yang utuh dalam periode waktu yang sangat pendek (kurang lebih 60 sampai 90 hari).

James Martin (1990) mendefinisikan pendekatan terhadap RAD menjadi empat fase seperti yang diilustrasikan pada Gambar 2 [3].



Gbr 2 Fase RAD menurut James Martin

1. *Requirements Planning Phase*
   Mengkombinasikan elemen-elemen dari fase perencanaan dan analisis sistem pada *Systems Development Life Cycle* (SDLC). Pengguna, manajer dan staff IT mendiskusikan dan mendefinisikan kebutuhan bisnis, skala proyek, batasan, dan spesifikasi sistem. Fase

ini berakhir ketika tim setuju terhadap hasil dan mendapatkan otorisasi untuk melanjutkan ke tahap selanjutnya.

2. *User Design Phase*
   Selama fase ini, pengguna berinteraksi kepada sistem analis dan mengembangkan model dan *protoype* yang mepresentasikan proses, input dan output dari sistem.
3. *Construction Phase*
   Pada tahap ini fokus terhadap pengembangan program dan aplikasi. Dalam RAD, pengguna dapat mengusulkan perubahan ataupun peningkatan terhadap sistem yang tengah dikembangkan. Fase ini terdiri atas *coding*, *unit-integration*, dan pengetesan sistem.
4. *Cutover Phase*
   Fase ini meliputi konversi data, pengetesan, dan perubahan ke sistem baru seperti fase implementasi pada SDLC. Namun, seluruh proses tersebut dipadatkan sehingga sistem baru tersebut dapat selesai lebih cepat [3].

III. METODELOGI PENELITIAN

Kegiatan penelitian dilakukan dengan mengadopsi metode RAD. Adapun beberapa tahapan yang dilakukan adalah sebagai berikut:

A. Requirements Planning

Requirements planning meliputi perencanaan dan analisa sistem. Pada tahap ini, peneliti melakukan studi literatur, observasi dan wawancara untuk mendapatkan kebutuhan dan spesifikasi yang dibutuhkan oleh sistem. Adapun kebutuhan yang diperlukan oleh aplikasi adalah:

a. Aplikasi mampu menyimpan data yang diberikan oleh pengguna.
b. Pengguna dapat melihat dan mengunduh file yang tersimpan pada aplikasi.
c. Orang yang tidak berkepentingan tidak dapat mengakses data aplikasi.

Sedangkan spesifikasi aplikasi yang dikembangkan adalah:

a. *Framework* : Restify
b. *Language* : Javascript
c. *Platform* : NodeJS
d. *Security standard* : OAuth 2.0
e. *Database* : MySQL

B. System Design

Dari spesifikasi dan kebutuhan sistem yang telah didapatkan kemudian dilakukan implementasi dalam bentuk desain sistem. Desain yang dibuat antara lain adalah desain kamus data, desain URL, dan desain respon yang diberikan layanan.

TABEL I
KAMUS DATA CLIENTS

Nama	Tipe Data	Panjang	Keterangan
Id	Integer		*Primary key*
clientId	Varchar	256	*Unique*
clientSecret	Varchar	256	

TABEL II
KAMUS DATA TOKEN

Nama	Tipe Data	Panjang	Keterangan
Id	Integer		Primary key
clientId	String	256	Unique, foreign key dari tabel clients
Token	String	256	

C. Construction

Tahap ini berfokus pada pengembangan layanan. Selama tahap *construction* dilakukan *coding* dengan mengikuti desain sistem yang telah dibuat. Setiap fungsi yang telah selesai kemudian diuji untuk mencari *bug* yang ada.

D. Implementation

Apabila layanan sudah selesai dikembangkan, maka layanan sudah siap untuk digunakan dan diakses bersama dengan layanan lain.

