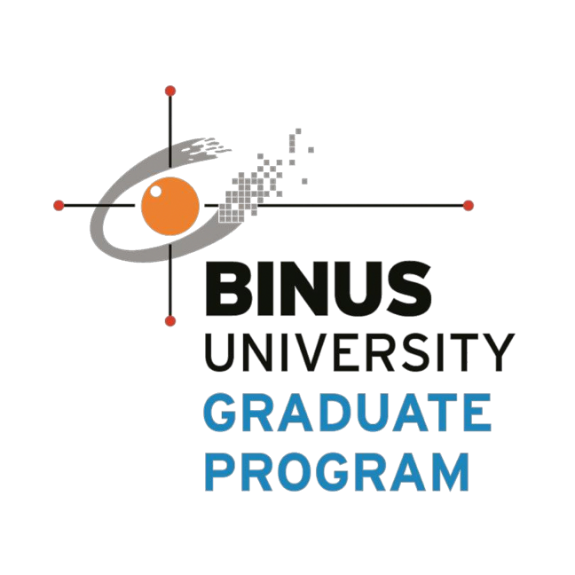
**PROPOSAL**

**OPTIMALISASI ARSITEKTUR REST PADA EKOSISTEM KOMPUTASI AWAN UNTUK MENDUKUNG PLATFORM E-LEARNING**



**RESEARCH**

*FAISAL SSI*

*2001847780*

**Program Pascasarjana Ilmu Komputer**

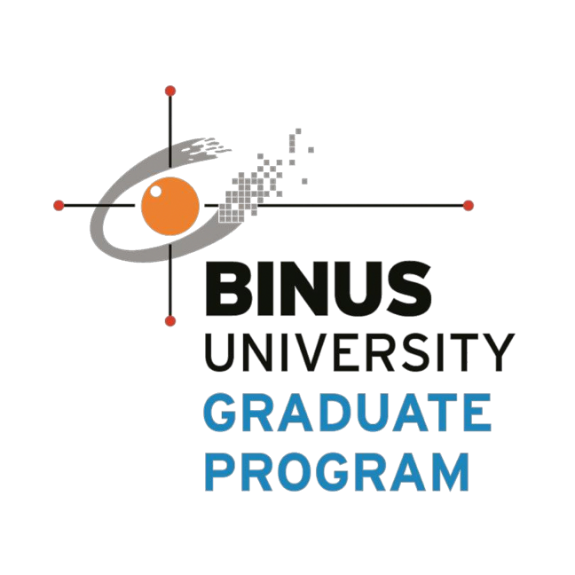
**PROGRAM STUDI TEKNIK INFORMATIKA JENJANG S2**

**UNIVERSITAS BINA NUSANTARA**

**JAKARTA**

**2019**

**PROPOSAL**

**OPTIMALISASI ARSITEKTUR REST PADA EKOSISTEM KOMPUTASI AWAN UNTUK MENDUKUNG PLATFORM E-LEARNING**

**RESEARCH**

*FAISAL SSI*

*2001847780*

Pembimbing:

**I Gede Putra Kusuma Negara, B.Eng., PhD**

03-05-2019

# Daftar Isi

[Daftar Isi iii](#_Toc7635573)

[Daftar Gambar v](#_Toc7635574)

[Daftar Tabel vii](#_Toc7635575)

[BAB I. PENDAHULUAN 1](#_Toc7635576)

[1.1 Latar Belakang 1](#_Toc7635577)

[1.2 Rumusan Masalah 6](#_Toc7635578)

[1.3 Tujuan dan Manfaat 6](#_Toc7635579)

[1.4 Ruang Lingkup 7](#_Toc7635580)

[BAB II. LANDASAN TEORI 8](#_Toc7635581)

[2.1 E-Learning 8](#_Toc7635582)

[2.2 Cloud Computing 9](#_Toc7635583)

[2.3 Service Oriented Architecture (SOA) 11](#_Toc7635584)

[2.3.1 Web Services 13](#_Toc7635585)

[2.4 SOA Design Pattern 19](#_Toc7635586)

[2.5 Software Quality Attributes 23](#_Toc7635587)

[2.5.1 Performance 24](#_Toc7635588)

[2.5.2 Scalability 25](#_Toc7635589)

[2.6 Architecure Tradeoff Analysis Method (ATAM) 27](#_Toc7635590)

[2.7 Rangkaian Analisis Tinjauan Pustaka (Jurnal Terkait) 31](#_Toc7635591)

[2.7.1 E-Learning dan Komputasi Awan 31](#_Toc7635592)

[2.7.2 E-Learning dan SOA 38](#_Toc7635593)

[2.7.3 Representational State Transfer (REST) 47](#_Toc7635594)

[2.7.4 Architecure Tradeoff Analysis Method (ATAM) 49](#_Toc7635595)

[BAB III. METODOLOGI 56](#_Toc7635596)

[3.1 Kerangka Pikir 56](#_Toc7635597)

[3.2 Rancangan Solusi Arsitektur 58](#_Toc7635598)

[3.2.1 REST Constraint 62](#_Toc7635599)

[3.2.2 SOA Design Pattern 64](#_Toc7635600)

[3.3 Rencana Evaluasi 66](#_Toc7635601)

[3.3.1 Skenario dan Pertanyaan Desain 66](#_Toc7635602)

[3.3.2 Skenario Kongkret 69](#_Toc7635603)

[3.3.3 Output ATAM: Analisis Arsitektur 70](#_Toc7635604)

[3.4 Rencana Kegiatan 76](#_Toc7635605)

[DAFTAR PUSTAKA 77](#_Toc7635606)

# Daftar Gambar

[Gambar II.1 Service dapat mengenkapsulasi sejumlah logic yang berbeda 12](#_Toc7635607)

[Gambar II.2 Gambaran proses interaksi client-server 14](#_Toc7635608)

[Gambar II.3 Sistem berlapis dalam gaya arsitektur REST 18](#_Toc7635609)

[Gambar II.4 Konsumen mengakses layanan kontrak yang berbeda 20](#_Toc7635610)

[Gambar II.5 Layanan kontrak yang seragam sebagai solusi kompleksitas 20](#_Toc7635611)

[Gambar II.6 Design Pattern berupa Endpoint Redirection 22](#_Toc7635612)

[Gambar II.7 HTTP *Endpoint Redirection* 23](https://binusianorg-my.sharepoint.com/personal/faisal_ssi_binus_ac_id/Documents/01%20-%20Thesis%20Fight/2001847780%20Proposal%20Tesis%20Faisal%20SSi%20v4%20rev1.docx#_Toc7635613)

[Gambar II.8 Model *Mobile Learning* dengan *cloud computing* 31](#_Toc7635614)

[Gambar II.9 HPC untuk ekosistem Mobile E-Learning 33](#_Toc7635615)

[Gambar II.10 Struktur system mobile E-Learning berbasiskan cloud computing oleh Chen, et.al. 35](#_Toc7635616)

[Gambar II.11 Gambaran model LMS yang diajukan oleh Chen, et.al 38](#_Toc7635617)

[Gambar II.12 LMS Moodle yang terhubung dengan software tools untuk praktikum ilmu komputer menggunakan prinsip SOA 39](#_Toc7635618)

[Gambar II.13 Komunikasi antara Tool Consumer dan Tool Provider via REST web services 41](#_Toc7635619)

[Gambar II.14 Penggunaan Service API Learning Interoperabilty Tool dengan Roslyn Compiler Service 41](#_Toc7635620)

[Gambar II.15 Arsitekur Moodle *Integration* dan *Service Development* *dengan Learning Interoperability Tool* 42](#_Toc7635621)

[Gambar II.16 Arsitektur *Virtual Campus* dengan *canonical interface* untuk menjembatani LMS dengan arsitektur yang berbeda 43](#_Toc7635622)

[Gambar II.17 Tiga roles dalam Service-Oriented Architecture (SOA ) dalam pengembangan asritektur aplikasi mobile untuk pendidikan 44](#_Toc7635623)

[Gambar II.18 Gambaran arsitektur Mobile Learning dengan JSON 46](#_Toc7635624)

[Gambar III.1 Kerangka pikir penelitian 57](#_Toc7635625)

[Gambar III.2 Rancangan desain Arsitektur E-Learning Multitenant 58](#_Toc7635626)

[Gambar III.3 Gambaran strategi deployment sistem E-Learning di Cloud 61](#_Toc7635627)

[Gambar III.4 SOA Design Pattern Core Controller LMS berupa Reusable Contract 65](#_Toc7635628)

[Gambar III.5 Tampilan proses stress test pada software Siege 74](#_Toc7635629)

# Daftar Tabel

[Tabel II.1 Contoh Uniform Contract CRUD 21](#_Toc7635630)

[Tabel II.2 Langkah-langkah ATAM 27](#_Toc7635631)

[Tabel II.3 Skenario umum yang dikonversi menjadi skenario pengujian 29](#_Toc7635632)

[Tabel II.4 Hasil tinjauan pustaka 51](#_Toc7635633)

[Tabel III.1Sintaks umum URI 63](#_Toc7635634)

[Tabel III.2 Penggunaan pola URI pada core controller 64](#_Toc7635635)

[Tabel III.3 Skenario Umum Atribut Kualitas 67](#_Toc7635636)

[Tabel III.4 Skenario Umum REST dalam availability 68](#_Toc7635637)

[Tabel III.5 Gambaran Skenario Kongret yang hendak dicapai 69](#_Toc7635638)

[Tabel III.6 Hasil analitik dari contoh skenario kongkret secara umum 71](#_Toc7635639)

[Tabel III.7 Contoh analitik dari contoh skenario kongkret SC2 72](#_Toc7635640)

[Tabel III.8 Contoh hasil analitik skenario kongkret SC2 dan P1 72](#_Toc7635641)

[Tabel III.9 Tabel evaluasi *stress test* aspek scalability cluster E-Learning 74](#_Toc7635642)

[Tabel III.10 Rencana tabel evaluasi untuk performance multitenant 75](#_Toc7635643)

[Tabel III.11 Rencana Kegiatan Penelitian 76](#_Toc7635644)

# PENDAHULUAN

## Latar Belakang

Penetrasi jumlah pengguna internet di Indonesia sangat berkembang pesat. Berdasarkan statististik jumlah pengguna internet di Indonesia adalah ke-lima terbesar di dunia dengan jumlah 143.26 juta atau sekitar 54% dari total populasi penduduk (“Internet World Stats,” 2019). Dengan internet maka era baru kebutuhan informasi menjadi lebih luas dan kebutuhan belajar kita meningkat, sehingga jenis pembelajaran jarak jauh seperti *Digital Learning* atau E*-Learning* yang bisa diakses melalui perangkat bergerak seperti laptop, tablet dan *handphone* sangat dibutuhkan untuk mencapai pembelajaran kapan saja dan di mana saja.

Menurut (Nasir, 2018), peningkatan akses pendidikan di Indonesia harus mengikuti perkembangan zaman yang saat ini sudah memasuki era revolusi industri 4.0. Pendidikan jarak jauh atau *online education* akan memiliki peran strategis untuk perluasan dan pemerataan pendidikan yang diharapkan seluruh masyarakat di pelosok wilayah Indonesia dapat memiliki kualitas pendidikan yang sama. Proses pembelajaran ini juga akan menjadi semakin fleksibel, terdistribusi dengan baik, semakin tepat waktu, dan sesuai permintaan.

Solusi *cloud computing* atau komputasi awan dalam dunia pendidikan menyediakan ide pengembangan baru yakni berupa *mobile learning*. Dengan sistem *cloud computing*, banyak sekali tugas komputasi dan penyimpanan yang dilakukan oleh mesin server yang ada di *cloud*. Dengan demikian tidak ada lagi keterbatasan baik dari fungsi software, daya komputasi dan kapasitas penyimpanan karena semuanya ditangani oleh mesin yang ada di *cloud.*

Mengacu pada perkembangan di atas, maka kebutuhan akan solusi teknologi informasi di bidang pendidikan untuk mendukung E-Learning juga meningkat. Lembaga pendidikan baik itu universitas, sekolah, lembaga kursus bahkan perusahaan beberapa telah mengimplementasikan sistem *digital learning* melalui *Learning Management System* (LMS). Salah satu platfom LMS cukup populer dipakai dan bersifat *Open Source* adalah Moodle *(Modular Object-Oriented Dynamic Learning Environment)*. Secara statistik Moodle telah diimplementasikan pada 92.971 situs terdaftar yang tersebar di 229 negara dimana terdiri dari 17,982,765 materi pelajaran dengan total user sebanyak 148,620,029 (“Moodle Statistics,” 2019).

Berdasarkan studi (Rodriguez & Lazano, 2011) telah meneliti terkait hubungan sebab akibat antara persepsi tentang kegunaan, persepsi kemudahan penggunaan dan perilaku penggunaan aktual pada teknologi Moodle melalui *Technology Acceptance Model* (TAM). Hasilnya mereka yang menggunakan *platform* Moodle tampaknya mendapatkan nilai yang lebih baik daripada mereka yang jarang atau tidak pernah menggunakannya.

Peneliti yang lain yakni (Deepak, 2017) telah melakuan studi terkait evaluasi fitur-fitur Moodle dengan metode *Qualitative Weight and Sum* (QWS) pada sebuah universitas di Finlandia yang digunakan pada 1000 kelas perkuliahan memberikan kesimpulan bahwa fitur *assignment, lesson*, forum, *quiz* dan *workshop* adalah fitur-fitur yang sangat penting digunakan dalam proses perkuliahan.

Kajian lainnya dilakukan oleh (Kitanov & Davcev, 2012) terkait penerapan Moodle yang lebih teknis pada *private cloud* universitas berupa implementasi infrastruktur *mobile learning* berbasis *cloud computing* dengan *High Performance Computing.* Selain itu (Al-Khanjari, Al-Roshdi, & Kraiem, 2014) dalam jurnal penelitiannya yang berjudul pengembangan desain arsitektur lanjut untuk *resource virtual* pada pembelajaran ilmu komputer telah mengajukan pengembangan desain arsitektur LMS Moodle dengan menggunakan prinsip *Service Oriented Architecture* (SOA). Protokol *web service* berbasis SOA yakni *Representational State Transfer (REST)* digunakan untuk konektivitas dengan aplikasi eksternal seperti *virtual lab* atau *virtual compiler*. Dengan penggunaan SOA ini dapat mengatasi keterbatasan LMS dalam mendukung kegiatan praktikum bidang ilmu komputer.

