BAB 2
TINJAUAN PUSTAKA

2.1 Profil Instansi

SESKOAU (Sekolah Staff dan Komando Angkatan Udara) merupakan sekolah pendidikan TNI Angkatan Udara di Bandung yang bertugas menyelenggarakan pendidikan pengembangan umum tertinggi TNI Angkatan Udara. *LMS (Learning Management System)* SESKOAU merupakan sistem berbasis website yang dapat membantu seluruh satker (Satuan Kerja) maupun para pasis (Perwira Siswa) dalam menjalankan tugas nya selama berada di SESKOAU. Pada aplikasi LMS SESKOAU terdapat 2 jurusan yaitu SESKO (Staff Umum dan Komando) yang kedua yaitu SESAU (Staff Umum dan Administrasi).

Aplikasi ini terbagi menjadi beberapa *module*, yaitu diantaranya Akademik, Anti Plagiat, Penugasan/Assignment, DMS (Document Management System), Dashboard, Digital Activity, Penilaian, Personel, E-learning dan E-library, namum dari module-module tersebut hanya terdapat 4 module yang intensitas nya tinggi dipergunakan oleh seluruh satker maupun pasis setiap harinya. *Module* tersebut adalah Elearning, Assignment, DMS, dan Penilaian. Dari 4 *module* tersebut terdapat 7 satker yang terlibat di dalamnya, dan dari setiap satker-satker tersebut terdapat pula staf-staf yang mempunyai tugas spesifik di dalam sistem LMS. Sampai pada tahun 2021 terdapat total 350 staf, dan 210 siswa yang mempunyai akses didalam sistem LMS. Dalam proses pelaksanaan nya sampai saat ini, modul-modul tersebut sangat memungkinkan untuk diakses secara bersamaan. Sistem LMS ini dibutuhkan berjalan 24 jam dan diharuskan berjalan secara optimal dari segi *performansi*, *availability*, maupun *scalability* dikarenakan data/proses yang terjadi sangat krusial, salah satu contohnya adalah dalam proses penyuratan, sistem harus dapat *menghandle* permintaan tinggi dari pengguna secara cepat.

2.2 Sejarah Instansi

Sekolah Staf dan Komando Angkatan Udara disingkat Seskoau sebagai lembaga pendidikan pengembangan umum tertinggi tingkat TNI Angkatan Udara, mempunyai tugas menyelenggarakan pendidikan pengembangan umum tertinggi di Angkatan Udara, pendalaman materi kejuangan, pengkajian dan pengembangan, doktrin masalah-masalah Pertahanan Negara, di Dirgantara serta pengkajian masalah-masalah pendidikan dan latihan di Angkatan Udara [4].

Dengan motto “Pragnya Paramartha Jaya” yang mempunyai arti “ Ilmu Pengetahuan Pangkal Kemenangan”, sejak lahirnya tanggal 1 Agustus 1963, Seskoau telah mengalami tiga kali pergantian nama dalam kurun waktu sebagai berikut:

Seskau tahun 1963. Semula berkedudukan di jalan Budi kemuliaan No. 16 Jakarta. Sejak tahun 1966 pindah ke Lembang. Sesko ABRI Bagian Udara tahun 1974. Mulai tahun 1974 Seskau menjadi bagian dari Sesko ABRI dengan sebutan Sesko ABRI Bagian Udara. Operasi Pendidikan berada di bawah kendali Komandan Jenderal Sesko ABRI yang bertanggung jawab kepada Menhankam / Pangab, sedangkan wewenang pembinaan Sesko ABRI Bagian Udara tetap pada TNI AU. Seskoau tahun 1984 – sekarang. Berdasarkan Keputusan Pangab Nomor: Kep/07/P/III/1984 tanggal 21 Maret 1984 dan Perintah Pangab Nomor: Prin/06/P/IV/1984 wewenang Komando Pengendalian Operasi Pendidikan Sesko Matra dikembalikan kepada masing-masing Kepala Staf Angkatan dan Kapolri. Sesko ABRI Bagian Udara menjadi Sekolah Staf dan Komando TNI Angkatan Udara (Seskoau). Berdasarkan Surat Keputusan Kepala Staf TNI AU Nomor: Skep / 34 / VI / 1987, pada kurun ini telah terjadi penggabungan Sekolah Staf Angkatan Udara (Sesau) yang lahir tahun 1977 di Jakarta semula bernama Kursus Jabatan Perwira Staf (Susjabpastaf). Dengan penggabungan Sesau ini pada tahun pelajaran 1988 dan 1989 penyelenggaraan pendidikan di Seskoau dibagi dalam dua jurusan, yaitu Jurusan Staf Umum dan Komando dan Jurusan Staf Umum dan Administrasi. Pada tahun pelajaran 1990 kembali menjadi satu jurusan yaitu Pendidikan Staf Umum dan Komando saja.

Saat ini penyelenggaraan pendidikan Kursus Staf (Susstaf) Bidang Matra Udara diselenggarakan di Seskoau lembang, berdasarkan Keputusan Kepala Staf Angkatan Udara nomor: Kep.688/XII/2013 tentang kurikulum Kursus Staf Bidang Matra Udara, dan Angkatan Pertama dimulai pada awal tahun 2014 dilaksanakan selama 5 bulan.

Selain Susstaf angkatan pertama, Tahun 2014 banyak sekali kegiatan di Seskoau yang bernilai sejarah, Sejak ditandatanganinya Piagam Kerja Sama antara Universitas Pertahanan Indonesia (Unhan) dan Sekolah Staf dan Komando Angkatan Udara nomor PKS/110/1/2014 dan Perjanjian Kerja Sama Nomor:Perjama/01/I/2014 tanggal 21 Januari 2014, merupakan catatan sejarah tersendiri bagi Seskoau karena mulai tanggal tersebut Perwira Siswa angkatan 51 dan Pasis angkatan berikutnya yang memenuhi Syarat setelah Lulus Seskoau berhak menyandang gelar S-2 dengan sebutan Master Megister Terapan Pertahanan Udara / M.Tr (PU).

Pada tanggal 17 Februari 2014, Seskoau bekerja sama dengan Yayasan Adi Upaya (Yasau) dengan nomor: PKS/2/II/2014 dan nota kesepahaman nomor: Kesepahaman/6/NK/II/Yasau dan Perjanjian kerjasama antara Seskoau dan Universitas Nurtanio Bandung (Unnur) nomor: perjama/2/II/2014 dan nomor: Unnur/02/Perjama/II/2014 untuk melaksanakan pendidikan Program Pasca Sarjana (S-2). Kebijakan pembentukan program Pasca Sarjana (S-2) Administrasi Publik bertujuan membekali perwira lulusan Seskoau agar memiliki kemampun kognitif, afektif dan psikomotorik yang sesuai dengan kaidah pendidikan tinggi.