IV. HASIL DAN PEMBAHASAN

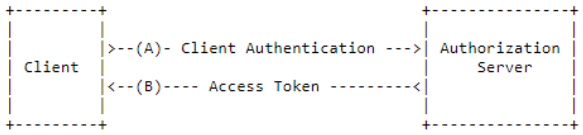
Hasil dari penelitian ini adalah sebuah aplikasi REST API layanan penyimpanan. Aplikasi ini berjalan di atas platform NodeJS dan menggunakan *framework* Restify. Hasil implementasi memiliki beberapa tambahan fungsionalitas sehingga berbeda dengan desain awal aplikasi dikarenakan mendapatkan tambahan c

Adapun fungsionalitas yang terimplementasi pada aplikasi ini adalah sebagai berikut:

1. Autentikasi OAuth menggunakan client credentials.
2. Mengunduh file.
3. Upload file.
4. Membuat direktori project.

A. Autentikasi OAuth

Pengimplementasian autentikasi OAuth sudah sesuai dengan standar yang ada dimana klien yang terautentikasi akan menerima *access token* yang dapat digunakan pada jangka waktu tertentu untuk mengakses *resource* milik aplikasi. Alur dari tipe client credentials grants diilustrasikan pada gambar berikut:



Gbr 2. Alur autentikasi OAuth

Pada Gambar 3, *client* akan meminta autentikasi kepada *authorization server* dengan mengirimkan POST *request* ke token *endpoint* yang terletak di */oauth/token* dengan parameter *grant_type = client_credentials*. Apabila klien tersebut telah terautentikasi maka *authorization server* akan mengirimkan respon sukses yang isinya adalah *access token* untuk klien seperti yang diilustasikan pada Gambar 4.

```
HTTP/1.1 200 Ok
{
    "access_token":"X1GKSJwgpmPcqoMqyd6pHdO/WFQrZ64n1HAEcQJxL+I=",
    "token_type": "Bearer"
}
```

Gbr 3. Respon sukses dari server

Access token tersebut disimpan ke dalam basis data bersama dengan waktu kadaluarsa token yang memiliki nilai *default* 1 minggu, sehingga klien dapat menggunakannya selama waktu yang telah didefinisikan. Setiap *access token* yang disimpan juga akan dihubungkan dengan ID klien yang telah terautentikasi.

B. Mengunduh File

Untuk dapat mengunduh file, sebelumnya klien harus sudah memiliki file yang tersimpan di dalam server. Apabila klien sudah pernah menyimpan file di server, maka untuk mengunduh file tersebut klien harus mengirimkan *GET request* ke alamat *[domain]/[nama_klien]/[nama_project]/[nama_file].[ekstensi_file]* seperti pada gambar 5.

```
localhost:3300/test/b0e6231b-c3cf-a228-10bd84.png
```

Gbr 4. Contoh alamat URL untuk mengunduh

Jika server tidak menemukan file yang dimaksud maka server akan memberikan respon gagal beserta pesan bahwa file yang dicari tidak dapat ditemukan (Gambar 6). Sebaliknya, apabila file ditemukan maka server akan mengirimkan file kepada klien.

```
HTTP/1.1 404 Resource Not Found
{
    "code": "ResourceNotFound",
    "message": "/test/58bccec7-988c-42a0-b446-7d483660ce1b.pngs"
}
```

Gbr 5. Respon error apabila file tidak ditemukan

C. Upload File

Fitur ini memungkinkan klien untuk memasukkan file ke dalam direktori project yang telah ada. Untuk memasukkan file maka klien harus melakukan POST *request* ke alamat *[domain]/[nama_project]/upload* dengan parameter *body* adalah file yang ingin di-upload. Apabila *request* berhasil maka file akan tersimpan di direktori yang dituju dan server akan mengirimkan alamat file berada seperti yang diilustrasikan pada Gambar 7. Sedangkan apabila direktori tidak ditemukan maka server akan mengirimkan respon gagal beserta pesan bahwa direktori yang dicari tidak ada (Gambar 8).

```
HTTP/1.1 200 Ok
{
   "url":"localhost:3300/test/58bccec7-988c-42a0-b4467d483660ce1b.png"
}
```