SOA merupakan konsep arsitektur dalam desain software yang menekankan penggunaaan kombinasi layanan yang *loosely coupled* untuk kebutuhan bisnis (Graham, 2008). *Loosely coupled* artinya sebuah service dapat dipanggil oleh program atau service lainnya tanpa program pemanggil tersebut perlu memperhatikan di mana lokasi service yang dipanggil berada dan platform atau teknologi apa yang digunakan oleh service tersebut. Kecenderungan saat ini dimana sistem informasi di bidang pendidikan seperti LMS membutuhkan banyak integrasi layanan dan *service* dapat digunakan kembali (*reuse*) maka prinsip SOA sangat penting untuk diterapkan.

Saat ini teknologi web yang cukup berkembang pesat dimana web service digunakan sebagai basis teknologi untuk layanan. *Representational State Transfer* (REST) merupakan sebuah desain SOA untuk hypermedia atau sistem terdistribusi (Thomas Fielding, 2000). REST adalah gaya arsitektur untuk membuat SOA berbasis web dan sering disebut RESTful web service. Ini telah menjadi *industry standard* pada arsitektur software barbasis SOA berskala besar (Fielding et al., 2017).

Memilih arsitektur yang tepat untuk sistem software berskala besar sangat penting untuk mencapai persyaratan fungsional, atribut kualitas, dan kebutuhan organisasi. Atribut kualitas perangkat lunak seperti *realibility, performance* dan *modifiability* sangat bergantung pada desain arsitektur perangkat lunak secara keseluruhan dibandingkan kebergantungan pada praktek pemilihan bahasa pemrograman, detail desain, algoritma, pengujian dan sebagainya (Kazman et al., 1998). Ini tidak berarti bahwa bahasa pemrograman atau algoritma tidak penting, tetapi mendefinisikan cetak biru struktur perangkat lunak sangat penting untuk mencapai atribut kualitas. Pada dasarnya, tidak ada arsitektur perangkat lunak terbaik atau terburuk. Semua desain arsitektur harus memiliki *tradeoff* atau kompromi dalam atribut kualitas. Sebagai contoh, jika kita meningkatkan keamanan suatu perangkat lunak harus ada *tradeoff* dalam *performance*. Untuk menghadapi masalah ini, terdapat metode yang disebut *Architecture Tradeoff Analysis Method* (ATAM). Metode ini memberikan evaluasi dalam atribut kualitas perangkat lunak seperti *realibility, performance, security, modifialibility* dan sebagainya. Selain itu, metode ini juga menguji dan mengevaluasi pertukaran teknik dan risiko antara keputusan desain arsitektur (Barbacci, et al., 2003).

Penggunaan *Representational State Transfer* (REST) sebagai gaya arsitektur untuk mengintegrasikan layanan dan aplikasi membawa beberapa manfaat, tetapi juga menimbulkan tantangan dan risiko. (Costa, et al., 2014) telah menyajikan pedoman mengenai penggunaan metode ATAM untuk membantu evaluator mengevaluasi arsitektur berbasis REST agar dapat mengidentifikasi *trade-off* dan risiko untuk mengatasi persyaratan atribut kualitas seperti seperti *security, realibility* dan *performance*.

Terkait kajian-kajian di atas, penulis menemukan celah penelitan terkait bagaimana melakukan optimalisasi pada arsitektur berbasis REST pada ekosistem komputasi awan untuk mendukung platform *E-Learning* khususnya dengan LMS Moodle. Pada penelitian ini penulis akan membuat desain arsitektur LMS Moodle secara *centralized* pada ekosistem komputasi awan dengan menggunakan prinsip SOA. Untuk mendesain arsitektur ini, penelitian ini menggunakan REST *Constraint*, SOA *design pattern* (Erl, et al., 2012) dan REST *design pattern* (Thomas Fielding, 2000). *Architecture Tradde-off Analysis Method* (ATAM) akan digunakan untuk mengevaluasi *trade-off* dan resiko pada pemilihan desain arsitektur (Barbacci, et al., 2003). Proses evaluasi adalah berbasis skenario dan fokus pada *software quality attributes* (Costa, et al., 2014).

Beberapa aspek yang akan menjadi fokus evaluasi itu antara lain :

1. Aspek *performance*, yakni bagaimana menangani banyaknya user yang mengakses LMS secara bersamaan (*concurrent*). Bagaimana menjaga sistem bisa diandalkan dalam menangani aktivitas pembelajaran seperti Quiz, PR, Forum dan lainnya.
2. Aspek *scalability*, yakni bagaimana mempersiapkan segala perubahan terkait ukuran sistem yang ada cloud. Penyediaan infrastruktur cloud yang *over provisioning* (*resource* berlebihan namun utilisasi rendah) yang berefek pada pemborosan biaya, ataupun *under provisioning* (utilisasi tinggi melebihi *resource* yang ditetapkan) yang berefek pada akses user menjadi lambat.

## Rumusan Masalah

Mengacu pada temuan masalah pada latar belakang di atas maka berikut adalah rumusan masalah dalam penelitian ini adalah:

* Bagaimana melakukan evaluasi dan optimalisasi terkait arsitektur REST LMS Moodle pada ekosistem komputasi awan sehingga menghasilkan nilai *trade-off* yang optimal baik dari sisi *performance dan scalability*?
* Bagaimana gambaran desain model arsitektur REST LMS Moodle yang terbaik setelah dilakukan evaluasi dan optimalisasi.

## Tujuan dan Manfaat

Tujuan penelitian ini adalah:

* Melakukan evaluasi terkait atribut kualitas di aspek *performace* dan *scalability*  pada LMS Moodle di ekosistem komputasi awan dengan dengan menggunakan metode ATAM (*Architecture Trade-Off Analysis Method*)
* Melakukan optimalisasi arsitektur *backend* LMSMoodle pada ekosistem komputasi awan mengacu pada hasil evaluasi dengan metode ATAM.

Manfaat:

* Memberikan evaluasi secara menyeluruh terkait desain arsitektur LMS Moodle berbasis komputasi awan melalui pendekatan ATAM
* Menjawab tantangan terkait aspek *performace dan scalability* pada sistem LMS Moodle sehingga bisa dijadikan referensi untuk mengimplemtasikan sistem *E-Learning* yang bersifat *multi-tenant* untuk melayani *stakeholder* terkait seperti universitas, sekolah, lembaga kursus dan perusahaan.

## Ruang Lingkup

Ruang lingkup penelitian ini adalah:

* Menggunakan sistem *E-Learning* Moodle versi terupdate sebagai objek utama penelitian dan di*host* pada layanan cloud *Google Cloud Platform* (GCP).
* Melakukan pendekatan ATAM dalam mengevaluasi arsitektur Sistem *E-Learning* pada ekosistem komputasi awan. Proses evaluasi arsitektur fokus pada atribut kualitas *performance* dan *scalability.*
* Menggunakan beberapa tools untuk simulasi pengukuran performa sistem *E-Learning* yakni Apache Jmeter dan Siege.

# LANDASAN TEORI

## E-Learning

Menurut (Darin E. Hartley, 2001) *E-Learning* merupakan suatu jenis belajar mengajar yang memungkinkan tersampaikannya bahan ajar ke siswa dengan menggunakan media Internet, Intranet atau media jaringan komputer lain. Sistem pembelajaran yang digunakan sebagai sarana untuk proses belajar mengajar yang dilaksanakan tanpa harus bertatap muka secara langsung antara guru dengan siswa.

*E-Learning* berperan penting dalam proses pembelajaran karena memfasilitasi dalam akses dan distribusi pengetahuan dengan menggunakan perangkat elektronik (Zhang D, 2004). Beberapa universitas dan lembaga pendidikan di seluruh dunia mengadaptasi lingkungan *E-Learning* selain sebagai alat pembelajaran utama juga untuk melengkapi metode pengajaran di kelas konvensional.

Sistem *E-Learning* umumnya diimplementasikan dalam bentuk aplikasi *Learning Management System* (LMS). LMS adalah aplikasi perangkat lunak untuk administrasi, dokumentasi, *tracking, reporting*, dan penyampaian materi pendidikan, program pelatihan, atau program pembelajaran dan pengembangan (Davis, Carmean, & Wagner, 2009). Umumnya berupa aplikasi berbasis web yang digunakan untuk merencanakan, mengimplementasikan, dan menilai proses pembelajaran tertentu. LMS menyediakan layanan bagi para instruktur cara untuk membuat dan menyampaikan konten pembelajaran, memantau partisipasi siswa, dan menilai kinerja siswa. LMS juga dapat memberi siswa kemampuan untuk menggunakan fitur interaktif seperti forum diskusi dan video conference.

Moodle (*Modular Object-Oriented Dynamic Learning Environment*) adalah salah satu platform LMS *open-source* paling populer dengan sejumlah besar implementasi di sejumlah institusi pendidikan di dunia. LMS cukup populer dipakai dan bersifat Open Source. Moodle ditulis dalam bahasa pemrograman PHP dan didistribusikan di bawah Lisensi Publik Umum GNU. Moodle diciptakan oleh Martin Dougiamas untuk membantu para pendidik untuk membuat pembelajaran *online* dengan fokus pada interaksi dan konstruksi konten yang kolaboratif dengan evolusi yang berkelanjutan (Dvorak, 2011).

Universitas, sekolah, bisnis, dan instruktur individual menggunakannya untuk memberikan pembelajaran online dan untuk mendukung pembelajaran tatap muka tradisional. Desain modular Moodle memudahkan untuk membuat materi pembelajaran baru dengan menambahkan konten yang akan melibatkan peserta didik. Moodle menyediakan seperangkat tools berbasis web yang powerfull untuk berbagai aktivitas dan resource, seperti forum, *messaging*, tes, *quiz*, *assignment*, *feedback, wiki, blog*, *glosarium*, dan sebagainya (Nash & Rice, 2018).

Secara statistik Moodle telah diimplementasikan pada 92.971 situs terdaftar yang tersebar di 229 negara dimana terdiri dari 17,982,765 materi pelajaran dengan total user sebanyak 148,620,029 (“Moodle Statistics,” 2019).

## Cloud Computing

*Cloud computing* atau komputasi awan adalah proses komputasi yang mengacu pada *delivery* sumber daya IT yang tervirtualisasi dan terdistribusi melalui internet dimana proses komputasi dikirimkan sebagai *service* yang bersifat *on demand* dan *user* atau perusahaaan dapat berlangganan dengan mem-bayar biaya s*ubscription* tertentu (Debashis De, 2016).

*Cloud Computing* adalah pengembangan dari *distributed computing*, *parallel computing* dan *Grid Computing*. Ini adalah realisasi komersial dari konsep ilmu komputer tersebut. Prinsip dasar dari *cloud computing* adalah untuk mengintegrasikan sejumlah besar data yang disimpan di komputer yang terdistribusi dan sumber daya prosesor untuk keperluan tugas kolaboratif serta menyediakan layanan (komputasi, penyimpanan, software dan hardware) untuk para pengguna.

Model Layanan *Cloud Computing* terbagi menjadi sebagai berikut:

1. *Infrasturucture as a Service* (IaaS). IaaS adalah sebuah model layanan dimana penyedia *cloud* menyediakan *hardware* (komputer server, penyimpanan data, jaringan, dan sebagainya) untuk pelanggan. Manajemen perangkat keras menjadi tanggung jawab penyedia layanan, dan pelanggan mengontrol *operating system* serta aplikasi yang diinstal ke dalam server.
2. *Platform as a Service* (PaaS). PaaS adalah model layanan yang menyediakan semua hal yang dibutuhkan untuk mengembangkan sebuah aplikasi pada *cloud*. PaaS menawarkan fasilitas untuk mengembangkan, *testing, deployment*, hingga mainte-nance aplikasi tanpa harus membeli infrastruktur dan software environment (Operating System)
3. *Software as a Service* (SaaS). SaaS singkatnya adalah perangkat lunak berbentuk layanan yakni suatu model penyampaian aplikasi perangkat lunak oleh suatu vendor perangkat lunak yang mengembangkan aplikasi web yang diinangi dan dioperasikan (baik secara mandiri maupun melalui pihak ketiga) untuk digunakan oleh pelanggannya melalui Internet. Pelanggan tidak mengeluarkan uang untuk memiliki perangkat lunak tersebut melainkan hanya dengan menggunakannya dengan cara berlangganan.

## Service Oriented Architecture (SOA)

Dalam mendukung topik penelitian ini maka penulis akan mulai dari pembahasan terkait *Service Oriented Architecture* (SOA) yang pendekatannya mendasari suatu arsitektur komputasi modern. Kemudian dihubungkan dengan penerapannnya pada *E-Learning* dan bagaimana implementasinya di ekosistem *cloud computing*.

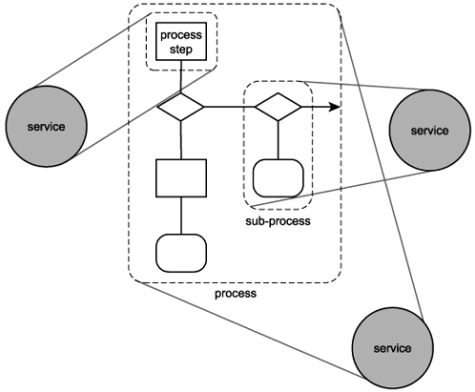
Perkembangan teknologi komputasi yang sangat pesat khususnya terkait *cloud computing* membuat evolusi terkait cakupan SOA. Sebelum membahas tentang SOA, ada baiknya kita memahami terminologi yang mendukung definisi terkait SOA yakni *Service-Oriented Computing.*

Menurut (Erl et al., 2011), s*ervice-oriented computing* adalah istilah umum yang mewakili platform *distributed computing* generasi baru. Hal ini mencakup banyak hal, termasuk paradigma desainnya sendiri dan prinsip-prinsip desain, katalog pola desain, bahasa pola, model arsitektur yang berbeda, dan konsep, teknologi, dan kerangka kerja terkait.