2.3 Visi dan Misi Instansi

1. VISI

Mengembangkan kemampuan perwira TNI AU yang handal, profesional dan proporsional yang dilandasi jiwa kejuangan patriotism. [4]

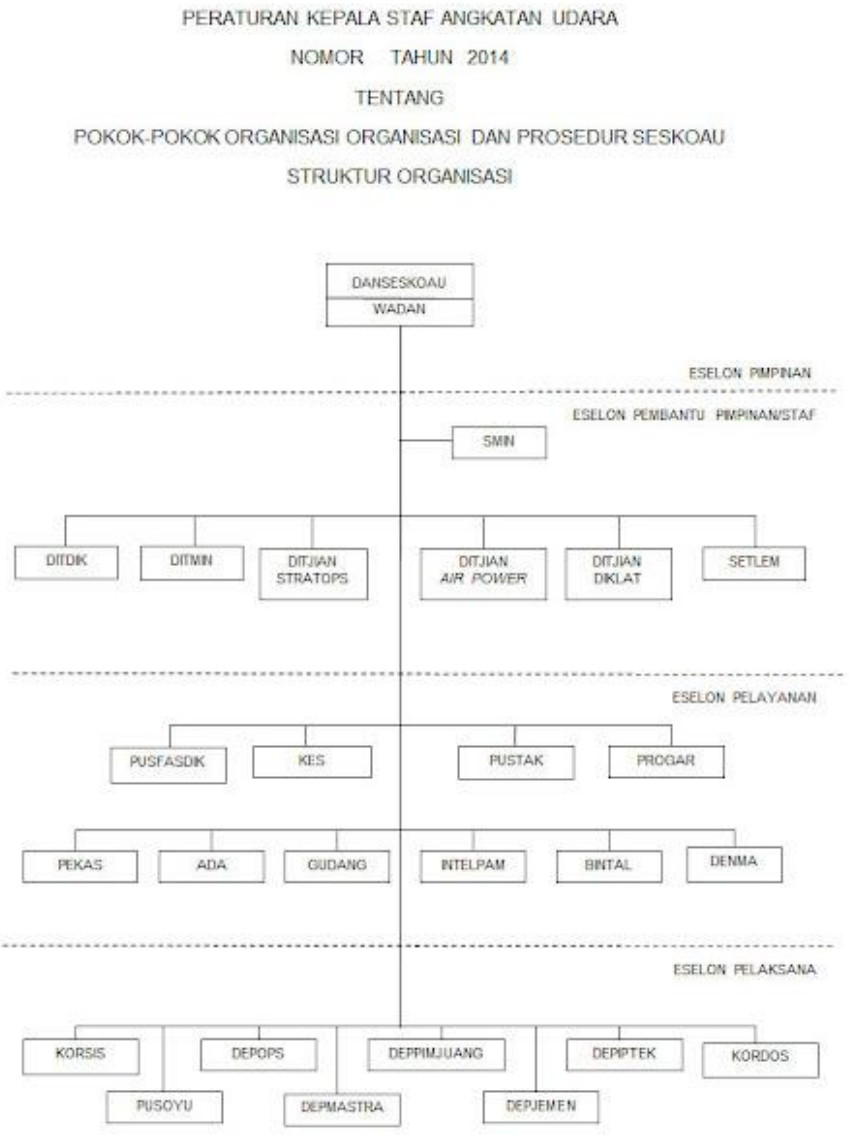
2. MISI

Menyelenggarakan pendidikan pengembangan umum tertinggi di lingkungan TNI AU, menyelenggarakan pendalaman materi kejuangan, dan menyelenggarakan

pengkajian dan pengembangan doktrin serta kekuatan matra udara tingkat strategi [4].

2.4 Struktur Organisasi dan Uraian Tugas

Berikut ini merupakan struktur organisasi di SESKOAU [5]



Gambar 21 Struktur Organisasi SESKOAU

2.5 Web Service

Web service merupakan suatu standar yang digunakan untuk melakukan pertukaran data atau komunikasi antar aplikasi atau sistem. Web service dirancang untuk memiliki antarmuka yang digambarkan dalam format yang dapat diproses suatu aplikasi. Protokol yang paling umum digunakan untuk komunikasi yaitu HTTP sedangkan format pertukaran data yang biasa digunakan sebagai sarana komunikasi suatu layanan adalah SOAP dan REST. Penelitian ini sendiri menggunakan REST dalam pengimlementasian suatu *service.* REST sendiri merupakan suatu arsitektur metode komunikasi yang menggunakan protokol HTTP untuk pertukaran data. Setiap *request* yang dilakukan, akan terdapat suatu metode yang digunakan agar server mengerti apa yang sedang di *request client*. Metode tersebut diantaranya sebagai berikut :

1. GET

Metode *GET* merepresentasikan suatu data yang spesifik, metode ini digunakan untuk membaca atau mendapatkan data dari sumber.

2. POST

Metode *POST* digunakan untuk membuat data baru dengan menyisipkan data pada bagian *body* saat *request* dilakukan, metode ini biasanya akan menyebabkan perubahan suatu status atau efek samping pada server.

3. PUT

Metode *PUT* biasanya digunakan untuk melakukan perubahan data pada server.

4. DELETE

Metode *DELETE* biasanya digunakan untuk menghapus suatu spesifik data pada server.

Selain metode, pada RESTful API terdapat yang namannya status code atau biasa disebut HTTP *Response* Code. HTTP response code adalah kode standarisasi dalam menginformasikan hasil *request* kepada *client*. Terdapat 3 bagian pada *Response Code* yaitu :

1. Redirection Message

Redirection Message terjadi biasanya karena suatu *resource* yang diminta oleh client kepada server terdapat kesalahan, berikut merupakan kesalahan yang mungkin terjadi beserta *response code* yang diterima :

- 300 *Multiple Choice*
- *301 Moved Permanently*
- *302 Found*
- *303 See Other*
- *304 Not Modified*
- *305 Use Proxy*
- *306 unused*
- *307 Temporary Redirect*
- *308 Permanent Redirect*

2. *Client Error Message*
   - *400 Bad Request*
   - *401 Unauthorize*
   - *402 Payment Required*
   - *403 Forbidden*
   - *404 Not Found*
   - *405 Method Not Allowed*
   - *406 Not acceptable*
   - *407 Proxy Authentication Required*
   - *408 Request Timeout*
3. *Server Error Response*
   - *501 Not Implemented*
   - *502 Bad Gateway*
   - *503 Service Unavailable*
   - *504 Gateway Timeout*

