**BAB III**

**METODE PERANCANGAN**

**3.1 Sistematika Perancangan**

**3.1.1** **Kasus Galat Sistem Pada KRS daring SAKTI STIKI**

Kegagalan sistem pada saat proses pengambilan KRS adalah sebuah hal yang sering sekali ditemui oleh mahasiswa khususnya pada saat pergantian semester. Menurut pengamatan peneliti, kejadian tersebut selalu terjadi berulang-ulang dikarenakan saat diakses secara bersamaan *server* mengalami *load* yang ringgi sehingga membebani sistem dan menyebabkan kegagalan sistem. Keadaan tersebut diperparah dengan KRS daring yang sifatnya seperti *battle royale* yang berdampak pada adu cepat antar mahasiswa.

Jika digambarkan kejadian ini hamper sama dengan *flash sale* yang biasa dilakukan oleh market place pada saat promo. Jadi bisa dibayangkan bagaimana besarnya *request* yang diterima oleh server. Karena itu kejadian galat sistem selalu terjadi pada saar KRS Daring setiap tahunnya.

**3.1.2 Proses Pemetaan Pelayanan**

Secara garis besar ada beberapa layanan yang diberikan oleh SAKTI STIKI, yaitu ;

1. Keuangan
2. Mata Kuliah
3. Kehadiran
4. Akademik
5. Penilaian Mahasiswa

Layanan tersebut belum termasuk dengan beberapa sistem yang juga terintegrasi dengan beberapa layanan pihak ke-3 yang digunakan oleh kampus STIKI.

**3.1.3** **Proses Pembagian Layanan**

Setelah didapatkan layanan apa saja yang digunakan pada SAKTI, maka didapatkan penggunaan seperti di bawah ini;

|  |  |  |  |
| --- | --- | --- | --- |
| **Layanan** | **Jumlah Pengguna  Rata-rata/bulan** | **presentase error** | **Rata-rata waktu** |
| Keuangan | 20 | 0.0075% | 100ms |
| Mata Kuliah | 100 | 0.05% | 1230ms |
| Kehadiran | 30 | 0.01% | 200ms |
| Akademik | 20 | 0.0155% | 1350ms |
| Penilaian Mahasiswa | 10 | 0.0003% | 150ms |

Tabel 1.1 adalah table perbandingan layanan dan tingkat kegalatan

Dari tabel diatas dapat disimpulkan layanan Mata Kuliah dan Akademik adalah 2 layanan yang paling sering mengalami galat dan banyak pengakses. Maka dari sini kita dapatkan menyimpulkan layanan mana yang paling mengalami galat dan dapat diperbaiki.

Pada dasarnya *merefactor* sebuah aplikasi *monolith* ke arsitektur *microservice* dapat dilakukan secara parsial dan hanya sebagian. Jadi dari dasar data di atas dapat disimpulkan cukup layanan yang berhubungan dengan KRS yang perlu di-*refactor* menjadi *microservices.*

**3.1.4** **Implementasi**

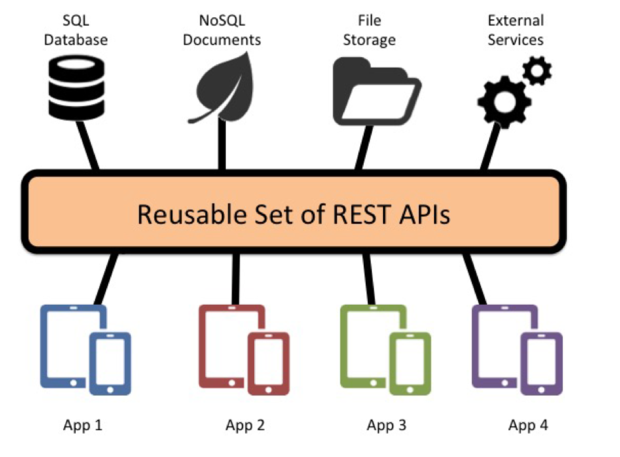
Dalam Implementasi ada tahapan yang peneliti lalui untuk membuktikan teori yang digunakan, yaitu ;

1. Pemisahan KRS daring dari SAKTI STIKI

Langkah pertama yang dilakukan adalah memisahkan layanan KRS Daring dari SAKTI STIKI dan mengembangkannya dengan arsitektur *microservices.* Dengan begini layanan akan bersifat independen dan tidak bergantung pada sistem utama SAKTI STIKI.