*Service-orientation* adalah paradigma desain yang dimaksudkan untuk pembuatan unit logika solusi yang dibentuk secara individual sehingga mereka dapat secara kolektif dan berulang kali digunakan dalam mendukung realisasi tujuan strategis spesifik dan manfaat yang terkait dengan *service-oriented computing.*

Logika solusi yang dirancang sesuai dengan orientasi layanan dapat dikualifikasikan dengan "*service-oriented*," dan unit logika solusi berorientasi layanan disebut sebagai "*service*." Sebagai paradigma desain untuk komputasi terdistribusi, orientasi layanan dapat dibandingkan dengan desain berorientasi objek.

*Service* sendiri dapat dipandang sebagai enkapsulasi *logic* dari satu atau sekumpulan aktivitas tertentu. Bila dicontohkan dalam sebuah otomasi bisnis, *service* dapat dilihat pada Gambar II.1 berikut.



Gambar II.1 Service dapat mengenkapsulasi sejumlah logic yang berbeda

Otomasi bisnis merupakan sekumpulan aktivitas yang disusun dalam langkah-langkah sebagai implementasi proses bisnis. Lingkup *service* dapat mengenkapsulasi sebuah proses besar atau hanya satu langkah proses kecil.

Dari paparan di atas, maka menurut (Erl et al., 2011) *Service-Oriented Architecture* (SOA) adalah model arsitektur teknologi untuk solusi berorientasi layanan dengan karakteristik yang jelas dalam mendukung realisasi *service orientation* dan tujuan strategis yang terkait dengan *Service-oriented computing*.

Empat karakteristik dasar yang ingin dibangun dalam bentuk SOA adalah:

* *Business-Driven*: Arsitektur teknologi diselaraskan dengan arsitektur bisnis saat ini. Konteks ini kemudian terus dipertahankan sehingga arsitektur teknologi berkembang seiring dengan bisnis dari waktu ke waktu.
* *Vendor-Netral*: Model arsitektur tidak hanya didasarkan pada platform vendor eksklusif, yang memungkinkan teknologi vendor yang berbeda untuk digabungkan atau diganti dari waktu ke waktu untuk memaksimalkan pemenuhan kebutuhan bisnis secara berkelanjutan.
* *Enterprise-Centric*: Ruang lingkup arsitektur mewakili segmen yang berarti dari perusahaan, memungkinkan untuk digunakan kembali dan komposisi layanan dan memungkinkan solusi berorientasi layanan untuk menjangkau silo aplikasi tradisional.
* *Composition-Centric*: Arsitektur secara inheren mendukung mekanisme agregasi layanan berulang, memungkinkannya mengakomodasi perubahan konstan melalui *agile assembly* dari komposisi layanan.

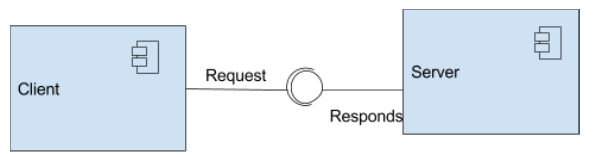
Karakteristik ini secara kolektif menentukan persyaratan mendasar yang harus dipenuhi oleh arsitektur teknologi untuk sepenuhnya mendukung orientasi layanan. Sebagai bentuk arsitektur teknologi, implementasi SOA dapat terdiri dari kombinasi teknologi, produk, API, ekstensi infrastruktur pendukung, dan berbagai bagian lainnya.

### Web Services

SOA dapat diimplementasikan melalui *Web service*. *Web services* adalah sistem pertukaran informasi berbasis XML yang menggunakan internet untuk interaksi antara aplikasi. Teknologi ini merupakan standar yang diadopsi oleh banyak vendor perangkat lunak, karena memiliki standar terbuka. Standar terbuka ini yang memungkinkan aplikasi web service yang diimplementasi oleh vendor berbeda dapat berkomunikasi satu sama lain. Perkembangan *web services* yang begitu cepat menyebabkan lahirnya ekstensi-ekstensi *web services* yang memperluas fungsi web services itu sendiri.

*Web services* generasi pertama adalah pondasi dari teknologi *web services* ini sendiri. Implementasi dari teknologi ini, yang berupa aplikasi *web services*, saling berinteraksi satu sama lain dengan menggunakan dokumen berformat XML dan protokol pengiriman pesan SOAP (*Simple Object Access Protocol)* melalui HTTP. Format XML, SOAP, dan HTTP ini juga merupakan standar terbuka yang dapat diadopsi.

Teknologi web service lain selain SOAP dan cukup populer saat ini adalah REST. REST atau singkatan dari *Respresentational State Transfer* merupakan salah satu pendekatan dalam mengimplementasikan SOA. REST adalah konsep arsitektur (*architectural style*) yang dibuat oleh Roy Fielding pada tahun 2000 dan menjadi populer karena kesederhanaan dan model pengembangannya yang ringan (Ejsmont, 2015). REST merupakan sebuah arsitektur client-server berdasarkan pada sebuah desain respon pemintaan. REST digunakan pada aplikasi terdistribusi dengan menggunakan HTTP untuk mengirim pesan agar bisa berkomunikasi diantara komponen-komponen.



Gambar II.2 Gambaran proses interaksi client-server

Pihak *client* mengirimkan permintaaan dan pihak *server* memberikan respon, namun di REST proses komunikasinya adalah berdasarkan *resource* (*resource-based*). Ini artinya bahwa pesan yang dikirim adalah sebagai representasi dari *resources*. *Resources* dapat berupa segala bentuk informasi yang berdiri sendiri, yakni dapat berupa dokumen, gambar, representasi objek dan lain sebagainya.

Tidak seperti SOAP, REST adalah layanan yang bersifat *resource centric*. Perbedaan antara SOAP dan REST dijelaskan oleh Khan dan Abbasi (Khan & Abbasi, 2015) menjadi beberapa kriteria yang berbeda sebagai berikut:

* REST mengakses *resources* sementara itu SOAP mengakses operasi.
* SOAP memanggil layanan dengan memanggil metode RPC yang mengekspos logika aplikasi sebagai layanan. Sedangkan REST hanya cukup memanggil layanan melalui jalur URL yang mengekspos *Single Public API*.
* REST mengakses resource dengan *single interface* yang konsisten, SOAP menerapkan logika bisnis dengan *interface* yang berbeda.
* SOAP tidak mengembalikan hasil yang dapat dibaca manusia, sementara hasil REST dapat dibaca dengan format XML atau JSON.
* REST memiliki *performance* dan *scalability* yang lebih baik karena dapat di-cache. SOAP tidak bisa di-*cache.*

Perbedaan lainnya yaitu SOAP menggunakan *Web Services Description Language* (WSDL) untuk komunikasi antara *consumer* dan *provider*, sedangkan REST hanya menggunakan XML atau JSON untuk mengirim dan menerima data. Selain itu SOAP tidak hanya melalui HTTP, tetapi juga menggunakan protokol lain seperti SMTP, FTP, dan lain-lain. Sedangkan REST hanya melalui HTTP.

(Thomas Fielding, 2000) telah menggambarkan enam batasan / *constraint* REST, masing-masing mencerminkan satu atau banyak atribut kualitas perangkat lunak. Ini memiliki simbol dan dapat digambarkan sebagai berikut:

(1)

* *Client-Server* (C-S)

Arsitektur *client-server* menerapkan pemisahan masalah melalui peran *client* dan *server* dengan tanggung jawab spesifik yang saling berinteraksi. *Server* menyediakan layanan kepada *client*, dan *client* memberikan pengguna antarmuka untuk mengakses layanan. Arsitektur *client-server* memungkinkan aplikasi REST menjadi sangat *scalable* dan memungkinkan pengembangan *client* dan *server* terjadi secara independen. *Client* menyediakan pengguna dengan antarmuka yang sederhana dan cepat tanpa mempengaruhi *server*, sementara *server* dapat memanipulasi *set* data yang lebih besar karena dibebaskan dari keharusan melaksanakan tanggung jawab *client*.

* *Stateless* (S)

Dalam arsitektur REST, interaksi harus *stateless*. Ini berarti bahwa *server* tidak menyimpan informasi tentang keadaan *client* saat ini atau permintaan sebelumnya yang dibuat oleh client. Server hanya memperhatikan bahwa *client* ada saat permintaan dibuat. Semua informasi yang diperlukan untuk *server* untuk memahami dan menanggapi permintaan datang melalui permintaan, dan permintaan terkandung satu sama lain. Ini meningkatkan *performance* layanan web, karena *server* tidak harus mengingat status *client* saat ini dalam sistem. Namun *trade-off* nya adalah bahwa ini memberlakukan batasan signifikan pada cara *client* dan *server* berkomunikasi. Setiap kali *client* mengirim permintaan ke *server*, ia harus menyediakan dan menyimpan informasi tentang keadaan saat ini.

* *Cache* ($)

Batasan ini berarti bahwa client dapat menyimpan salinan lokal (*cache*) dari respons server untuk digunakan untuk permintaan nanti, tergantung pada informasi apa yang ditambahkan server ke respons untuk memberi label sebagai *cacheable* atau *non-cacheable*. Ini dapat membantu meningkatkan *performance* dengan mengurangi jumlah permintaan untuk *resource* yang sama, dan membantu memastikan bahwa *client* tidak menyimpan data yang berlebihan atau tidak berguna.

* *Uniform Interface* (U)

Intinya harus ada *interface* yang seragam untuk komunikasi antara *client* dan *server*. *Constraint* ini memiliki dampak tertentu. Yang pertama adalah bahwa ada metode khusus yang dapat dipahami. REST menggunakan metode HTTP umum yakni GET, PUT, POST, dan DELETE, untuk mengkomunikasikan tindakan berbeda yang ingin dilakukan client pada resources.Yang kedua adalah bahwa sumber daya harus diidentifikasi dalam permintaan dengan *Uniform Resource Identifier* (URI) tertentu yang akan merepresentasikan *uniform resource*. Respon memiliki tajuk spesifik, dan *resource* ditulis dalam tiga cara spesifik: XML, JSON, atau teks sederhana.

* *Layered System* (L)

REST adalah system berlapis. REST dapat terdiri dari beberapa lapisan arsitektur *software* atau *hardware* yang dapat dipanggil oleh *client* dan *server*. Lapisan-lapisan ini dapat digunakan untuk meningkatkan performance, menerjemahkan pesan, dan mengelola lalu lintas.



Gambar II.3 Sistem berlapis dalam gaya arsitektur REST

Ini membantu meningkatkan penggunaan kembali (*reusability*) REST *Web service* karena lapisan dapat ditambahkan dan dihapus berdasarkan pada layanan yang dibutuhkan *client*.

* *Code-on-Demand* (CoD)

*Code-on-Demand* (COD) adalah satu-satunya constraint opsional dalam REST. Ini memungkinkan client untuk meningkatkan fleksibilitasnya karena sebenarnya serverlah yang memutuskan bagaimana hal-hal tertentu dilakukan. Misalnya, dengan *Code-On-Demand*, client dapat mengunduh JavaScript, Applet Java atau bahkan aplikasi Flash untuk mengenkripsi komunikasi sehingga server tidak mengetahui adanya rutinitas / kunci enkripsi yang digunakan dalam proses ini. Namun, menggunakan COD mengurangi visibilitas, itulah sebabnya *constraint* ini bersifat opsional. Juga, tidak setiap API membutuhkan fleksibilitas semacam ini. Interoperabilitas juga menurun karena kode harus kompatibel dengan target *consumer*. Keamanan juga menjadi perhatian karena bisa disusupkan dengan kode jahat.

*Constraint* atau batasan ini menjadikan REST gaya arsitektur yang fleksibel dan berkinerja tinggi untuk membangun sistem berorientasi layanan berdasarkan standar web. REST memberikan manfaat termasuk *scalability* tinggi, reusability, dan *loose coupling* yang memungkinkannya memenuhi kebutuhan aplikasi modern dengan jutaan pengguna.

## SOA Design Pattern

*Design pattern* merupakan solusi desain yang sudah terbukti untuk masalah umum dalam desain perangkat lunak. Permasalahan didokumentasikan dalam format standar dan dengan cara yang konsisten (Erl, 2009). *Design pattern* digunakan untuk merancang arsitektur berdasarkan kasus masalah karena menyediakan solusi yang teruji di lapangan sehingga *design pattern* dapat mempercepat proses pengembangan. Dalam konteks SOA, ada banyak kategori *design pattern* yang membahas aspek berbeda dari sistem berbasis SOA diantaranya: *service messaging patterns, service implementation patterns, service security patterns, composition implementation patterns*, dan sebagainya. Erl telah menetapkan delapan puluh lima profil desain pattern untuk SOA. Juga ada tujuh *design pattern* baru yang terinspirasi REST untuk memecahkan masalah dengan menggunakan kemampuan REST (Erl et al., 2012).

Berikut adalah contoh ilustrasi terkait dengan *design pattern* pada arsitektur REST yakni pattern berjenis *Uniform Contract*. Misal terdapat aplikasi layanan bea cukai yang melayani beda negara dengan tarif pajak yang berbeda.

* *Pattern*: Bagaimanakah konsumen dapat mengambil manfaat dari beberapa titik layanan di masing-masing negara?



Gambar II.4 Konsumen mengakses layanan kontrak yang berbeda

* Permasalahan : Mengakses layanan serupa mengharuskan konsumen untuk mengakses kemampuan yang dinyatakan dalam kontrak khusus layanan. Konsumen harus tetap *up to date* sehubungan dengan banyaknya individual kontrak yang ada.
* Solusi: Standarisasi kontrak yang seragam di seluruh titik akhir layanan alternatif yang disarikan dari kemampuan spesifik masing-masing layanan.