2.6 Microservices

Microservices adalah desain arsitektur aplikasi yang memecah aplikasi menjadi service - service kecil yang terpisah sesuai dengan fungsinya sehingga

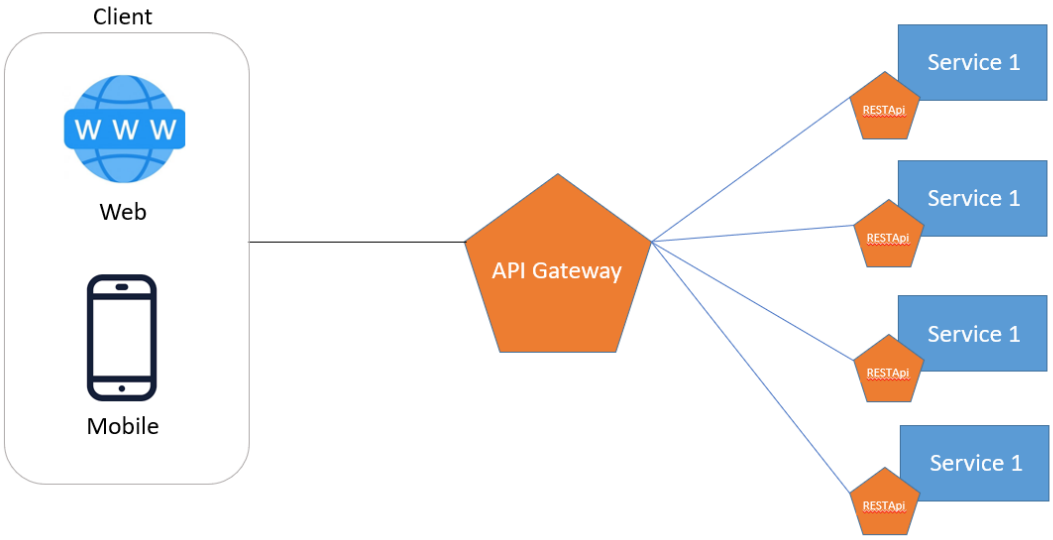
fungsi akan bekerja secara *independent.* Dalam hal ini setiap service mungkin saja dapat mempunyai teknologi stack yang berbeda antara satu dengan yang lain tergantung dari kebutuhan service. Ini artinya akan terdapat teknologi yang berbeda – beda dalam satu aplikasi besar. Adapun beberapa keunggulan yang dapat diambil dari arsitektur microservices antara lain:

1. Menghasilkan produk yang lebih cepat dan mengurangi ketergantungan antara satu tim dengan yang lain.
2. Menjadikan setiap service bekerja secara paralel sehingga meningkatkan performa dari web services.
3. Dapat mempunyai teknologi yang berbeda pada setiap service nya sehingga service lebih optimal karena hanya teknologi yang dirasa paling baik yang akan digunakan pada setiap servicenya.
4. Menghasilkan teknologi yang beragam sehingga ketika satu teknologi yang digunakan gagal dan tidak optimal maka akan mudah untuk memperbaikinya

Meski memiliki banyak kelebihan, arsitektur microservices juga memiliki kekurangan karena semakin besar suatu web service menjadikan arsitektur menjadi semakin kompleks. Hal ini berbanding lurus dengan keunggulan yang ditawarkan pada arsitektur ini. Dalam penerapannya arsitektur ini memiliki bagian-bagian lain yang harus diperhatikan seperti:

2.3.1. API Gateway

Seperti yang telah dijelaskan sebelumnya bahwa arsitektur *Microservice* akan membagi satu halaman sebuah website kedalam banyak layanan atau *service*, hal tersebut mengakibatkan *client* suatu aplikasi yang berbasis *microservice* akan mengakses suatu layanan secara *individual.* Oleh sebab itu diperlukan API Gateway sebagai titik masuk awal ketika *user* ingin mengakses suatu layanan. Gambaran mengenai API Gateway dapat dilihat pada gambar 21 API Gateway

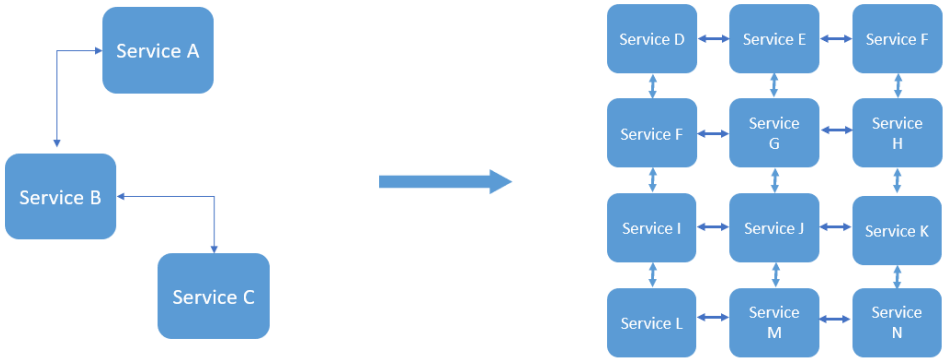


Gambar 22 API Gateway

Fungsi API Gateway selain sebagai jembatan awal antara *client* dan *service*, selain itu API Gateway juga bertugas untuk *menghandle* proses seperti *management* API, *authentication, Authorization, monitoring* dan *load balancing*.

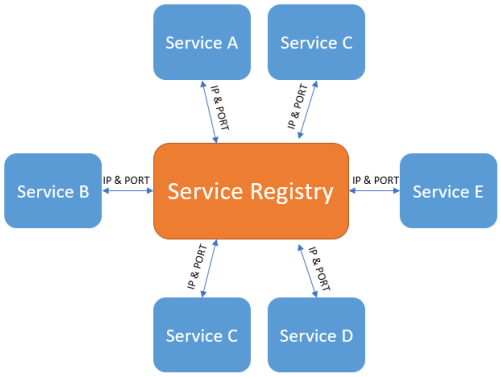
2.3.2. Service Discovery

Seperti yang dijelaskan sebelumnya, *Microservice* akan membagi aplikasi kedalam *service-service* yang spesifik, oleh karena itu *service-serivice* tersebut harus dapat saling berkomunikasi dengan baik, *service discovery* bertugas sebagai alat komunikasi antar *service* di dalam *microservice.*



Gambar 23 *Service Discovery Communication*

Pada arsitektur *microservice*, jumlah *service* yang digunakan akan terbilang banyak, akibatnya sangat sulit bagi satu *service* dalam mempertahankan alamat IP dari semua *service* yang saling berkomunikasi ke dalam satu file konfigurasi. Oleh sebab itu alamat IP seluruh *service* harus ditetapkan secara dinamis agar *microservice* dapat menemukan lokasi jaringan *service* lain walaupun alamat IP pada *service* tertentu mengalami perubahan.