Tahapan lebih lanjut pemisahan layanan akan dijelaskan pada tahapan selanjunya. Pada prinsipnya kunci dari tahapan ini adalah memisahkan layanan kecil dari layanan utama. Dengan begini pengembangan kedepannya dari layanan ini akan lebih mudah dan cepat.

1. Pemisahan Backend & Frontend dengan API



Gambar 1.1 Ilustrasi REST API

Langkah selanjutnya membangun sebuah REST API yang dapat diakses dari sisi backend dan frontend. Hal ini dapat mengurangi beban *server*, beban sistem dan juga meningkatkan kemudahan pengembangan layanan ini kedepannya. Jadi Frontend dan Backend tidak terhubung secara langsung melainkan menggunakan perantara API yang kemudian akan terhubung ke *database*

1. Penulisan Ulang Aplikasi

Selanjutnya dari sisi pemprograman, dilakukan pengecekan ulang terhadap baris kode yang sudah ditulis untuk memastikan tidak ada baris yang menyebabkan pemborosan sumber daya maupun galat pada sistem secara meyeluruh.

SAKTI menggunakan platform web yang terhitung cukup mudah untuk melakukan proses pengecekan ulang. Jadi pemilihan bahasa pemprograman juga menjadi faktor pendukung dalam tahapan ini.

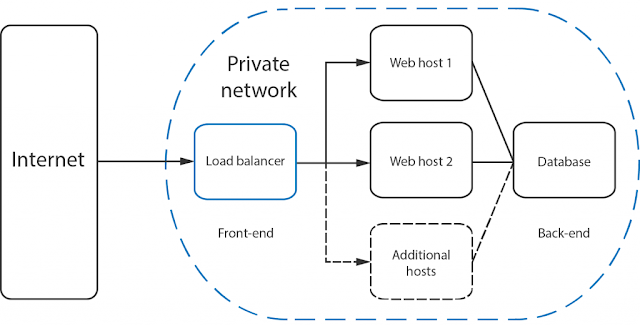
1. Optimasi *Query*

Selanjutnya kita perlu memastikan bahwa *query* atau proses pemanggilan data dari database. Seringkali terjadi kesalahan pemilihan *query* yang menyebabkan proses yang tidak optmal dan membuang banyak sumber daya dan memakan banyak waktu eksekusi.

1. Containerization dengan docker

Setelah membagi layanan ke beberapa layanan berbeda selanjutnya membedakan *instance* atau server dari layanan. Hal ini perlu dilakukan untuk memastikan bahwa layanan KRS daring benar-benar independen dan terisolasi. Sehingga layanan dapat berjalan lebih optimal karena terisolasi dalam sebuah *instance* mandiri.

1. Load Balancing

  
Gambar 1.2 Ilustrasi Load Balancing

Setelah instance dipisahkan selanjutnya proses load balancing dan manajerial sumber-sumber yang digunakan oleh layanan KRS. Setelah dipisahkan maka proses pemanggilan sumber daya secara terpisah dan lebih cepat.

**3.2** **Sumber Data**

Data didapatkan dari analisis *log server* dan layanan, serta survei ke beberapa mahasiswa yang sudah pernah menggunakan layanan dan mengalami galat ataupun pelambatan sistem.

**3.3** **Teknik Pengambilan Data**

Teknik pengambilan data menggunakan *sampling log* dan survei ke beberapa mahasiswa.

**3.4 Analisa Data**

Analisis dilakukan sebelum layanan dilakukan dan setelah layanan berjalan sesuai dengan pendekatan dan instrument yang digunakan dalam penelitian ini.