Gambar II.5 Layanan kontrak yang seragam sebagai solusi kompleksitas

* Manfaat: Abstraksi layanan, *loose coupling, reusability, discoverability, composability.*

Contoh *uniform contract* dalam bentuk table CRUD:

Tabel II.1 Contoh Uniform Contract CRUD

|  |  |  |
| --- | --- | --- |
| **CRUD** | **REST** |  |
| CREATE | POST | *Create* / menciptakan sub *resource* |
| READ | GET | *Retrieve* / mengambil keadaan terkini (*current state*) dari *resource* |
| UPDATE | PUT | Menginisialisasi atau memperbarui keadaan dari suatu *resource* pada URI yang diberikan |
| DELETE | DELETE | Menghapus suatu *resource*, setelahnya URI tidak valid lagi |

Secara detil perbedaannya adalah sebagai berikut:

* GET adalah operasi yang bersifat *read-only*. Ini dapat diulangi tanpa mempengaruhi keadaan dari *resource* (*idempotent*) dandapat di *cached.* Dapat membaca berkali-kali dengan hasil yang sama. Sebuah HTTP GET seharusnya tidak pernah digunakan untuk mengubah data.
* POST adalah operasi *read-write* dan dan bisa mengubah keadaan dari *resource* (tidak *idempotent*) dan memancing efek samping pada server.
* PUT adalah operasi yang digunakan untuk memperbarui keadaan resource. Jika operasi PUT dijalankan *N* kali, permintaan pertama akan memperbarui resource, kemudian sisanya yakni permintaan *N-1* hanya akan menimpa keadaan resource yang sama lagi dan lagi dimana secara efektif tidak mengubah apa pun (*idempotent*).
* DELETE adalah operasi untuk menghapus suatu resource. Saat dilakukan *N* permintaan DELETE yang serupa, permintaan pertama akan menghapus *resource* dan responsnya adalah 200 (OK) atau 204 (tidak ada konten). Permintaan *N-1* lainnya akan mengembalikan 404 (tidak ditemukan). Jelas responsnya berbeda dari permintaan pertama, tetapi tidak ada perubahan status untuk *resource* apa pun di sisi server (*idempotent)* karena resource asli sudah dihapus.

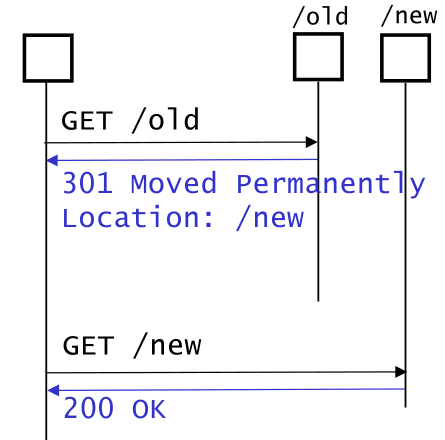
Berikutnya adalah contoh *design pattern* lainnya yakni *Endpoint Redirection*.

* *Pattern*: Bagaimana konsumen dari suatu *service endpoint* beradaptasi ketika *inventory* layanan sedang direstukturisasi?
* Permasalahan: Inventory layanan bisa saja berubah dari waktu ke waktu untuk alasan bisnis dan teknis. Adalah tidak mungkin untuk mengganti semua referensi pada *endpoint* yang lama secara simultan.
* Solusi : Mereferensikan secara otomatis konsumen layanan yang mengakses pengindetifikasi *endpoint* yang sudah kadaluarsa menuju pengindetifikasi yang terkini.



Gambar II.6 Design Pattern berupa Endpoint Redirection

Jika digambarkan terkait pattern endpoint redirection dengan HTTP ada pada gambar sebagai berikut:

Secara native HTTP mendudukung *pattern endpoint redirection* menggunakan kombinasi dari kode status 3xx dan *header* standar:

Gambar II.7 HTTP *Endpoint Redirection*

* + - * 301: Pindah secara permanen
      * 307: *Temporary redirect*
      * Location : /newURI

## Software Quality Attributes

*Quality attributes* atau atribut kualitas adalah persyaratan nonfungsional. Atribut kualitas menangkap persyaratan fungsional yang dicapai oleh aplikasi (Gorton, 2006). Atribut kualitas menetapkan standar minimum untuk aplikasi. Atribut kualitas yang sering kita dengar adalah *scalability, security, performance*, dan *realibility*. Agar benar-benar bermanfaat, atribut kualitas harus ditentukan secara jelas bersama dengan skenario umumnya. Pernyataan yang mengatakan "Aplikasi harus dapat *scalable*", tidak cukup untuk secara jelas menentukan *scalability* seperti apa yang dihadapi aplikasi. Apakah aplikasi harus mampu menangani lebih banyak permintaan? berapa banyak permintaan? Berapa banyak pengguna *concurrent* yang harus mampu dilayani oleh sistem? Atau semua itu diperlukan dalam sistem. Itulah sebabnya atribut kualitas harus dideskripsikan atau diwujudkan sebagai skenario umum seperti:

*Ketika rata-rata penggunaan CPU di atas 90%, sistem harus dapat mengandakan diri sehingga menaikkan spesifikasi server secara keseluruhan hingga rata-rata penggunaan CPU di bawah 70% dengan nol downtime.*

Pernyataan di atas adalah contoh skenario umum yang baik, tepat dan bermakna. Dengan mengikuti skenario itu, seorang arsitek perangkat lunak harus membuat *deployment* atau penyebaran terjadi secara otomatis dan tanpa usaha. Daftar atribut kualitas yang digunakan dalam penelitian ini adalah *performance* dan *scalability.*

### Performance

*Performanc*e adalah jumlah eksekusi yang harus dilakukan suatu aplikasi dalam waktu tertentu. *Performance* dan *scalability* memiliki hubungan yang kuat, *performance* yang buruk dapat mempengaruhi *scalability* dan sebaliknya. Masalah utama p*erformance* adalah peningkatan waktu respon *client*, peningkatan konsumsi memori, peningkatan akses database, dan peningkatan *bandwidth* jaringan. P*erformance* dapat diukur melalui *throughput*, latensi, dan tenggat waktu (*deadline*).

1. *Throughput*

*Throughput* diukur dalam *transaction per second* (tps). Misalnya, sistem *quiz* di LMS harus dapat memproses 100 tps dari jawaban yang dikirim oleh siswa.

2. Waktu Respon (*Lantecy*)

Waktu respons atau *latency* adalah waktu yang diambil oleh aplikasi untuk memproses permintaan. Waktu respon tergantung pada kompleksitas permintaan seperti ukuran data dan logika aplikasi. Waktu respon seringkali berbeda dalam setiap permintaan. Jadi, untuk mengukurnya, waktu respons rata-rata digunakan. Misalnya, 85% dari semua permintaan harus diproses dalam waktu kurang dari 5 detik dan tidak ada permintaan yang diproses lebih dari 18 detik.

3. Batas waktu (*deadlines*)

Batas waktu atau *deadlines* adalah jumlah waktu yang diberikan untuk aplikasi untuk menyelesaikan pekerjaan. *Deadline* sering dikaitkan dengan pekerjaan *batch*. Misalnya, aplikasi harus mengekstrak dan mentransformasikan data ke data *warehouse* selama tengah malam dan harus selesai pada pukul 05.00 pagi.

### Scalability

*Scalability* atau skalabilitas adalah seberapa baik respons aplikasi ketika ukuran beban meningkat (Gorton, 2006). *Scalability* mempengaruhi performance. Misalnya, aplikasi dapat memproses 85% dari semua permintaan dalam 5 detik. Ini terjadi ketika sejumlah beban permintaan di bawah 1000 pengguna per menit. Saat aplikasi diakses oleh 10.000 pengguna per menit, aplikasi tidak dapat menjalankan 85% dari semua permintaan dalam 5 detik. Kondisi ini membutuhkan *scalability* untuk membuat persyaratan *performance* tetap dapat tercapai. Untuk membuat *scalability* menjadi skenario umum, kita perlu memahami seberapa ukuran yang diharapkan untuk dapat menjadi lebih besar.

1. *Request Load*

Aplikasi dirancang untuk server 1500 tps pada beban puncak. Ketika permintaan memuat bertambah sepuluh kali lipat, dapatkah aplikasi tetap mendukung 1500 tps pada beban puncak? *Scalability* muncul untuk menjawab peningkatan permintaan. Solusinya adalah meningkatkan daya komputasi aplikasi atau yang sering disebut *'scale up'*. Juga, ketika permintaan menurun, *scalability* harus *'scale down'* kemampuan sistem untuk membuatnya tetap hemat biaya.

2. *Simultaneous Connections*

*Simultaneous connections* atau koneksi simultan adalah banyaknya permintaan yang terjadi tepat pada waktu saat bersaman. Misalnya, ada batasan jumlah koneksi untuk satu permintaan. Katakanlah batasannya adalah maksimum 1000 pengguna mengakses permintaan yang sama pada saat yang sama. Ketika jumlah pengguna bersamaan di meningkat menjadi 2000, *scalability* harus menjawab masalah ini. Solusinya adalah dengan mereplikasi jumlah *server* untuk menangani banyaknya permintaan.

3. *Data Size*

*Data size* atau ukuran data adalah aspek penting dari performance dan *scalability*. Peningkatan data dapat mengurangi *performance* aplikasi dari waktu ke waktu. Pada tahap awal ukuran data aplikasi tidak akan menjadi masalah. Tetapi ketika sistem tumbuh besar, ukuran data menjadi berlipat. *Scalability* menjawab masalah ini melalui *‘scale up’* padaserver basis data dan memisahkannya menjadi *master* dan *slave.*

4. *Deployment*

*Scale up* dan *scale down* bisa saja terjadi seketika itu juga dan tidak boleh mempengaruhi *experience* dari sisi pengguna. Aplikasi harus dapat disebarkan (*deploy*) dengan mudah dan otomatis sehingga bisa dilakukan *‘scale up’* ataupun *‘scale down’* pada aspek daya komputasi. Strategi *deployment* menjadi bagian terakhir saat menjaga *scalability*.

Dapat difahami bahwa *scalability* adalah proses yang kompleks dan tidak mudah dilakukan. Karena itu masalah *scalability* harus dipertimbangkan ketika merancang arsitektur sistem.

## Architecure Tradeoff Analysis Method (ATAM)

*Architecure Tradeoff Analysis Method* (ATAM) adalah metode untuk menganalisis seberapa baik arsitektur perangkat lunak memenuhi tujuan kualitas tertentu. Juga menyediakan *insigh*t terkait kebergantungan atribut kualitas, yakni bagaimana komponen-komponen di dalam arsitektur saling tarik-ulur *(trade-off*) terhadap satu sama lain (Kazman et al., 1998). Metode ini juga dapat digunakan untuk menganalisis *legacy system* (sistem lama). Hal ini sering terjadi ketika sistem lama perlu dilakukan modifikasi besar, integrasi dengan sistem lain, *porting*, atau peningkatan signifikan lainnya. Dengan asumsi bahwa arsitektur sistem lama tersedia yang diperoleh dengan menggunakan metode ekstraksi arsitektur maka menerapkan metode ATAM dapat meningkatkan pemahaman terkait atribut kualitas dari sistem.

Menurut (Kazman et al., 1998), metode ATAM terdiri dari empat fase:

Tabel II.2 Langkah-langkah ATAM

|  |  |
| --- | --- |
| **Fase I** | Kumpulkan Skenario |
| Mengumpulkan Persyaratan, Batasan, Lingkungan |
| **Fase II** | Gambarkan Tampilan Arsitektur |
| Wujudkan Skenario |
| **Fase III** | Analisis Atribut Spesifik |
| **Fase IV** | Identifikasi Sensitivitas |
| Identifikasi *Tradeoff* |

Tabel II.2 menunjukkan langkah-langkah ATAM melalui empat fase. Langkah pertama dalam fase satu adalah mengumpulkan skenario umum dari para pemangku kepentingan. Sebuah skenario diperlukan untuk mengklarifikasi apa yang harus dicapai sistem dalam bisnis. Langkah kedua adalah fase dua yakni mengumpulkan persyaratan, kendala, dan lingkungan. Persyaratan, kendala, dan pengumpulan lingkungan digunakan untuk menyelesaikan skenario. Semuanya harus menggambarkan dengan jelas atribut kualitas perangkat lunak apa yang perlu dievaluasi pada setiap skenario (misal *realibility* atau *performace*). Atribut kualitas adalah persyaratan nonfungsional seperti *performace, realiblity, security* dan sebagainya. Atribut kualitas penting untuk sistem perlu disebutkan terlebih dahulu. Atribut kualitas diperlukan untuk mengubah skenario umum menjadi skenario pengujian.

Fase kedua adalah penyajian arsitektur dan dibatasi berdasarkan data yang dikumpulkan dari para pemangku kepentingan. Karena yang akan menjadi fokus adalah REST dalam aplikasi maka arsitektur harus mengikuti pondasi REST dan batasan desain yang dibuat oleh Roy Fielding. Arsitektur harus memiliki pertanyaan desain berdasarkan REST (Costa, et al., 2016). Juga, atribut kualitas seperti apa yang terpengaruh dalam setiap pertanyaan desain.

Pada fase ketiga, arsitektur dievaluasi secara individual. Evaluasi didasarkan pada skenario yang dibuat untuk mencapai atribut kualitas tertentu. Atribut kualitas dan skenario umumnya menghasilkan skenario pengujian. Misalnya, untuk mencapai ketersediaan, *server* web harus tetap responsif. Pertanyaan desain yang digunakan pada tahap kedua adalah "Bagaimana melindungi sistem untuk permintaan yang berlebihan?". Akhirnya, skenario pengujian adalah "permintaan ditanggapi rata-rata 0,5 detik".

Tabel II.3 Skenario umum yang dikonversi menjadi skenario pengujian

|  |  |  |  |
| --- | --- | --- | --- |
| **Atribut Kualitas** | **Skenario Umum** | **Pertanyaan Desain** | **Skenario Pengujian** |
| *Availability* | REST web server harus tetap responsif | Bagaimana melindungi sistem dari overload? | Seluruh request ke web server harus direspon dalam waktu rata-rata 0.5 detik |

Tabel II.3 dihasilkan berdasarkan data yang dicapai pada fase pertama dan fase kedua. Fase pertama adalah membuat skenario umum dan atribut kualitas penting. Tahap kedua adalah menganalisis arsitektur berdasarkan pertanyaan desain dan atribut kualitas apa yang terpengaruh. Pada fase ketiga skenario umum diubah menjadi skenario pengujian atau pada bab selanjutnya disebut skenario konkret.

Tahap keempat dianalisis hasilnya dan tentukan titik sensitivitasnya. Titik sensitivitas adalah perubahan signifikan dalam hasil evaluasi. Misalnya, dengan menggunakan arsitektur A waktu respons secara signifikan cepat dalam 10 ms, tetapi setelah berganti ke arsitektur B waktu respons turun menjadi 5s.