Gambar 24 *Service Registry*

Alamat IP beserta port yang digunakan oleh seluruh *service* akan tersimpan pada *central registry* sehingga ketika alamat IP dan port yang digunakan pada suatu *service* mengalami perubahan dengan alasan tertentu, maka IP dan port tersebut akan tersimpan ulang pada *registry*.

2.3.3. Service *Invocation*

Pada arsitektur *monolith* proses pemanggilan suatu fungsi menjadi sederhana karena akan didefinisikan pada sistem *monolith* itu sendiri, setiap aksi atau fungsi dari aplikasi akan dilakukan pada sistem *monolith*. Hal tersebut tidak akan menjadi masalah ketika aplikasi yang dibangun tidak memiliki kompleksitas yang tinggi. Pada arsitektur *microservice*, aplikasi akan dipecah sesuai dengan domainnya sehingga pemanggilan antar *service* tidak akan terjadi dengan mudah. Diperlukan adanya mekanisme yang dapat mengintegrasikan antar aplikasi pada arsitektur *microservice*, secara garis besar terdapat dua jenis cara mengintegrasikan antar aplikasi pada *microservice*, yaitu sebagai berikut [4].

1. *API-Driven*

Pada *API-Driven*, mekanisme komunikasi antar *service* akan dilakukan melalui pemanggilan API, contohnya adalah menggunakan RESTful atau *Remove Procedure Call* (RPC) dan komunikasi akan dilakukan secara *synchronous*, artinya pada komunikasi ini *client* akan mengirimkan *request* kepada suatu *services* dan menunggunya sampai terdapat *response* dari *request* yang terjadi diterima oleh *client*. *API-Driven Architecture* biasanya disebut dengan *Orchestration Pattern*, yang dalam artian akan ada satu *service* yang berperan sebagai orkestrator. *Service* orkestrator ini bertanggung jawab terhadap jalannya alur bisnis pada aksi tersebut [4]. Kekurangan dari mekanisme seperti ini salah satunya adalah dari segi ketergantungan suatu *service* orkestrator dengan *service – service* lainnya sehingga jika terdapat kesalahan pada salah satu *service* maka akan mempengaruhi *service – service* lainnya.

2. *Event-Driven*

Berbeda dengan *API-Driven*, pada mekanisme ini komunikasi akan dilakukan secara *asynchronous* menggunakan konsep *message event*. Mekanisme ini tidak akan menunggu *response* dari suatu *service* selesai untuk dapat melanjutkan ke proses selanjutnya. Kelebihan mekanisme ini dibandingkan dengan *API-Driven* adalah setiap *service* tidak akan saling bergantung, artinya setiap *service* akan mengetahui apa yang akan dilakukan ketika menerima suatu *event* dan akan bertindak sesuai dengan domainnya.

2.3.4. *Load* Balancing

Load balancing adalah cara untuk mendistribusikan beban kerja ke berbagai *resources* untuk membantu jaringan dalam mengatasi terjadinya server *down* yang akan mengganggu *service* dalam memberikan kinerja optimal kepada pengguna [5]. Tujuan dari *load balancing* adalah untuk memaksimalkan pemakaian sumber daya dan mempercepat kinerja suatu *service*, sehingga sistem dapat tetap bekerja secara optimal meskipun terjadi meningkatan pemakaian oleh pengguna. Berikut merupakan berbagai macam algoritma yang digunakan pada load *balancing*.

1. *Round Robin*

Pada algoritma ini permintaan yang berasal dari *client* akan didistribusikan ke server aplikasi secara bergantian dan merata [5] . Sebagai contoh jika terdapat 3 server aplikasi, *client* pertama melakukan permintaan dan akan diarahkan kepada server aplikasi yang pertama, *client* kedua melakukan permintaan kepada server aplikasi kedua, *client* ketiga akan melakukan permintaan kepada server aplikasi yang ketiga, dan *client* keempat akan melakukan permintaan kepada server aplikasi yang pertama dan seterusnya.

2. *Weighted Round Robin*

Algoritma *weighted round robin* adalah modifikasi versi algoritma *round robin*, tetapi dengan memperhitungkan karakteristik server aplikasi yang berbeda beda dan relatif [6]. Pihak administrator akan menentukan bobot setiap server aplikasi berdasarkan kriteria yang dipilih untuk mempertunjukan kapabilitas server aplikasi dalam menangani *traffic.* Sebagai contoh server aplikasi pertama lebih bagus dari segi spesifikasinya dibandingkan server aplikasi kedua dan ketiga, maka bobot server aplikasi pertama akan lebih besar dibandingkan server aplikasi kedua dan ketiga, Ketika terdapat 5 permintaan *client* secara sekuensial maka permintaan kesatu dan kedua akan diterima oleh server aplikasi yang pertama, permintaan ketiga akan diterima server aplikasi kedua, permintaan keempat akan diterima server aplikasi ketiga dan permintaan kelima akan diterima server aplikasi yang pertama,

3. *Least Connection*

Algoritma ini termasukan kedalam *dynamic load balancing algorithm*. Pada algoritma ini permintaan klien akan didistribusikan ke server aplikasi dengan jumlah koneksi aktif paling sedikit pada saat permintaan klien diterima, jadi pembagian beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah server. Server dengan koneksi yang paling sedikit akan diberikan beban berikutnya [5].

4. *Weighted Least Connection*

Algoritma *weighted least connection* dibangun diatas algoritma *least connection*, tetapi dengan memperhitungkan karakteristik server aplikasi yang

berbeda beda. Pihak administrator akan menentukan bobot setiap server aplikasi berdasarkan kriteria yang dipilih untuk mempertunjukan kapabilitas server aplikasi dalam menangani *traffic*. Perbedaan nya terletak pada prioritas permintaan klien akan dikirimkan kepada server aplikasi dengan mempertimbahkan berdasarkan koneksi yang aktif dan bobot dari server yang telah ditentukan.

