**BAB II**

**LANDASAN TEORI**

1. **REST (*Representational State Transfer)***
   * 1. **Definisi REST**

REST (*Representational State Transfer*) merupakan seperangkat prinsip arsitektur yang melakukan transmisi data melalui antarmuka yang terstandarisasi seperti HTTP (Roy Thomas Fielding, 2000). REST API bekerja layaknya seperti aplikasi *web* biasa. *Client* dapat mengirimkan permintaan kepada *server* melalui protokol HTTP dan kemudian *server* memberikan respons balik kepada klien. REST dikembangkan oleh Roy Fielding yang merupakan *co-founder* dari Apache HTTP Server Project. Arsitektur REST menjelaskan enam batasan. Adapun keenam batasan arsitektur REST adalah sebagai berikut (restfulapi.net):

a. *Client Server*

Batasan *client server* menjelaskan suatu antarmuka yang memisahkan bagian *client* dan *server*. Melalui pemisahan antarmuka, REST memberikan keuntungan yaitu *client* tidakperlu berurusan dengan masalah penyimpanan. Hal tersebut meningkatkan portabilitas antarmuka pengguna di berbagai *platform* dan meningkatkan skalabilitas dengan menyederhanakan komponen *server*. Pemisahan antarmuka pengguna memungkinkan komponen dapat terus berkembang secara independen. *Uniform Interface* yang menghubungkan antara *client* dan *server*.

b. *Stateless*

Batasan ini menjadi hal penting pada arsitektur REST. Batasan ini menjelaskan bahwa *server* tidak menyimpan *state* atau penanda *client*. Maksudnya adalah setiap pesan yangdikirimkan oleh *client* bersifat *self-descriptive* atau dengan kata lain setiap pesan yang dikirimkan memiliki informasi atau konteks yang cukup untuk *server* dalam memeroses pesan tersebut. Setiap *state* atau penanda *session* tersimpan pada pihak *client*.

c. *Chaceable*

Batasan ini menjelaskan bahwa respons *server* bersifat *cacheable*. Setiap respons *server* dapat disimpan secara implisit, eksplisit, atau *negotiated*.

d. *Unform Interface*

*Unform interface* menjelaskan antarmuka antara *client* dan *server*. Hal inimenyederhanakan dan memisahkan arsitektur kedua pihak, yang memungkinkan setiap bagian dapat berkembang secara independen. Batasan ini merupakan hal fundamental untuk desain RESTful. RESTful menggunakan HTTP *method* (GET, POST, PUT, DELETE) untuk menjelaskan metode permintaan, URI (*Uniform Resource Identifier*) untuk mengidentifikasi nama sumber, HTTP *response* (*status*, *body*) untuk menjelaskan informasi yang dikembalikan oleh *server*.

e. *Layered System*

Batasan ini menjelaskan bahwa pihak *client* tidak bisa secara langsung terhubung ke *server*. Terdapat perantara antara *server* dan *client*. Hal tersebut berkaitan dengan poinpemisahan antara *client* dan *server*. Perantara tersebut meningkatkan skalabilitas dengan memungkinkan *load-balancing* dan dengan menyediakan *cache* bersama. Serta *layered* *system* dapat menerapkan kebijakan keamanan.

f. *Code On Demand* (Opsional)

Batasan ini menjelaskan bahwa *server* dapat memberikan atau menyesuaikan fungsionalitas *client* secara sementara dengan mentransferkan *logic* ke dalamnya sehingga dapat dijalankan. Contohnya yaitu komponen yang dikompilasi seperti Java applet dan Javascript.

**2.1.2 Perbandingan Soap dan Rest**

Terdapat *framework* yang populer yang digunakan untuk membuat suatu layanan *web* yaitu SOAP (*Simple Object Access Protocol*) dan REST (*Representational State Transfer*). Kedua *framework* tersebut memiliki ciri khasnya yang berbeda. Adapun perbandingan antara SOAP dan REST dijabarkan pada [Tabel 1.1](#page28) (Wagh & Thool, 2012).