Langkah selanjutnya adalah menganalisis *tradeoff* antara arsitektur. Ketika titik sensitivitas arsitektur ditentukan, *tradeoff* dapat ditemukan dengan menganalisis elemen arsitektur yang memiliki beberapa titik sensitivitas. Misalnya, membuat layanan *microservices* untuk sistem dapat meningkatkan *performance* sistem karena beban didistribusikan secara merata di setiap layanan *microsersevices*. Tetapi keamanan bisa menurun karena mengandung lebih banyak layanan daripada arsitektur *monolith* (jumlah serangan potensial meningkat). *Monolith* dan *microservices* kemudian menjadi *tradeoff* dalam hal *performance* dan keamanan. *Monolith* dan *microservices* kemudian menjadi strategi arsitektur (*architecture strategy*)

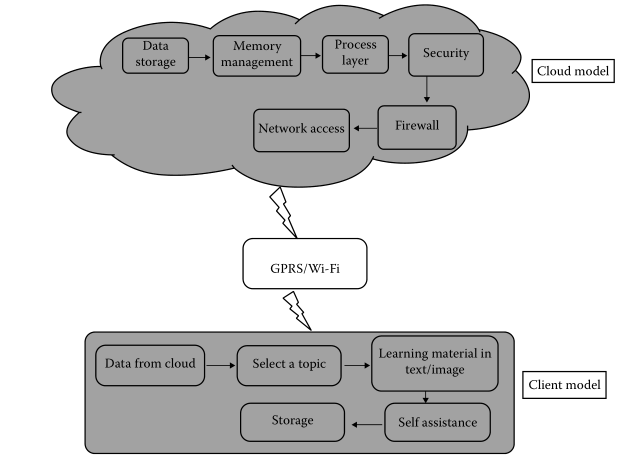
Kesimpulannya, hasil dari ATAM adalah (Bianco, Kotermansk, & Merson, 2007):

1. Risiko, arsitektur yang dipilih akan menciptakan masalah untuk beberapa atribut kualitas di masa yang akan datang. Contoh, *framework* JavaScript versi saat ini bisa saja tidak lagi didukung oleh vendor dan akan menciptakan kerentanan di masa mendatang karena tidak ada tambalan keamanan.
2. Non-risiko, arsitektur yang dipilih meningkatkan performance atribut kualitas. Contoh, dengan menggunakan desain arsitektur REST, performance sistem meningkat 50% karena transfer datanya yang ringan dan mudah.
3. *Tradeoff*, arsitektur yang dipilih akan memiliki *tradeoff* atau dampak pada beberapa atribut kualitas. Contoh, REST mengekpos API tunggal, sehingga memiliki tingkat keamanan yang lebih rendah daripada prosedur SOAP yang memiliki beberapa prosedur keamanan.
4. Titik sensitivitas, suatu simpulan dari *tradeoff*. Contohnya meskipun REST meningkatkan performance, tetapi memiliki tingkat keamanan yang lebih rendah daripada prosedur SOAP.

## Rangkaian Analisis Tinjauan Pustaka (Jurnal Terkait)

### E-Learning dan Komputasi Awan

(Rao, C.Sasidhar, & Kumar, 2010) mengusulkan model pengembangan *E-Learning* melalui model *mobile learning* dengan *cloud computing* seperti pada gambar II.8 berikut :



Gambar II.8 Model *Mobile Learning* dengan *cloud computing*

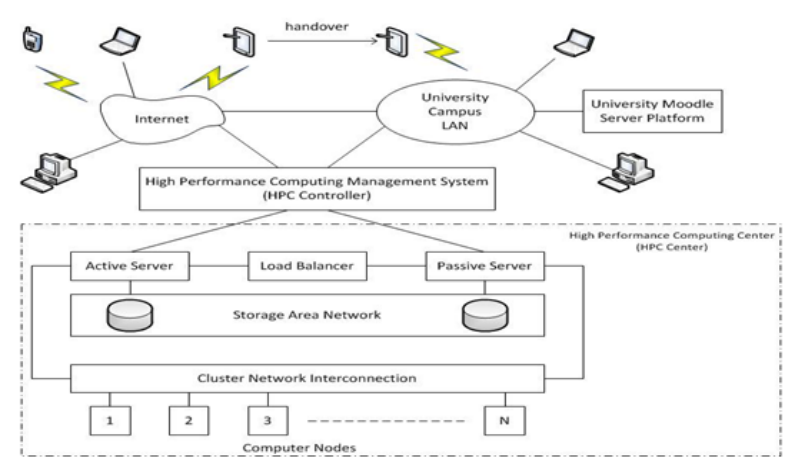
Pada *cloud model* terdiri dari beberapa komponen layanan yang meng*host* sistem E*-Learning* di *cloud* antara lain:

* *Data storage*: digunakan untuk menyimpan data terkait materi pembelajaran di *cloud*
* *Memory management*: pengaturan memory sangat penting dilakukan pada setiap distribusi server di *cloud* untuk mendukung proses komputasi
* *Process layer*: lapisan ini berupakan inti dari komputasi yang terdapat di eksositem *cloud* yang mana setiap mesin virtual di *cloud* membutuhkan processor untuk memproses setiap permintaaan yang dilakukan oleh pihak user di sisi client
* *Security*: layanan yang berjalan di *cloud* seperti aplikasi *E-Learning* memerlukan perhatian terkait keamanan misal bagaimana terkait otentikasi dan otorisasi. Juga pengamanan terkait aplikasi misal bagaimana agar terhindar dari serangan SQL injection.
* *Firewall* : pengaturan firewall yang mumpuni di *cloud* sangatlah penting khusunya terkait keamanan untuk ekosistem server misal agar terhindar dari serangan *Denialed of Service* (DdoS)
* *Network access*: di lapisan ini *cloud* memerlukan pengaturan akses jaringan seperti load balancer agar proses akses dari sisi client lancar dan tidak terganggu meskipun diakses oleh ribuan bahkan jutaan pengguna.

Pada *client model*, *offloading* komputasi yang ada pada sisi *cloud* dikirim ke perangkat mobile yang ada di sisi user/client. User bisa memilih topik pembelajaran yang diinginkan, yang mana topik pembelajaran ini bisa berupa teks, gambar maupun video. Prosesnya berlangsung secara self assistance/self service dan user bisa mengunduh materi tersebut langusng ke perangkat mobilenya.

Adapun benang merah yang dapat diambil dari model yang diusulkan pada gambar 8 di atas, yakni model ini terbatas pada *proof of concept* model *cloud* *computing* pada *mobile learning* tanpa adanya evaluasi seperti pembuatan prototipe terkait desain LMS pada sisi *cloud* maupun desain aplikasi pada sisi client (end user). Namun begitu model yang diusulkan dapat dasar menjadi referensi dalam pengembangan lebih lanjut dalam tataran praktis terkait implementasi mobile learning berbasis *cloud* *computing*.

Model *E-learning* berikutnya adalahh model yang diajukan oleh (Kitanov & Davcev, 2012) dengan nama *High Performace Computing* (HPC) untuk Mobile *Distance Learning*. Gambar modelnya adalah sebagai berikut:



Gambar II.9 HPC untuk ekosistem Mobile E-Learning

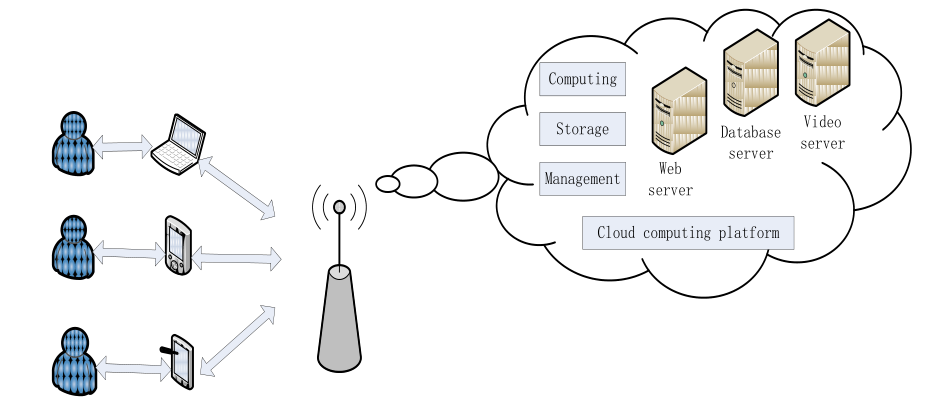
Pada model di gambar II.9 menyajikan sistem *mobile learning* yang infrastrukturnya berada di dalam *private cloud* universitas. Pada level bawah terdapat interkoneksi jaringan *cluster* yang menghubungkan banyak *node* komputer (*slave*) untuk melayani pemrosesan komputasi sejumlah besar pengguna yang mana *node-node* ini terhubung dengan komputer *master* berupa server yang sifatnya aktif maupun pasif. Server ini mengutilisasi *Storage Area Network* untuk penyimpanan aplikasi dan data konten edukasi. Server aktif dan pasif ini dihubungkan dengan *load balancer* yang berfungsi untuk membagi beban *traffic* akses *mobile learning* dari pengguna sehingga tidak terjadi *delay* di sisi pengguna. Selain itu pula load balancer ini mengatur sistem *mobile learning* jika terjadi kegagalan pada salah satu server. Dengan demikian tidak terjadi gangguan pada layanan.

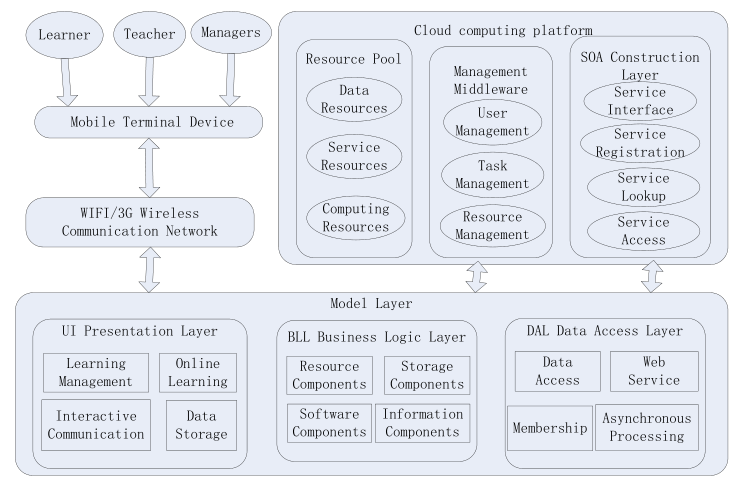
Adapun dari sisi aplikasi, model *mobile learning* ini secara spesifik telah menggunakan software/ aplikasi *Learning Managegement System* (LMS) *opensource* yakni moodle.org yang menyediakan API untuk konektivitas konten *E-Learning* antar universitas.

Pada model ini, keseluruhan sistem di atas disebut *High Performace Computing (HPC). High-performance Computing (HPC)* sebenarnya adalah terminologi penggunaan *cluster* komputer atau super komputer dan teknik *paralel processing* untuk memecahkan permasalahan komputasi yang kompleks. HPC khususnya digunakan untuk kegiatan penelitian melalui pemodelan komputer, simulasi dan analisis. Berbeda dengan model pertama yang menggunakan linux, pada model kedua ini HPC controller menggunakan Windows HPC Server 2008 SP1 sebagai operating system layanan *cluster.*

HPC terkoneksi dengan *Local Area Network* (LAN) universitas dan tentunya terkoneksi dengan internet. Dari model *mobile learning* ini, pengguna dari luar universitas bisa mengakses konten *mobile learning* unversitas lain yang terhubung lewat API *platform* Moodle.

(Chen et al., 2013) mengusulkan model pengembangan E*-Learning* bentuknya adalah sebagai berikut:





Gambar II.10 Struktur system mobile E-Learning berbasiskan cloud computing oleh Chen, et.al.

Desain sistem model seperti pada gambar II.10 terdiri dari 4 bagian yakni: *mobile learning client*, jaringan komunikasi *wireless*, platform jaringan berbasiskan arsitektur 3 lapis dan platform *cloud* *computing*.

1. *Mobile learning client*: ini adalah media interaktif pengguna dan sistem, dan digunakan untuk mempresentasikan resources yang dipush oleh *cloud* *computing* kepada pengguna. Peralatan pendukungnya adalah perangkat seluler cerdas dengan sistem mobile learning berbasiskan *cloud* *computing*.
2. Jaringan komunikasi nirkabel: Pengguna berkomunikasi dengan platform jaringan dengan peralatan terminal seluler melalui jaringan nirkabel WIFI / 3G. Jaringan nirkabel dapat berupa jaringan pribadi yang dibangun oleh beberapa agen khusus (seperti berbagai jaringan pendidikan), juga dapat berupa jaringan publik yang disediakan oleh operator jaringan (seperti jaringan 3G disediakan oleh operator seluler, selama mereka dapat menyediakan fungsi akses nirkabel dan seluler, juga dapat mencapai bandwidth tertentu.
3. Platform jaringan berdasarkan arsitektur tiga-tier:

Platform jaringan dibagi menjadi lapisan presentasi antarmuka pengguna, lapisan logika bisnis dan lapisan akses data, peran utamanya adalah untuk mengidentifikasi layanan seperti apa yang Anda butuhkan, dan kemudian memanggil antarmuka yang relevan untuk mengakses layanan terkait yang disediakan oleh platform komputasi awan.