2.7 UML (Unified Modeling Language)

Dalam menentukan domain-domain apa saja yang akan ada nantinya, maka dibutuhkan suatu pemodelan yang dapat mengatasi masalah tersebut. Sehingga didapatlah solusi yaitu dengan menggunakan Class Diagram. Class Diagram menjelaskan struktur dan deskripsi kelas, package, metode, dan object yang berhubungan antar kelas. Terdapat beberapa elemen-elemen yang ada pada class diagram antara lain [7]:

a. Nama Class

Class menggambarkan kumpulan dari suatu object yang mempunyai karakteristik yang sama, dengan demikian dibutuhkan penamaan dari suatu object yang dapat mewakili dari keseluruhan object. Nama class umumnya bersifat umum yang bersifat mengategorikan object tersebut misalnya: BMW, Mercedes, Ferrari merupakan kumpulan object dari suatu mobil, maka nama class dapat diisi dengan mobil.

b. Atribut

Seperti yang sudah dibahas sebelumnya bahwa suatu class menyatakan kumpulan dari object, dan setiap class akan mempunyai karakteristik yang berbeda. Karakteristik inilah yang kemudian dinamakan atribut, atribut dapat berupa ciri dari suatu object. Dalam mendefinisikan atribut ada beberapa hal yang harus dipelajari:

1) Visibility

Menyatakan hubungan sifat atribut dapat terlihat oleh class lain. Visibility dapat dibagi menjadi beberapa macam yaitu: Private, yang berarti bahwa atribut tersebut hanya bisa digunakan pada class tersebut dan tidak bisa dipanggil di luar

class. Protected, yang berarti hanya bisa dipanggil oleh class yang berhubungan dan class lain yang mewarisinya. Public, merupakan class yang dapat dipanggil oleh class lain tanpa terkecuali.

2) Nama

Mendefinisikan dari nama atribut yang akan digunakan. Umumnya nama atribut memiliki ciri khas yang pendek dan didahului oleh huruf kecil, namun untuk huruf awal pada kata berikutnya menggunakan huruf kapital.

3) Tipe

Tipe pada class menyatakan klasifikasi data pada suatu atribut. Umumnya tipe data merujuk pada suatu bahasa pemrograman tertentu. Terdapat beberapa tipe data yang sering digunakan, seperti: int, string, boolean, dll.

c. Operation

Merupakan salah satu spesifikasi dari class yang menyatakan perilaku yang dapat dilakukan oleh suatu object. Mirip seperti atribut, operation memiliki sifat yang akan berhubungan jika digunakan pada class lain, yaitu:

1) Visibility

Menyatakan hubungan sifat operation dapat terlihat oleh class lain. Visibility dapat dibagi menjadi beberapa macam yaitu: Private, yang berarti bahwa operation tersebut hanya bisa digunakan pada class tersebut dan tidak bisa dipanggil di luar class. Protected, yang berarti hanya bisa dipanggil oleh class yang berhubungan dan class lain yang mewarisinya. Public, merupakan class yang dapat dipanggil oleh class lain tanpa terkecuali.

2) Nama

Mendefinisikan dari nama operation yang akan digunakan. Umumnya nama operation memiliki ciri khas yang pendek dan didahului oleh huruf kecil, namun untuk huruf awal pada kata berikutnya menggunakan huruf kapital.

3) Parameter

Mendefinisikan variabel masukkan dari luar operation yang nantinya akan digunakan di dalam operation. Parameter akan dibutuhkan apabila variabel yang dimaksud dapat berupa atribut yang merupakan hasil dari operation lain.

Penamaan parameter sama seperti penamaan pada operation yang diawali huruf kecil dan huruf kapital untuk huruf awal pada kata berikutnya.

4) Tipe Return

Merupakan tipe pengembalian pada suatu operation, tipe ini bersifat optional sesuai dengan jenis operation. Tipe pengembalian bisa berupa tipe data tertentu atau object dari suatu class. Pada operation jenis fungsi tentunya akan terdapat tipe pengembalian, namun untuk jenis prosedur tidak terdapat pengembalian di dalamnya.

d. Relationship

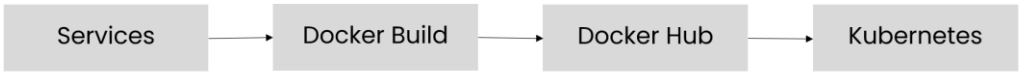
Relationship berfungsi dalam mengetahui hubungan antara satu class dengan class lainnya. Tanpa adanya relationship, kita tidak didapat mengetahui mekanisme dari sistem yang akan diteliti.

2.8 Containerizing Applications (Docker)

Containerizing Applications merupakan teknologi *virtualization container* atau wadah pada tingkat OS (*Operating System*). Teknologi ini menyediakan beberapa fitur yang mendukung untuk mengimplementasikan *microservice* dimana dapat digunakan untuk membangun, mendistribusikan, dan menjalankan aplikasi secara portabel dan ringan, selain itu pada aspek biaya pun akan berkurang, itu dikarenakan fungsi dari *docker container* akan menggantikan mesin virtual tradisional [8].

Penelitian ini akan memanfaatkan *Docker* sebagai *containerizing application* pada *web services* yang bertujuan untuk mengisolasi *service* dan untuk menjadikan *service* tersebut bersifat otonom atau independen. Setiap *service* yang telah dipecah menjadi aplikasi – aplikasi yang mempunyai fungsi spesifik akan di *build* untuk mengasilkan sebuah *image docker*. *Image docker* yang sudah terbuat selanjutnya akan di *upload* ke *container registry* bernama *docker hub*. *Container registry* ini berfungsi untuk menyimpan seluruh *image docker* yang telah dibuat sebelumnya. Seluruh *image* yang sudah dibuat akan digunakan oleh *container orchestrator* untuk melakukan proses *deployment*. Berikut ini merupakan proses

deployment yang akan dilakukan, dimulai dari memecah suatu aplikasi sampai dengan *deployment* menggunakan *kubernetes.*



Gambar 25 *Docker* Proses

2.9 Code Refactoring

Code refactoring adalah perubahan yang dilakukan pada struktur internal dari sebuah software agar lebih mudah dipahami tanpa mengubah fungsi dari kode itu sendiri. Tujuan melakukan *code refactoring* adalah membuat perbaikan atau perubahan pada struktur code, selain itu tujuan nya adalah untuk membuat kode lebih dapat diperbaiki ketika mengalami error atau memiliki *maintainability* yang tinggi. Pada proses nya tidak menutup kemungkinan *developer* akan menemukan kerentanan pada kode yang bahkan sebelumnya tidak ditemukan pada saat melakukan testing dan proses *code refatoring* ini tidak menutup kemungkinan akan menimbulkan kesalah baru pada kode atau bahkan kode jadi tidak berfungsi [2]