Gbr 6. Respon sukses upload

```
HTTP/1.1 400 Bad Request
{
    "message": "DirectoryNotExist"
}
```

Gbr 7. Respon error upload

D. Membuat Direktori

Klien dapat membuat direktori project dengan melakukan POST *request* ke alamat *[domain]/[nama_project]* seperti yang telah diilustrasikan pada Gambar sebelumnya Apabila berhasil, server akan mengirimkan respon berhasil kepada klien. Jika ternyata direktori yang dibuat sudah ada, server akan mengirimkan respon gagal beserta pesan bahwa direktori sudah ada.

```
localhost:3300/jaki
```

Gbr 8. contoh URL pembuatan direktori

```
HTTP/1.1 200 Ok
{
    "message": "DirectoryCreated"
}
```

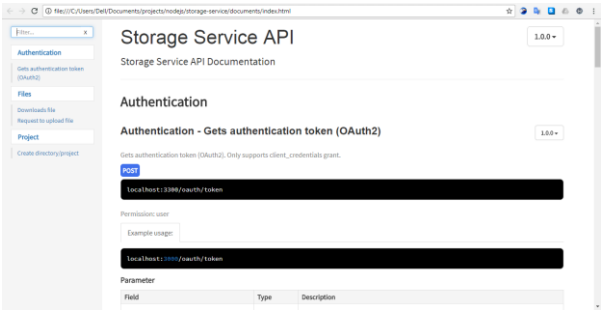
Gbr 9. Respon sukses apabila direktori berhasil dibuat

```
HTTP/1.1 400 Bad Request
{
 "error": "DirectoryExists"
}
```

Gbr 10. Respon error ketika direktori sudah ada

C. Dokumentasi API

Dalam mengembangkan sebuah aplikasi peran dokumentasi yang menjelaskan aplikasi tersebut sangatlah penting. Sebuah dokumentasi akan mempermudah pengembang lain apabila mereka mengembangkan atau memakai aplikasi tersebut. Oleh karena itu, dibuatlah dokumentasi sederhana menggunakan APIDoc yang nantinya akan mengolah tampilan dokumen menjadi halaman HTML (Gambar 12) berdasarkan kode yang telah didefinisikan sebelumnya. Isi dari APIDoc ini adalah dokumentasi dari API yang telah dikembangkan dan dapat diakses melalui browser.



Gbr 11. Dokumentasi REST API

V. KESIMPULAN

Berdasarkan hasil yang didapatkan, maka kesimpulan yang dapat diambil dari aplikasi yang telah dibuat adalah :

1. Dalam penelitian ini telah dirancang dan dikembangkannya REST API layanan penyimpanan untuk PT. XYZ
2. Layanan yang dikembangkan sudah sesuai dengan kebutuhan pengguna berdasarkan fungsi-fungsi yang telah dibuat dan setelah dilakukan pengetesan, hasil API yang didapatkan berjalan sebagaimana mestinya.
3. Aplikasi mendapatkan fitur baru yaitu dokumentasi API yang bertujuan agar memudahkan pengembang lain dalam menggunakan layanan dan mengalami perubahan standar kode sesuai dengan standar yang diminta oleh PT. XYZ.

REFERENSI

[1] Chen, Lianping, 2018. Microservices: Architecting for Continuous Delivery and DevOps. The IEEE International Conference on Software Architecture (ICSA 2018). IEEE.

[2] Richardson, C., 2017. Microservice architecture pattern. [Online] Tersedia di: http://microservices.io/patterns/microservices.html [Diakses 27 Agustus 2017].

[3] Martin, J., 1991. Rapid Application Development. 3th penyunt. s.l.:Macmillan Publishing Company..

[4] World Wide Web Consortium, 2004. Web Services Architecture. [Online] Tersedia di: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest [Diakses 31 Agustus 2017].