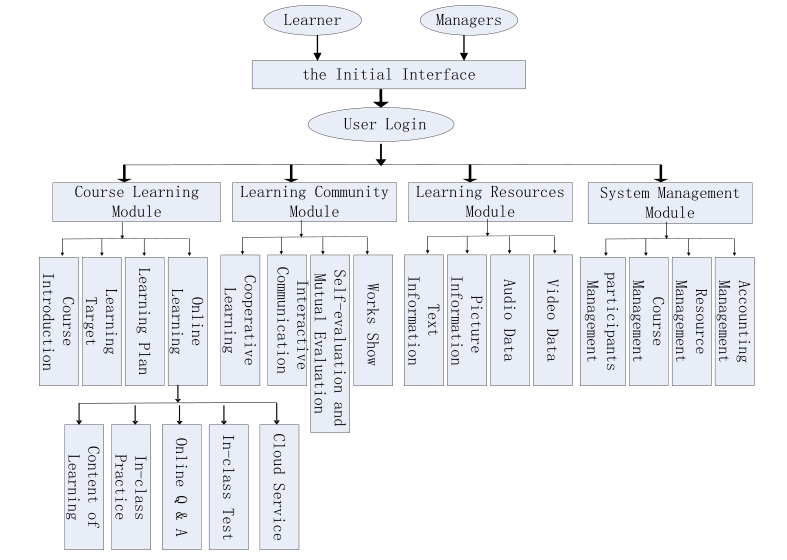
* + Lapisan UI presentation adalah antarmuka interaktif pengguna dan sistem, dan bertanggung jawab untuk sesi pemrosesan dengan pengguna, dan mewujudkan komunikasi antara pengguna dan lapisan logika bisnis dengan menjalankan program di browser seluler;
  + Lapisan logika bisnis bertanggung jawab untuk memproses permintaan layanan aplikasi dari lapisan UI dan membuat analisis logis bisnis, jika menemukan bahwa bisnis perlu mengakses lapisan akses data, maka lapisan logika bisnis akan memanggil layanan yang relevan dalam data lapisan akses dan meneruskan bisnis ke lapisan akses data, jika tidak perlu mengakses lapisan akses data, bisnis akan diteruskan ke platform *cloud* *computing* secara langsung melalui antarmuka yang relevan dan diproses oleh server *cloud*;
  + Lapisan akses data digunakan untuk menerima permintaan bisnis dari lapisan logika bisnis, dan meneruskan bisnis ke platform *cloud* *computing* dengan menganalisis dan memanggil antarmuka yang relevan untuk diproses.

1. Platform *cloud* *computing*:

Ini adalah bagian inti dari sistem dan modul eksekusi akhir dari keseluruhan fungsi sistem. Layanan yang diperlukan pengguna untuk menghubungkan platform komputasi awan melalui antarmuka dan dilakukan pemrosesan, kemudian data yang diproses diteruskan ke pengguna pada gilirannya oleh antarmuka . Dalam lapisan kumpulan sumber daya, sejumlah besar sumber daya komputasi terdistribusi isomorfik atau dekat dengan sumber daya isomorfik membentuk berbagai kumpulan sumber daya virtual melalui teknologi virtualisasi, seperti: kumpulan sumber daya data, kumpulan sumber daya layanan, kumpulan sumber daya komputasi dll., Lapisan ini adalah dasar dari platform *cloud* *computing*; layer manajemen middleware terutama bertanggung jawab untuk berbagai manajemen platform komputasi awan termasuk manajemen pengguna, manajemen tugas dan manajemen sumber daya, dll; Lapisan konstruksi SOA merangkum semua jenis sumber daya komputasi awan ke dalam layanan WEB standar untuk pengguna, termasuk antarmuka layanan, pendaftaran layanan, penemuan layanan dan akses layanan, dll.

Dari model arsitektur di atas terlihat layanan-layanan yang mendukung sebuah ekosistem mobile learning dari end user sampai ke layanan *cloud*.

Adapun dari sisi LMS-nya sendiri model yang diajukannya adalah terdiri dari beberapa modul sebagai berikut

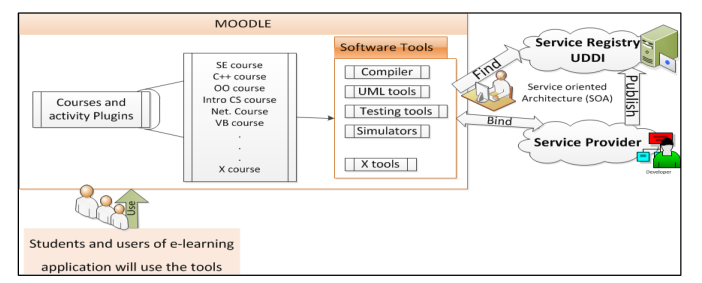


Gambar II.11 Gambaran model LMS yang diajukan oleh Chen, et.al

### E-Learning dan SOA

(Al-Ajlan & Zedan, 2007) telah melakukan studi yang bertujuan untuk memperluas ide inti di balik tool *Virtual Learning Environment* (VLE) yakni Moodle yang mendominasi institusi akademik. Kontribusi yang ingin dituju adalah membangun VLE dengan pendekatan konsep *web service* yaitu SOA dan teknik terkaitnya. Kontribusi mendasar dari studi yang diusulkan adalah untuk menunjukkan bahwa VLE dapat dimungkinkan sebagai layanan yang dapat dipublikasikan, ditemukan (*discovered*) dan disusun sebagaimana dipersepsikan dalam paradigma SOC (*Service Oriented Computing*).

(Zuhoor Al-Khanjari, Al-Roshdi, & Kraiem, 2014) dalam jurnal penelitiannya yang berjudul Pengembangan desain arsitektur lanjut untuk *resource virtual* pada pembelajaran ilmu komputer dengan menggunakan SOA mengajukan pengembangan desain arsitektur LMS Moodle dengan menggunakan prinsip SOA. Pada jurnal ini peneliti berusaha memanfaatkan fitur web service yang ada di LMS untuk konektivitas dengan aplikasi external seperti virtual lab/ virtual kompiler. Dengan penggunaan SOA ini dapat mengatasi keterbatasan LMS dalam mendukung kegiatan praktikum bidang ilmu komputer. Berikut adalah gambarannya:



Gambar II.12 LMS Moodle yang terhubung dengan software tools untuk praktikum ilmu komputer menggunakan prinsip SOA

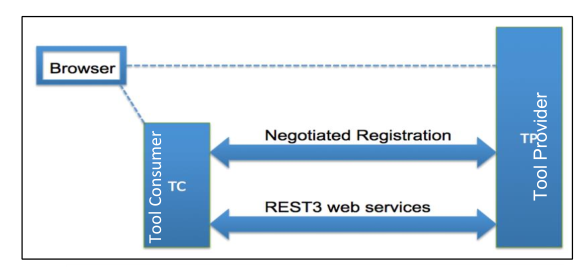
Pendekatan yang diusulkan sebagai *service* yang bertujuan untuk memperluas desain arsitektur Moodle untuk mendukung perkuliahan ilmu komputer dengan mengkoneksikannya dengan software tools external untuk praktikum ilmu komputer sepeti kompiler, UML, *testing tool* dan simulator.

Perluasan desain arsitektur Moodle lebih mudah daripada LMS lain karena ini tersedia sebagai *open source* dan mendukung plugin eksternal. Moodle juga mengimplementasikan web service dalam platformnya sebagai struktur untuk memfasilitasi pengembangannya menggunakan SOA. Ini mengikuti filosofi kegunaan dan fungsionalitas dari *web service.* Oleh karena itu, Moodle mengimplementasikan protokol web service yang berbeda sejak Moodle versi 2.0 sampai versi terbaru. Protokol-protokol ini adalah:

* XML-Remote Procedure Call (XML-RPC).
* Action Message Format (AMF).
* Representational State Transfer (REST).
* Simple Object Access Protocol (SOAP).

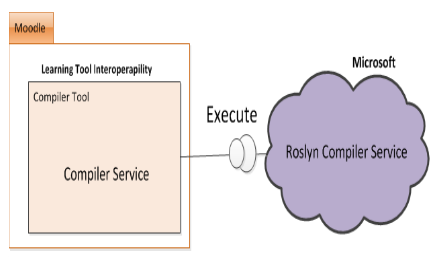
Untuk mengembangkan layanan yang dibutuhkan ada beberapa persyaratan sistem, yang harus dipertimbangkan. Persyaratan ini meliputi: Moodle v2.5, Windows Server 2008 R2, .NET lingkungan Microsoft, *Learning Interoperability Tool* (LTI) dan *tool* perangkat lunak yang diperlukan untuk ilmu komputer seperti: *compiler*.

Dalam arsitektur yang diusulkan ini, harus mempertimbangkan interoperabilitas antara *tools* dan LMS tidak hanya untuk konten tetapi untuk cakupan yang lebih luas dari fitur dan layanan yang dapat ditawarkan aplikasi pembelajaran. Oleh karena itu, pada tahun 2005 Konsorsium Pembelajaran Global IMS menerbitkan tool baru untuk integrasi antara alat dan LMS yang merupakan *Learning Interoperability Tool* (LTI) atau dikenal sebagai Basic LTI. Ini terdiri dari dua elemen utama: *Tool Provider* (TP) di sisi alat dan *Tool Consumer* (TC) di sisi platform pembelajaran . Seperti yang ditunjukkan pada Gambar 2.10, komunikasi antara TC dan TP dapat diaktifkan melalui menggunakan protokol REST dari layanan web dan negosiasi dapat melalui menyediakan tiga informasi utama dari TP ke TC yaitu: URL dari *tool*, *key*, *shared secret* antara kedua aplikasi tersebut.



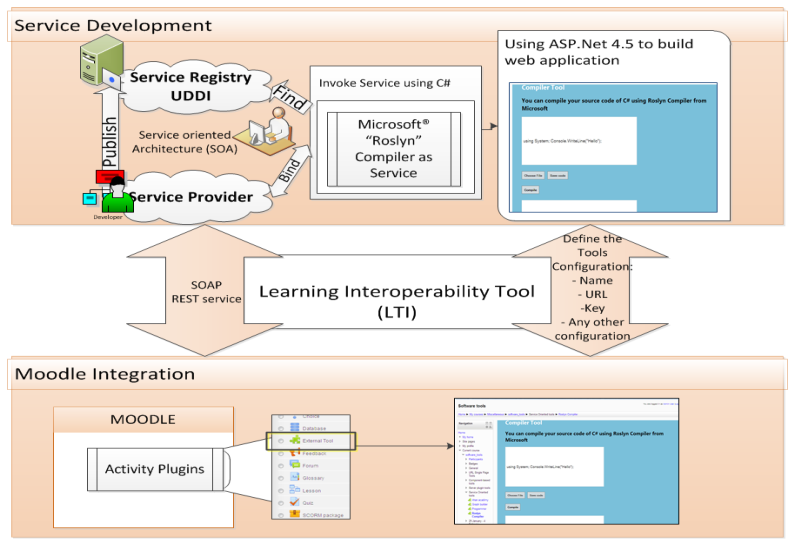
Gambar II.13 Komunikasi antara Tool Consumer dan Tool Provider via REST web services

Integrasi harus didefinisikan baik dengan menggunakan *Learning Tool Interoperability* (LTI) sehingga integrasi aplikasi itu dapat dilakukan di lingkungan Moodle dengan menggunakan *Web Service Description Language* (WSDL). Untuk integrasi layanan-layanan tersebut ke dalam Moodle, eksekusi interface dari scripting API digunakan untuk pengembangan tool kompiler seperti yang ditunjukkan pada Gambar 2.14. Antarmuka ini bertanggung jawab untuk mengeksekusi kode saat *runtime.*



Gambar II.14 Penggunaan Service API Learning Interoperabilty Tool dengan Roslyn Compiler Service

Gambar II.15 di bawah ini menunjukkan keseluruhan arsitektur dan lapisan proses pengembangan untuk layanan :

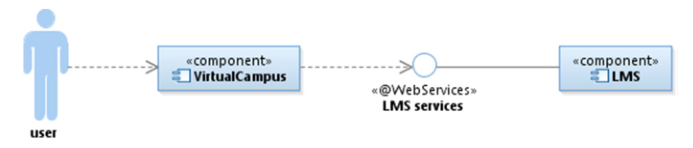


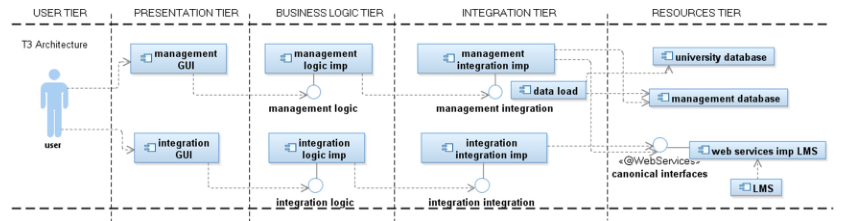
Gambar II.15 Arsitekur Moodle *Integration* dan *Service Development* *dengan Learning Interoperability Tool*

Adapun benang merah dari model intergrasi LMS ini adalah sebagai berikut:

1. *Learning Interoperability Tool* (LTI) cukup berperan sebagai *midleware* dalam proses integrasi antara LMS dengan eksternal software tool seperti virtual compiler
2. Protokol REST web service membantu dalam proses eksekusi ke tool external Roslyn *Compiler as a Service*
3. Pendekatan *Service Oriented Architecture* (SOA) mendukung fungsionalitas dan memperluas desain LMS yang sudah ada yakni Moodle sehingga bisa diintegrasikan dengan *virtual lab* melalui fitur *single sign on*.
4. Pada model ini, konektivitas LTI yang menghubungkan Moodle dengan *tool compiler* Roslyn masih dalam *enviroment* *on premise* dan belum memanfaatkan fitur layanan *cloud* *computing*

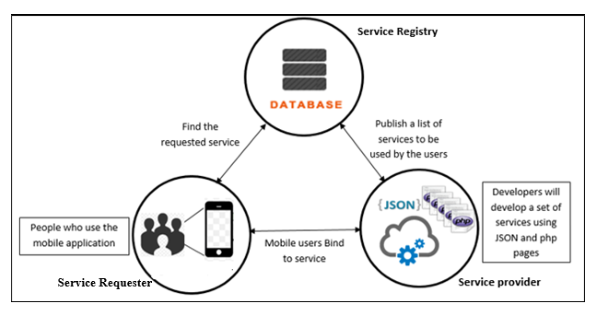
Selanjutnya (Huertas & Navarro, 2015) mengusulkan protipe virtual campus (VC) yang berbasis LMS dan mengusulkan arsitektur VC alternatif dimana bergantung pada infrastuktur SOA yang mengisolasi VC dari platform *E-Learning*. Mendesain interface yang bernama *canonical interface* berupa API yang bisa menghubungkan platform LMS dengan aritektur yang berbeda yakni: Blackboard, Moodle dan Sakai. Dengan demikian ini bisa mempersingkat waktu implementasi pagi stakeholder pendidikan terkait dan bisa mempromosikan evolusinya menuju model SaaS.