2.10 Kubernetes

Saat suatu aplikasi yang terdiri dari banyak container bekerja bersamaan dibutuhkan suatu mekanisme untuk mengontrol dan mengatur setiap container yang berjalan. Salah satu teknologi yang berkembang saat ini berfungsi mengatur container yang banyak dinamakan dengan Container Orchestration. Container orchestration diperlukan dalam suatu aplikasi untuk mengotomatisasi container, sehingga proses deployment, scaling, dan proses services antara container dapat diotomatisasi. Teknologi ini juga berfungsi mengetahui container tersedia yang terdapat pada setiap host. Selain itu beberapa orchestration juga mendukung penanganan ketika suatu container mengalami kegagalan baik dari segi sistem atau hardware, sehingga container dapat dengan cepat diubah status ketersediaannya hingga container kembali siap atau dihilangkan dari daftar

ketersediaan. Pada penelitian ini container orchestration yang digunakan adalah Kubernetes. Dalam container orchestration Kubernetes terdapat beberapa jenis resource didalamnya beberapa diantaranya yaitu:

1. Pods

Pods merupakan kumpulan aplikasi didalam container dan dijalankan secara bersamaan. Meskipun demikian pods bukanlah sebuah container, sebuah pods dapat terdiri dari beberapa container didalamnya. Jadi setiap container yang ada didalam suatu pods pasti akan berjalan dalam satu mesin yang sama dan node yang sama didalam kubernetes.Pada kubernetes pods merupakan bagian terkecil yang dapat dideploy. Karena didalam kubernetes pods dapat berisi beberapa container itu berarti setiap container akan memiliki IP dan port yang sama. Serta penyimpanan container pada pods akan sama dan saling berbagi. Namun pada sebuah pods akan tetap terisolasi dengan pods lain bahkan dengan pods yang berada satu node yang sama.

2. Service

Service merupakan kumpulan dari pods dengan jenis sama yang berjalan dalam kubernetes cluster. Dalam suatu cluster dapat memiliki banyak services didalamnya, hal ini dikarenakan suatu services dapat berjalan tanpa membutuhkan banyak resource prosesor dan ram. Service dalam kubernetes telah mendukung banyak fitur didalamnya salah satunya adalah service discovery. Fitur ini menjadikan proses pencarian suatu service yang tersedia dan manajemennya lebih mudah untuk dilakukan.

3. Deployment

Pada kubernetes untuk mengelola service yang berjalan dibutuhkan konfigurasi yang berfungsi mengatur tingkah laku service, konfigurasi ini dinamakan deployments. Deployment juga berfungsi dalam menentukan versi dari image yang akan diambil untuk dideploy pada kubernetes. Selain itu deployments juga mendukung fitur availability pada services yang berjalan. Sehingga ketika suatu services mengalami kegagalan atau error, services dapat dengan mudah diduplikasi untuk mendukung ketersediaan web services. Dalam Kubernetes untuk mendeploy container menjadi suatu service yang berdiri sendiri dibutuhkan

konfigurasi untuk memanajemen tingkah laku dari service baik dari segi url docker image, nama service, dan lain sebagainya. Sebagai solusi untuk hal tersebut, Kubernetes menggunakan file yaml untuk melakukan konfigurasi pada clusternya. Konfigurasi yang dilakukan meliputi deployment, pods, service, ingress, dll.

2.11 REST

REST (Representational State Transfer) merupakan protokol standar dalam suatu arsitektur web service. Dalam hal ini REST menggunakan protocol HTTP (Hypertext Transfer Protocol) dalam melakukan proses pertukaran data antara client dan server. Dengan demikian interaksi yang terjadi antara client dan server menggunakan interface yang dapat diakses oleh protokol HTTP, protokol ini sering disebut juga sebagai API (Application Programming Interface). Saat mengakses suatu resource, REST menggunakan konsep URI (Uniform Resource Identifier) yang memungkinkan satu service memiliki alamat yang berbeda dengan service lain dalam server yang sama. Dalam pembangunan API RESTful Web Service terdapat beberapa prinsip yang harus dipenuhi :

1. Setiap resource memiliki URI yang berbeda

Setiap resource memiliki URI yang berbeda dalam pengaksesannya, seperti halnya nomor KTP yang selalu berbeda antara satu dengan yang lain. URI yang digunakan juga diharapkan dapat dibaca dengan mudah oleh developer.

2. Menggunakan metode standar HTTP

Dalam mengakses suatu resource dibutuhkan metode-metode untuk mengetahui proses apa yang akan dikirimkan menuju server. Metode HTTP ini terdiri dari 8 buah, antara lain:

1. GET
2. POST
3. PUT
4. DELETE
5. OPTIONS
6. HEAD
7. TRACE

8. CONNECT

Sedangkan metode-metode yang akan sering digunakan pada penelitian ini dapat dilihat pada tabel

Tabel 21 *HTTP Method*

Method	Action	Status Code
GET	Meminta data terhadap *resource* yang ditentukan, metode ini hanya untuk mengambil data	200 OK : berhasil mendapat data 404 Not Found : *resource* yang diminta tidak tersedia 500 Internal Server Error : terdapat kesalahan pada server
POST	Mengirim data kepada *resource* yang ditentukan.	201 Created : *resource* baru berhasil ditambahkan. 400 Bad Request : *parameter* yang dikirim tidak sesuai dengan format ketentuan di server. 500 Internal Server error : terdapat kesalahan pada server
PUT	Mengubah data kepada *resource* yang ditentukan.	200 OK : *resource* berhasil diubah 400 Bad Request : *parameter* yang dikirim tidak sesuai dengan ketentuan di server. 500 Internal Server error : terdapat kesalahan pada server
DELETE	Menghapus data kepada *resource* yang ditentukan.	200 OK : *resource* berhasil dihapus.

		400 Bad Request : *parameter* yang dikirim tidak sesuai dengan ketentuan di server. 500 Internal Server error : terdapat kesalahan pada server

2.12 JSON

JSON (*JavaScript object notation*) merupakan format yang menyimpan informasi terstruktur dan biasanya digunakan untuk mentransfer data antara server dengan klien. Kelebihan JSON dibandingkan format pertukaran informasi yang lainnya adalah dari segi struktur datanya yang sederhana dan mudah dipahami. Selain itu JSON tidak hanya dapat digunakan pada bahasa pemrograman JavaScript tetapi juga mendukung pada bahasa pemrograman yang lain seperti PHP, Python, Ruby, C++, dan lain lain.