*Tabel 1.1 Perbandingan SOAP dan REST*

|  |  |
| --- | --- |
| **SOAP** | **REST** |
| Merupakan teknologi lama | Merupakan teknologi baru |
|  |  |
| Masih cocok untuk *enterprise* dan B2B | Belum cocok untuk *enterprise*, namun |
|  | dapat diimplementasikan pada aplikasi |
|  | perbankan |
| Ketat aturan penulisan pada interaksi | Lemah aturan pada interaksi *client-server* |
| *client-server* |  |
| Dalam hal implementasi, SOAP lebih | Namun *developer* REST berpendapat |
| unggul sebab sudah ada perangkat | bahwa, REST memiliki fleksibilitas |
| pengembangan | antarmuka |
| Memiliki muatan yang berat dalam | Transfer data sangat ringan karena melalui |
| transfer data jika dibandingkan dengan | *interface* atau yang umum dikenal sebagai |
| REST | URI |
|  |  |
| Perubahan layanan dalam kasus SOAP, | Perubahan layanan dalam kasus REST, |
| maka akan terjadi perubahan pada sisi | maka tidak memerlukan perubahan apapun |
| *Clint* | dalam kode milik *client* |
| Membutuhkan *parsing* lampiran binary | Mendukung semua tipe data secara |
|  | langsung |
| SOAP bukan infrastruktur nirkabel | REST merupakan infrastruktur nirkabel |
| Selalu mengembalikan data dalam bentuk | Fleksibilitas dalam hal jenis data yang |
| XML | dikembalikan |
| Menggunakan lebih banyak *bandwidth* | Menggunakan lebih sedikit *bandwidth* |
| sebab respons SOAP membutuhkan 10 | sebab respons yang ringan |
| kali lebih besar byte |  |
| Permintaan SOAP menggunakan POST | Dapat menggunakan permintaan GET |
| dan membutuhkan permintaan XML yang | sederhana, *intermediate proxy servers* / |
| kompleks untuk dibuat, yang mana | *reserve-proxies* dapat menyimpan respons |
| membuat respons *cache* sulit | dengan sangat mudah |
|  |  |
| Dirancang untuk menangani lingkungan | Mengasumsikan sebuah model komunikasi |
| komputasi terdistribusi | *point to point* dan bukan untuk lingkungan |
|  | komputasi terdistribusi |
|  |  |
| Lebih susah untuk dikembangkan, | Jauh lebih sederhana untuk dikembangkan |
| membutuhkan *tools* |  |
| SOAP menggunakan WS-Security. WS- | REST beranggapan bahwa transportasi |
| Security dibuat karena spesifikasi SOAP | akan berupa HTTP (atau HTTPS) dan |
| merupakan *transport-independent* dan | mekanisme keamanan yang ada di dalam |
| tidak ada asumsi mengenai keamanan | protokol akan tersedia |
| yang tersedia pada lapisan transport |  |
|  |  |
| Memiliki dukungan yang lebih baik dari | Kurangnya dukungan standar untuk |
| standar lain (WSDL, WS) dan *tools* dari | keamanan, kebijakan, dan lain – lain, |
| vendor | sehingga layanan yang memiliki |
|  | persyaratan lebih canggih maka lebih sulit |
|  | untuk dikembangkan |

* + 1. **Keunggulan REST**

Adapun beberapa keunggulan yang dimiliki oleh REST adalah sebagai berikut (Dhingra, 2013):

1. REST menyediakan infrastruktur yang bagus dalam proses *caching* melalui metode HTTP GET. Hal ini dapat meningkatkan performa jika informasi tidak diubah dan tidak dinamis.
2. REST memisahkan perspektif *server* dan *client* melalui interaksi yang menggunakan HTTP.
3. REST dapat mengembalikan respons dalam format yang beragam dan sesuai dengan permintaan *client*.
4. REST dapat dikembangkan menggunakan bahasa pemrograman manapun selama bahasa tersebut dapat membuat permintaan berbasis *web* melalui HTTP.
5. REST cocok digunakan pada aplikasi perangkat bergerak.
   1. **DJANGO FRAMEWORK**
      1. **Definisi Django Framework**

Django adalah kerangka kerja web Python tingkat tinggi yang memungkinkan pengembangan cepat dari situs web yang aman dan dapat dipelihara. Dibangun oleh pengembang berpengalaman, Django menangani banyak kerumitan pengembangan web, jadi pengguna dapat fokus pada penulisan aplikasi tanpa perlu menemukan kembali roda. Django ini gratis dan open source, memiliki komunitas yang berkembang dan aktif, dokumentasi yang bagus, dan banyak pilihan untuk dukungan gratis dan berbayar. *(Django Introduction).*

**2.3 Penelitian Terkait Dengan Pengembangan RESTful API Untuk Backend aplikasi berbasis web**

Pada penelitian kali ini, penulis membuat suatu RESTful API untuk aplikasi berbasis web (Peken 4.0). Sampai saat ini, banyak penelitian yang pernah dilakukan sebelumnya yang membahas tentang pengembangan REST API dan aplikasi berbasis web. Adapun rangkuman dari penelitian sebelumnya ditunjukkan pada [Tabel 2.2.](#page36)