2.13 DDD (Domain Driven Design)

Pengembangan perangkat lunak sering digunakan untuk mengotomatisasi proses yang terdapat di dunia nyata atau memberikan solusi untuk masalah kehidupan yang nyata. Proses bisnis pada perangkat lunak digambarkan oleh masalah di kehidupan nyata. Perangkat lunak dengan skala besar dibangun dengan kode-kode yang rumit dan kompleks sehingga ketika mencari suatu masalah pada perangkat lunak, *developer* cenderung akan menghabiskan banyak waktu pada kode, padahal masalah tersebut dapat diatasi dengan memandang perangkat lunak sebagai suatu objek dan metode. [9]

DDD (Domain Driven Design) adalah pendekatan pengembangan perangkat lunak yang membantu dalam mengimplementasikan perangkat lunak menjadi lebih akurat sesuai dengan proses bisnis yang akan berjalan pada aplikasi. Penggunaan konsep DDD dapat membantu dalam proses perancangan

microservice sehingga setiap layanan akan memenuhi kebutuhan fungsional bisnis. Tahapan-tahapan dalam

Domain Driven Design menggabungkan penerapan desain dan development practice, dan menunjukan bagaimana desain dan proses development dapat bekerja sama untuk menciptakan solusi yang lebih baik. Desain yang baik akan mengakselerasi pengembangan, sementara umpan balik dari proses development akan menyempurnakan desain [9]. Adapun tahapan-tahapan dalam metode ini adalah sebagai berikut.

1. Bounded Context

Bounded Context merupakan deskripsi tentang batasan-batasan dimana *model* didefinisikan dan berlaku. Fungsi utama dari bounded context adalah untuk menjaga agar sebuah context tetap konsisten dalam menggunakan modelnya sesuai dengan batasan yang sudah ditetapkan [2]. Selain itu tujuan dari *bounded context* juga adalah untuk mendukung setiap *service* dapat berjalan secara *independent.*

2. Context Map

Context Map mendeskripsikan hubungan antar *bounded context* dengan kumpulan *pattern*. Keragaman perspektif ini memungkinkan untuk mendapatkan gambaran holisitik tentang tim dan hubungan konteks yang terikat. [10]. Dalam membuat context map terdapat jenis relasi antara masing-masing bounded context, berikut adalah jenis relasi yang terdapat dalam context map:

1. Partnership

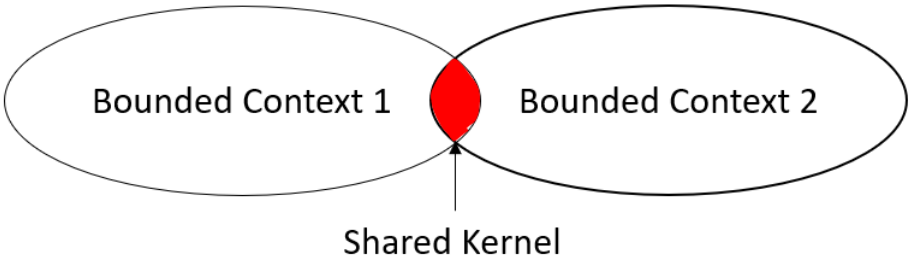
Hubungan antar *bounded context* ini terjadi ketika 2 tim yang bekerja pada 2 konteks mempunyai serangkaian tujuan yang selaras dan bergantung pada kedua konteks tersebut. Gambaran antar konteks tersebut dapat dilihat pada gambar 24

Gambar 26 *Context Map Partnership*

Integrasi antar konteks bersifat dua arah. Tidak ada batasan dalam bahasa yang digunakan pada dua tim yang berbeda, Tim akan menyelesaikan perbedaannya, dan memilih solusi yang paling tepat. Selain itu, kedua belah pihak bekerja sama dalam menyelesaikan masalah integrasi yang mungkin muncul. Kolaborasi yang matang, komitmen tingkat tinggi, dan sinkronisasi yang sering antar tim semuanya diperlukan untuk integrasi yang berhasil dengan cara ini. [10]

2. Shared Kernel

Shared Kernel adalah cara yang lebih formal untuk mendefinisikan kontrak antara beberapa konteks yang dibatasi. Untuk lebih detail dapat dilihat pada gambar 25

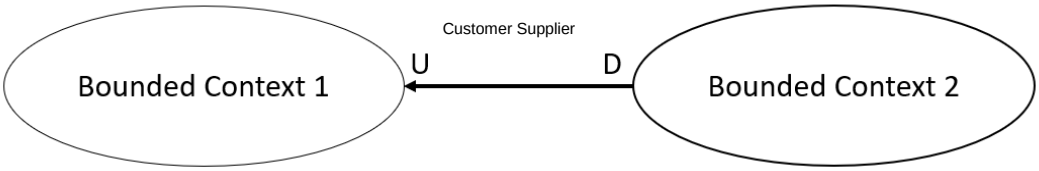


Gambar 27 *Context Map Shared Kernel*

Shared Kernel direferensikan dan dimiliki oleh beberapa konteks terbatas. Setiap tim bebas untuk memodifikasi *library* yang mendefinisikan *integration contract*. Namun, perubahan pada kontrak dapat merusak pembangunan tim lain, oleh karena itu, seperti dalam kasus relasi *partnership*, pola ini membutuhkan komitmen dan sinkronisasi tingkat tinggi antar tim [10].

3. Customer-Supplier

Customer-Supplier menempatkan dua *bounded context* kedalam *upstream* dan *downstream*, di mana *upstream* adalah pemasok/*supplier*, harus mencoba dan memenuhi harapan *customer*/pelanggan (*downstream*). Tetapi keputusan akhir tentang apa yang didapat *customer* berasal dari *supplier*. Untuk lebih detail dapat dilihat pada gambar 26

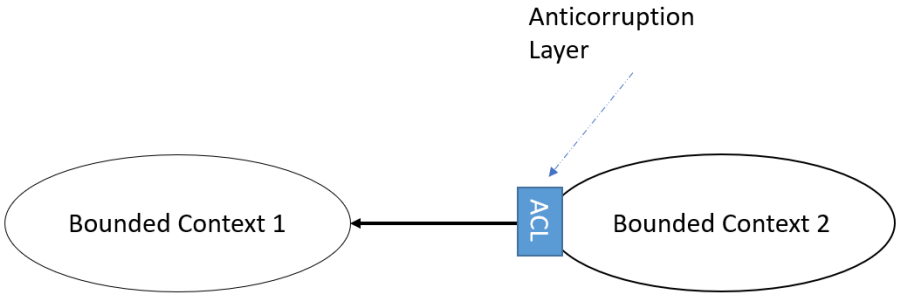


Gambar 28 *Context Map Customer Supplier*

Saat dua buah tim dengan tujuan *context* yang berbeda saling berhubungan akan besar kemungkinannya bagi tim yang menjadi *upstream* untuk melakukan keputusan yang merugikan bagi tim yang menjadi *downstream*. Kerugian ini dapat mengakibatkan pekerjaan dari tim sebagai downstream menjadi tidak sesuai. Untuk mengatasi masalah tersebut dapat digunakan jenis relasi customer-supplier [2]

4. Anticorruption layer

Anticorruption layer merupakan hubungan *upstream / downstream* pada tingkat yang lebih rendah, dimana *bounded context downstream* mengimplementasi layer antar dirinya dan *upstream* tersebut, artinya *bounded context downstream* tidak akan menyesuaikan permintaan yang dilakukan oleh *bounded context upstream*, yang dapat dilakukannya adalah menerjemahkan model *bounded context upstream* ke dalam model yang disesuaikan dengan kebutuhannya sendiri melalui lapisan yang disebut lapisan ACL (*Anticorruption Layer*). [10]



Gambar 29 *Context Map Anticorruption*

5. Open-host service

Pada pola ini, kekuasaan penuh dimiliki oleh pihak *customer*, artinya *supplier* akan melindungi *costumer* nya dan memberikan layanan terbaik. Misalnya, untuk

menyesuaikan dengan API *external*, *downstream* akan memiliki keyakinan dalam integrasi dengan memahami dokumentasi yang disediakan oleh API *external* tersebut. [10]

6. Separate ways

Pada jenis ini, komunikasi antar *bounded context* sama sekali tidak berelasi artinya tidak ada kolaborasi antar tim yang dilakukan. Pola ini dapat muncul karena alasan yang berbeda, dalam kasus di mana tim tidak mau atau tidak dapat berkolaborasi. [10]

2.14 QoS in Web Service

Parameter pengukuran yang digunakan pada penelitian ini berdasarkan parameter *performance* yang terdapat pada QoS (*Quality of Service*), berikut ini merupakan parameter ukur yang digunakan pada penilitian ini.

2.11.1. Latency

Pengukuran waktu latency merupakan waktu delay dimana nilai tersebut didapatkan dari waktu *client* meminta suatu data ke server ditambah dengan waktu server memberikan kembali hasil data ke *client*. Berikut ini merupakan rumus yang digunakan untuk menentukan nilai *latency*:

response time – request time

Gambar 210 *Latency*

Pengukuran latency akan dilakukan dengan cara memanfaatkan *tools* apache jmeter, dimana pada *tools* tersebut dapat menampilkan nilai *latency* pada suatu *service* yang telah ditentukan sebelumnya.

2.11.2. Response time

Pada pengukuran ini merupakan perhitungan waktu yang diperlukan untuk menyelesaikan request. Waktu yang dihitung adalah waktu dimana server melakukan execution time dan memberikan response kepada client. Untuk mendapatkan nilai response time dapat menggunakan rumus sebagai berikut:

execution time + data transferred time

Gambar 211 *Response Time*

Data transfer time merupakan waktu dimana server mengirimkan data json ke client. Pengukuran ini juga dapat menggunakan logging atau pencatatan, waktu yang dicatat adalah waktu yang diperlukan untuk business logic yang didapat dari waktu ketika server mendapatkan request hingga business logic selesai dilakukan, dan waktu yang diperlukan untuk mengirim data ke client yang didapat dari waktu business logic selesai dilakukan hingga data tersebut sampai di client.

2.11.3. Throughput

Pengukuran throughput dapat dilakukan dengan cara melakukan lebih dari satu request dalam kurun waktu tertentu dan dalam dilakukan secara bersamaan. Untuk mendapatkan nilai maksimal throughput dapat menggunakan rumus sebagai berikut:

$$\frac{\text{jumlah maksimal request ditangani}}{\text{satuan waktu}}$$

Gambar 212 *Throughput*

logging atau pencatatan jumlah request yang di-handle oleh server dalam kurun waktu tertentu.

2.11.4. Reliability

Reliability merupakan tolak ukur untuk tingkat maintain *Web Service*. Mengukur tingkat kegagalan pada sistem setiap hari, minggu, bulan, atau tahun tergantung dari seberapa besar kemampuan pengembang untuk melakukan maintain. Karena tidak ada jaminan semua pesan atau response akan sampai ke tujuan.

2.15 Performance Test

Performance Testing merupakan suatu proses untuk menguji batas ketahanan dan kestabilan sistem termasuk modul aplikasi dan infrastrukturnya, serta menguji bagaimana sistem/aplikasi dapat bekerja kembali saat setelah terjadi *down* atau

kegagalah dalam suatu *request* dalam kondisi *load* yang tinggi. Sebelum melakukan pengujian, akan disusun terlebih dahulu skenario testing. Skenario akan disusun berdasarkan kebutuhan dari bisnis sekarang yang akan digambarkan dalam kondisi *high peak,* dimana ini bertujuan untuk mengetahui kesiapan sistem/aplikasi dalam menghadapi kondisi tersebut. Adapun fokus yang terdapat pada *performance test* ini adalah sebagai berikut.

a. Speed : menentukan seberapa cepat respon suatu aplikasi.

b. Scalability : menentukan beban pengguna maksimum suatu aplikasi.

c. Stability : menentukan apakah suatu aplikasi dapat berjalan pada beban yang bervariasi.

d. Reability : kemampuan suatu aplikasi untuk menghandle suatu beban dengan beban load yang berat.

Penelitian ini akan menggunakan jenis testing yaitu stress testing dimana stress testing digunakan untuk mengetahui stabilitas dari service yang ada, service akan diuji dengan mengirimkan banyak permintaan sehingga menimbulkan traffic yang tinggi dan beban bagi aplikasi. Pengujian ini dimaksudkan untuk mengetahui ketahanan dari service dalam menangani request dan bagaimana error yang terjadi akan ditangani ketika server dalam kondisi beban yang tinggi. Berikut ini beberapa tools yang dapat digunakan dalam pengujian:

a. Jmeter

Jmeter software berbasis open source yang dikembangkan oleh Apache Software Foundation, Jmeter merupakan perangkat lunak yang sepenuhnya dibangun menggunakan Java application. Pada awalnya aplikasi dibangun hanya untuk menguji fungsional pada suatu aplikasi dan mengukur seberapa besar performansinya. Seiring berjalannya waktu Jmeter saat ini telah mendukung lebih banyak tipe pengujian seperti Load Testing, Distributed Testing, Database Application Testing, dan sebagainya.

b. LoadUI

Sedikit berbeda dengan Jmeter, LoadUI memang khusus dibangun untuk menganalisis performansi dari suatu aplikasi web. Tools ini dapat digunakan untuk menguji kecepatan dan tingkat skalabilitas dari API, menampilkan hasil analisis sebelum produk dapat dikeluarkan ke publik.

c. NeoLoad

Tidak jauh berbeda dengan LoadUI, NeoLoad khusus digunakan untuk menganalisis aplikasi web. Kelebihan dari NeoLoad terdapat pada sisi fitur yang disediakan dikarenakan fokus terhadap performansi suatu aplikasi web menjadikan tools ini memang dirancang untuk hal tersebut.