Gambar II.16 Arsitektur *Virtual Campus* dengan *canonical interface* untuk menjembatani LMS dengan arsitektur yang berbeda

Tinjauan literatur berikutnya adalah terkait Pengembangan arsitektur aplikasi mobile untuk pendidikan menggunakan SOA yang diajukan oleh (Z. Al-Khanjari, Al-Kindi, Al-Kindi, & Kraiem, 2015). Pada penelitian ini mengetengahkan latar belakang terkait tantangan yang dimiliki dunia pendidikan yang selama ini menggunakan platform *E-Learning* namun belum menjamah ekosistem mobile agar para siswa lebih interaktif lagi dalam mengikuti pembelajaran. Institusi pendidikan masih terkendala dengan aspek keamanan terkait penerapan sistem *E-Learning* pada perangkat mobile. Rumusan masalah penelitian ini adalah bagaimana mengembangkan sistem aplikasi mobile dengan prinsip SOA yang bisa digunakan secara aman dan terkoneksi dengan sistem E-Leaning yang sudah ada. Prinsip yang digunakan tergambar pada gambar sebagai berikut:

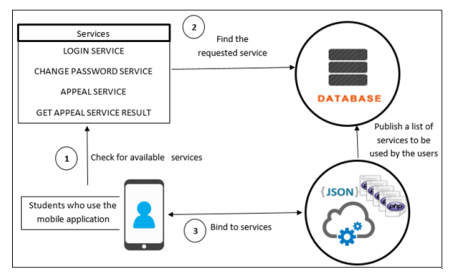


Gambar II.17 Tiga roles dalam Service-Oriented Architecture (SOA ) dalam pengembangan asritektur aplikasi mobile untuk pendidikan

1. Komponen pertama adalah *Service provider* yang bertanggung jawab untuk merancang dan mengimplementasikan services, menentukan antarmuka untuk *services* dan menerbitkan informasi terkait services ini di registri pusat (*directory services*).
2. Komponen kedua disebut *Service Requester* itu juga disebut *client* atau entitas yang membutuhkan fungsi (services) tertentu untuk melakukan beberapa tugas atau mungkin aplikasi atau *service* yang bergantung pada interaksi dengan beberapa layanan.
3. Yang ketiga adalah *Service Registry* (broker) yang merupakan lokasi deskripsi layanan (yaitu di mana *Service Provider* menerbitkan deskripsi layanan) . Komponen yang disebutkan di atas berinteraksi satu sama lain seperti yang diberikan di bawah ini:

* *Publish* - mencatat deskripsi layanan dalam direktori layanan, yang mencakup pendaftaran kemampuan, antarmuka, performance, dan kualitas yang ditawarkannya.
* *Find* - mencari layanan yang terdaftar di direktori layanan, asalkan memenuhi kriteria yang diinginkan dan dapat digunakan dalam proses bisnis, dengan mempertimbangkan deskripsi layanan yang diterbitkan.
* *Bind* - operasi ini bergantung pada layanan yang diminta atau melakukan interaksi dengan layanan pada saat runtime menggunakan informasi yang diperoleh dalam *service discovery.*

Developer pada penelitian saat ini mengembangkan hasil akhir berupa aplikasi *mobile* menggunakan SOA untuk siswa. Aplikasi ini adalah alat pendidikan yang digunakan untuk mendukung siswa selama masa studi. Sistem ini menyediakan permintaan layanan yang merupakan layanan baru yang dapat diakses menggunakan perangkat Android. Berikut adalah gambaran hasil akhir dari model SOA pada penelitian ini:



Gambar II.18 Gambaran arsitektur Mobile Learning dengan JSON

Gambar II.18 menunjukkan *framework* hasil akhir dari aplikasi mobile yang mengintegrasikan berbagai komponen menggunakan arsitektur SOA. Gambar tersebut menggambarkan layanan yang disediakan sistem untuk siswa, termasuk layanan login dan *change password* sebagai modul otentikasi. Semua layanan dapat menggunakan database yang menyimpan informasi terkait siswa. Selain itu, begitu siswa memiliki hak untuk mengakses aplikasi maka dia akan dapat menggunakan layanan lain. Gambar ini menggambarkan permintaan di seluruh jaringan untuk mengakses layanan melalui JSON (*JavaScript Object Notation*) yang merupakan penyampaian pertukaran-data yang ringan. Ini didasarkan pada subset dari Bahasa Pemrograman JavaScript untuk menanggapi client dengan data yang akurat.

### Representational State Transfer (REST)

REST menjadi populer sebagai solusi arsitektur dalam membangun layanan web. (Adamczyk, et al., 2011) mengevaluasi perbedaan antara REST dan SOAP sebagai layanan web. Penelitian ini juga menganalisis apakah *tools* yang digunakan untuk membuat REST sekarang sesuai dengan prinsip standar REST. Aplikasi layanan REST meningkat berdasarkan survei yang dilakukan pada 2007-2010. Aplikasi REST semakin populer pada tahun 2011 dan menunjukkan REST adalah arsitektur yang tepat untuk web. REST juga menghadapi tantangan dalam keamanan, standar desain, dan memecahkan masalah bagi perusahaan. REST mulai mendapatkan popularitas sejak 2008 dan berubah menjadi kerangka kerja yang lebih kuat dan holistik (Garriga, et al., 2016).

(Khan & Abbasi, 2015) menggambarkan arsitektur SOAP dan REST dalam layanan web. Kedua arsitektur kemudian dibandingkan menggunakan parameter yang berbeda seperti:

1. *Weight*

Ukuran layanan, layanan lebih ringan memiliki *performance* yang lebih baik. SOAP lebih berat daripada REST karena standarnya. Setiap respon yang datang dari layanan SOAP dikemas dengan beberapa prosedur. Respon dari REST hanya berisi data yang dibutuhkan. Itulah sebabnya yang sering disebut SOAP adalah mengirim surat dengan amplop sementara REST menggunakan kartu pos.

1. *Coupling*

Coupling menjelaskan bagaimana layanan tergantung atau mengikat ke *server*. *Server* web SOAP biasanya sangat bergantung pada server sementara REST tidak. Ini memberi REST manfaat lebih dibandingkan SOAP terutama dalam masalah *scalability*.

1. *Overhead*

REST tidak memiliki muatan tambahan, sementara SOAP menggunakan "amplop" untuk mengirim permintaan dan mengambil respons. Latensi *payload* tambahan meningkat dan membuat REST memiliki latensi yang lebih baik daripada SOAP.

1. Operasi CRUD

CRUD berarti Create, Read, Update, dan Delete dalam operasi basis data. REST mengambil lebih banyak keuntungan dalam proses CRUD daripada SOAP. Misalnya untuk menghapus *record*, REST menggunakan metode DELETE HTTP dengan id item di URL. Sementara itu SOAP, harus menyertakan data XML untuk mencapai fungsi ini.

1. *Network Latency*

REST memiliki latensi jaringan yang lebih baik daripada SOAP. Latensi REST meningkat secara linier sementara SOAP meningkat secara eksponensial.

1. *Security*

*Security* berarti bagaimana data berpindah dari satu tempat ke tempat lain dan tidak dapat diganggu. Layanan SOAP lebih aman daripada REST karena SOAP telah memperketat aturan dan standar yang harus diikuti. SOAP memiliki standar terperinci untuk transformasi data dan format keamanan. Sementara itu REST tidak menggunakan format keamanan spesifik apa pun.

Dari penjelasan di atas jelas terlihat bahwa REST memiliki kelebihan dibanding SOAP. (Khan & Abbasi, 2015) juga menguji latensi CRUD dan ukuran paket antara REST dan SOAP. Hasilnya adalah REST memiliki *performance* yang lebih baik dalam latensi dan ukuran paket yang lebih kecil. Makalah ini juga melakukan tes respons permintaan sinkron. REST terbukti lebih efisien dalam menangani respons permintaan yang sinkron. Kesimpulannya, REST selalu menjadi pilihan yang disukai di antara pengembang. Tetapi ketika keamanan menjadi perhatian utama seperti perbankan, SOAP jelas merupakan solusi yang lebih baik. Baik pada SOAP dan REST memiliki nilai positif. Pertanyaannya bukan solusi mana yang lebih baik tetapi bagaimana menggabungkan kedua solusi untuk mendapatkan manfaat maksimal.

Fielding, pembuat arsitektur REST juga menerbitkan makalah pada 2017 tentang bagaimana REST diimplementasikan dalam arsitektur web modern (Fielding et al., 2017). Sekarang, REST telah menjadi standar industri pada arsitektur perangkat lunak skala besar. Sistem menjadi lebih terdesentralisasi melalui layanan *microservices* yang mana dengan pendekatan ini *developer* dapat mendistribusikan beban pada setiap proses bisnis ke dalam layanan web terpisah.

### Architecure Tradeoff Analysis Method (ATAM)

Dalam makalah penelitian aslinya oleh (Kazman et al., 1998), Metode ATAM digunakan untuk mengevaluasi sensor suhu jarak jauh. Kerangka kerja ini membantu menentukan karakteristik yang berguna dari setiap opsi arsitektur. ATAM membantu menentukan setiap lokasi titik *tradeoff* arsitektural, dan itu membuat kita memahami batasan setiap opsi. Informasi ini berguna untuk membuat rencana aksi untuk melakukan evaluasi, memulai iterasi baru dari metode, dan memodifikasi arsitektur berdasarkan evaluasi. ATAM dibuat untuk membuat pilihan yang mungkin dan rasional antara opsi arsitektur perangkat lunak. Tidak hanya itu, ia juga berusaha meningkatkan kualitas arsitektur di setiap iterasi metode.

Untuk mengevaluasi atribut kualitas dan memahami pertukaran antara arsitektur, tes berbasis skenario harus dilakukan. Skenario pengujian tunggal harus dapat mencerminkan atribut kualitas perangkat lunak apa yang harus dicapai. (Bass, et al., 2001) mengorganisir laporan skenario ATAM ke dalam skenario umum yang terdiri dari kualitas yang dikaitkan untuk dicapai. Dengan memetakan skenario tertentu ke dalam skenario umum, dapat menunjukkan atribut kualitas yang menjadi perhatian di setiap proyek pengembangan perangkat lunak.

(Barbacci et al. 2003) menggunakan ATAM untuk mengevaluasi arsitektur perangkat lunak untuk lini produk sistem avionik. Eksperimen ini sampai pada kesimpulan bahwa ATAM dapat meningkatkan kesadaran kepada para *stakeholder*. Evaluasi arsitektur yang dilakukan sebelum kode dikembangkan dapat menyelesaikan risiko yang muncul sebelum terlalu mahal untuk diperbaiki. Mengevaluasi arsitektur sebelum atau ketika sistem dikembangkan juga bisa efektif dalam menghadapi bencana di masa depan. Hal ini juga membantu mengevaluasi insinyur arsitektur perangkat lunak dalam pembuatan software.

(Reza & Helps, 2013) menyajikan analisis kualitatif aspek keamanan aplikasi Berbasis Web yang memanfaatkan Arsitektur Berorientasi Layanan (SOA). Solusi arsitektur yang membahas persyaratan keamanan diperiksa dan dibandingkan dengan atribut kualitas lain yang relevan dengan sistem berbasis web. Lebih khusus lagi, analisis *tradeoff* berdasarkan ATAM dilakukan untuk menunjukkan korelasi antara keamanan dan kualitas sistem lainnya yang terkait dengan pemilihan SOA yang sukses. Solusi arsitektural yang optimal tidak hanya memenuhi persyaratan keamanan sistem berbasis web tetapi juga memenuhi atribut kualitas lainnya seperti *performance*, *availability*, kegunaan, kemampuan modifikasi, dan lain-lain.

Berdasarkan tinjauan pustaka di atas, maka berikut adalah rangkumannya dalam bentuk table 4 berikut:

Tabel II.4 Hasil tinjauan pustaka

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Publikasi** | **Topik Penelitian** | **Metode Penelitian** |
| 1 | (Al-Ajlan & Zedan, 2007) | *E-Learning* Moodle Berdasarkan Service Oriented Architecture (SOA) | Melakukan refinement desain arsitektur Moodle sehingga mendekati prinsip SOA |
| 2 | (Rao, Sasidhar, & Kumar, 2010) | *Cloud computing* pada implementasi mobile learning | Mengajukan model dan proof of concept sistem *cloud computing* pada mobile learning yang terdiri dari *cloud* model dan client model |
| 3 | (Stojan Kitanov, 2012) | Implementasi mobile learning pada lingkungan *mobile cloud computing* | Metode yang dilakukan peneliti adalah dengan melakukan survey dan studi kasus terhadap desain implemtantasi mobile learning berbasis private *cloud* di universitas. |
| 4 | (Chen, et al., 2013) | System *Mobile Learning* berbasiskan *cloud computing* | Studi konsep dan arsitektur sistem komputasi awan, kemudian merancang struktur sistem *mobile learning* berbasiskan *cloud computing* dan menganalisis modul proses dan fungsinya, merealisasikan beberapa modul fungsi sistem dengan teknologi JAVA dan platform Android |
| 5 | (Al-Khanjari, Al-Roshdi, & Kraiem, 2014) | Pengembangan desain arsitektur lanjut untuk resource virtual pada pembelajaran ilmu komputer dengan menggunakan SOA | Analisis aplikasi LMS yang sudah ada (Moodle), mengembangkan desain untuk konektivitas ke virtual lab berupa Roslyn Compiler Service |
| 6 | (Al-Khanjari, Al-Kindi, Al-Kindi, & Kraiem, 2015) | Pengembangan arsitektur aplikasi mobile untuk pendidikan menggunakan SOA | Memetakan sistem *E-Learning* yang sudah ada, kemudian mengajukan model arsitektur aplikasi mobile setelah itu membuat prototype aplikasi mobile dengan protokol data exchange JSON sehingga terkoneksi ke sistem *E-Learning* utama. |
| 7 | (Huertas & Navarro, 2015) | Dukungan SOA pada arsitektur maju virtual campus dengan antara muka yang bersifat kanonik | Membandingkan beberapa protipe virtual campus (VC) yang berbasis LMS dan mengusulkan arsitektur VC alternatif dimana bergantung pada infrastuktur SOA yang mengisolasi VC dari platform eLearning. Mendesain interface yang bernama canonical interface berupa API yang bisa menghubungkan platform LMS yang berbeda: Blackboard, Moodle dan Sakai. |
| 8 | (Reza & Helps, 2013) | Analisis Security Trade-off dari Arsitektur Software Berorientasi Layanan | Menggunakan metode ATAM untuk menunjukkan korelasi antara keamanan dan kualitas sistem berbasis web yang menerapkan SOA |
| 9 | (Adamczyk, et al., 2011) | REST dan Web Services: Dalam teori dan praktek | Terdapat dua gaya arsitektur yang digunakan untuk membuat Web Services yakni berbasiskan Web Services Standard (SOAP WS) dan berbasiskan RESTful services. Pemahaman dua jenis yang berbeda ini sangatlah penting untuk pengembangan di level Enterprise. |
| 10 | (Garriga, et al., 2016) | Survey terkait ulasan komposisi RESTful Services | Survery terkait penerapan REST semakin populer pada tahun 2011 dan menunjukkan REST adalah arsitektur yang tepat untuk web. REST juga menghadapi tantangan dalam keamanan, standar desain, dan memecahkan masalah bagi perusahaan. REST mulai mendapatkan popularitas sejak 2008 dan berubah menjadi kerangka kerja yang lebih kuat dan holistik untuk web. |
| 11 | (Khan & Abbasi, 2015) | Membedakan Parameter-parameter untuk memilih Arsitektur SOAP vs. Arsitektur berbasis REST | Menjelaskan arsitektur layanan web berbasis SOAP dan REST dan kemudian membandingkan keduanya menggunakan parameter yang berbeda. Tujuannya adalah membantu pengembang solusi dalam memilih layanan web terbaik dengan mempertimbangkan skenario yang diberikan. Parameter perbandingan mencakup banyak dimensi seperti Coupling, Format, Operasi CRUD, dan Latensi Jaringan. Makalah ini juga membahas contoh penerapan kedua jenis layanan web pada kerangka kerja middleware dan membandingkan hasil berdasarkan performance berkaitan dengan efisiensi dan *scalability*. |
| 12 | (Fielding et al., 2017) | Refleksi pada Gaya Arsitektur REST dan Prinsip Desain Arsitektur Web Modern | REST diimplementasikan dalam arsitektur web modern. REST bukanlah arsitektur, melainkan gaya arsitektur. REST merupakan serangkaian constraint yang, jika dipatuhi akan menginduksi serangkaian properti; sebagian besar properti tersebut diyakini bermanfaat untuk desentralisasi, aplikasi berbasis jaringan, sementara yang lainnya merupakan trade-off negatif yang dapat dihasilkan dari setiap pilihan desain. |
| 13 | (Kazman et al., 1998) | Metode Analisis Tradeoff Architecture (The Architecture Tradeoff Analysis Method - ATAM) | ATAM membantu menentukan setiap lokasi poin tradeoff arsitektural, dan itu membuat kita memahami batasan setiap opsi. Informasi ini berguna untuk membuat rencana aksi untuk melakukan evaluasi, memulai iterasi baru dari metode, dan memodifikasi arsitektur berdasarkan evaluasi. ATAM dibuat untuk membuat pilihan yang mungkin dan rasional antara opsi arsitektur perangkat lunak. Tidak hanya itu, metode ini juga berusaha meningkatkan kualitas arsitektur di setiap iterasi metode. |
| 14 | (Bass, et al., 2001) |  | Mengorganisir laporan skenario ATAM ke dalam skenario umum yang terdiri dari kualitas yang dikaitkan untuk dicapai. Dengan memetakan skenario tertentu ke dalam skenario umum, dapat menunjukkan atribut kualitas yang menjadi perhatian di setiap proyek pengembangan perangkat lunak. |
| 15 | (Barbacci et al. 2003) | Menggunakan ATAM untuk mengevaluasi arsitektur perangkat lunak untuk lini produk sistem avionik | Eksperimen ini sampai pada kesimpulan bahwa ATAM dapat meningkatkan kesadaran kepada para pemangku kepentingan. Evaluasi arsitektur yang dilakukan sebelum kode dikembangkan dapat menyelesaikan risiko yang muncul sebelum terlalu mahal untuk diperbaiki. Mengevaluasi arsitektur sebelum atau ketika sistem dikembangkan juga bisa efektif dalam menghadapi bencana di masa depan. |
| 16 | (Costa et al., 2016) | Evaluasi arsitektur REST – Pendekatan, Perangkat dan Pedoman | Mengusulkan pendekatan, perangkat, dan pedoman untuk membantu kegiatan evaluasi arsitektur dalam sistem berbasis REST. Pedoman ini dapat digunakan secara sistematis bersama dengan metode evaluasi untuk alasan tentang pertimbangan desain dan tradeoff. Untuk mendemonstrasikan bagaimana pedoman dapat membantu evaluator arsitektur, disajikan proof of concept yang menggambarkan bagaimana menggunakan pedoman dalam suatu evaluasi ATAM. |

Pada penelitian ini akan mengkombinasikan beberapa metode yang telah digunakan pada beberapa tinjauan pustaka di atas. LMS Moodle akan diterapkan pada ekosistem komputasi awan dengan beberapa skenario *deployment*. Kemudian pondasi REST, *template* skenario atribut kualitas dan pertanyaan desain yang diajukan oleh Costa et al. (Costa et al., 2016) digunakan untuk evaluasi dan optimalisasi terkait arsitektur REST LMS Moodle pada ekosistem komputasi awan sehingga menghasilkan nilai *trade-off* yang optimal baik dari sisi *performance dan scalability.*

# METODOLOGI

## Kerangka Pikir

Arsitektur awal dari sistem LMS Moodle yang umumnya diterapkan di lapangan adalah bersifat *monolith*, yakni semua kerangka kerja baik itu *resource* front end, *backend*, database dan sistem penyimpanan file semua berada di dalam satu server. Hal ini menimbulkan tantangan bagi *stakeholder* pendidikan dimana mereka harus menyediakan investasi infrastuktur di awal berupa *server* yang mahal dan tidak *scalable*. Penelitian ini berusaha untuk memecahkan tantangan tersebut yakni bagaimana menyediakan sistem *E-Learning* yang terjangkau, *scalable* dan memiliki waktu *deployment* yang cepat untuk diterapkan di banyak lembaga pendidikan yang berbeda. Jadi target utamanya adalah membuat sistem *E-Learning* yang memiliki model *Software as a Service* (SaaS) pada ekosistem komputasi awan dengan *performance* dan *scalability* yang baik.

*Scalability* menjadi perhatian utama dalam memilih arsitektur yang tepat untuk sistem *E-learning* ini. Dari masalah di atas, terdapat urgensi untuk meneliti arsitektur baru yang dapat diukur untuk digunakan dalam teknologi pendidikan tersebut. Arsitektur perlu dievaluasi untuk mencapai standar tertentu dalam *scalability* dan atribut kualitas terkait lainnya. Output evaluasi digunakan untuk memperbaiki arsitektur.

Gambar III.1 menggambarkan aktivitas penelitian yang akan dijalani untuk mengevaluasi dan mengoptimaliasi arsitektur REST LMS Moodle pada ekosistem komputasi awan. Tinjauan literatur sudah dibahas dalam bab sebelumnya. Bab ini membahas proses desain untuk *deployment* arsitektur awal dalam LMS Moodle di ekosistem komputasi awan , batasan REST, dan pola desain SOA.



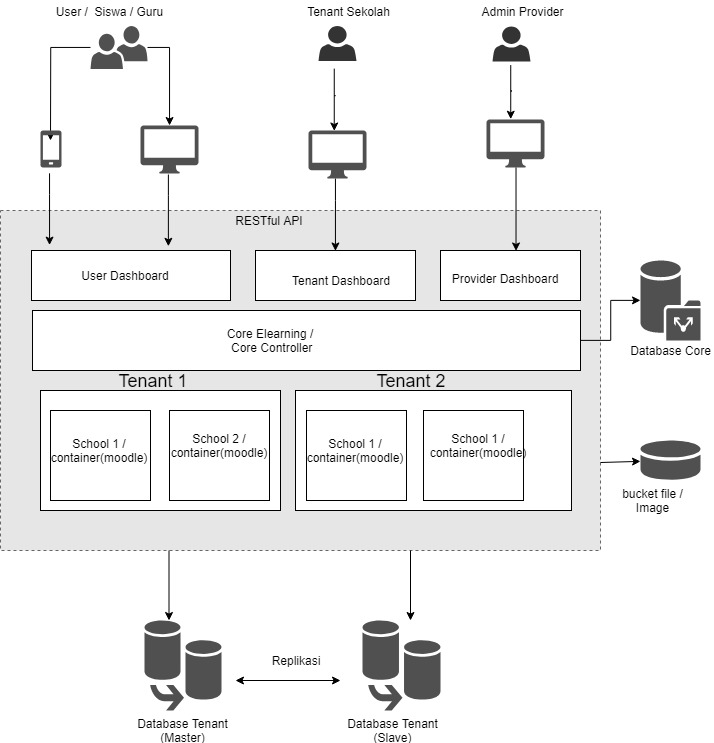
Gambar III.1 Kerangka pikir penelitian

Penelitian ini berfokus pada desain arsitektur REST di LMS Moodle di ekosistem komputasi awan. Prosesnya berupa merancang, mengevaluasi, dan memodifikasi strategi arsitektur untuk menangani tantangan bagaimana menyediakan sistem *E-Learning* yang terjangkau, *scalable* dan memiliki waktu *deployment* yang cepat untuk diterapkan di banyak lembaga pendidikan yang berbeda.

Untuk merancang arsitektur awal, akan digunakan implementasi LMS Moodle pada ekosistem komputasi awan di Google Cloud Platform (GCP). Kemudian masalah dirumuskan dan dipetakan ke pola desain SOA yang terinspirasi REST (Erl et al., 2012). Penelitian ini juga mendesain arsitektur dengan mengikuti batasan REST (Thomas Fielding, 2000). Hasil desain adalah arsitektur sistem *E-Learning* awal dan bagaimana strategi *deployment*nya. Evaluasi ATAM yang digunakan dalam penelitian ini adalah khusus untuk arsitektur REST (Costa et al. 2014, 2016). Evaluasi ini menggunakan skenario yang berkaitan dengan atribut kualitas yang diperoleh dari tinjauan literatur. Ini digunakan untuk meneliti standar REST, pertanyaan desain REST, pengujian berbasis skenario ATAM, dan *tradeoff* dalam keputusan arsitektur.

## Rancangan Solusi Arsitektur

Rancangan desain arsitektur awal yang diajukan adalah arsitektur *multi-tenant.* Server RESTful API (*backend*) dapat melayani banyak *tenant* yang berarti banyak sekolah di beberapa lembaga yang berbeda. *Web service* akan dipusatkan untuk melayani semua *client* Sistem *E-Learning*. Gambar III.2 akan memberikan tinjauan awal desain arsitektur yang diajukan.



Gambar III.2 Rancangan desain Arsitektur E-Learning Multitenant

Gambar III.2 menggambarkan dengan jelas bagaimana arsitektur awal arsitektur LMS untuk menangani *multitenancy*. User seperti guru dan siswa akan mengakses Sistem *E-Learning* melalui aplikasi web dan ponsel. Ada juga aplikasi untuk dashboard para partner atau tenant yang bisa memantau kontainer LMS Moodle yang *dideploy* di masing-masing lembaga atau yayasannya. Rancangan sistem *E-Learning* *multi-tenant* ini akan menerapkan metode MVC (*Model, View, Controller*) yang berisi dua bagian utama yakni core dan *tenant*. *Core controller* akan menangani logika bisnis bersama (model) yang digunakan untuk bersama seperti *login* dan manajemen user.

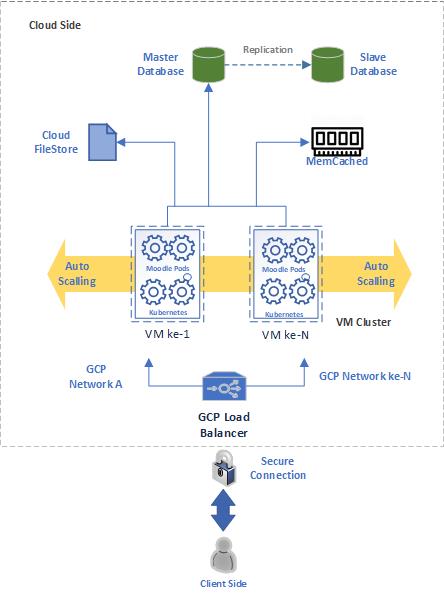
*Core controller* menentukan pengguna yang menjadi penyewa dalam setiap permintaan dengan menggunakan database sendiri yang sebagian besar berisi data umum *user* dan *tenant*. *Core controller* yang bertanggung jawab untuk mengelola request dan meneruskannya ke *controller* *tenant*, model dan database. Untuk proses *user login* dan otorisasi ke *core controller,* *web service* ini akan menggunakan RESTful API khusus untuk *login*.

Perhatian utama dalam arsitektur awal ini adalah menjaga mesin *web service* tetap *stateless*. *Stateless* berarti mesin layanan web tidak boleh mempertahankan atau menyimpan keadaan client. *Stateless* adalah pendekatan pertama untuk menangani *scalability*, dan otorisasi token adalah salah satunya. *Stateless* juga salah satu pondasi REST utama yang dijelaskan oleh Fielding (2000).

Menurut Ejsmont (2015), membuat mesin *stateless* memberikan keuntungan penting bagi *scalability*:

* Kita dapat mendistribusikan *traffic* permintaan ke aplikasi dengan menggunakan *load balancer*. Permintaan dapat didistribusikan secara merata ke semua mesin layanan web yang tersedia. Layanan web dapat memiliki banyak server dari area yang berbeda.
* Karena ada *load balancer* di depan mesin web service, maka kita dapat melakukan *scale out* dan *scale up* aplikasi. Scale up berarti mereplikasi aplikasi ke mesin layanan web lain yang berbeda. *Scale up* berarti meningkatkan daya komputasi mesin web service dengan meningkatkan CPU, RAM, dan lainnya.
* Umumnya *load balancer* sekarang dapat melakukan auto-*scalling*. Jika ada permintaan tinggi pada permintaan HTTP keseluruhan atau penggunaan CPU, *load balancer* akan secara otomatis menambah / mereplikasi mesin *web service*.
* Kita dapat melakukan *restart* atau mematikan mesin *web server* kapan saja tanpa memengaruhi pengalaman pengguna selama masih ada mesin server web lain yang beroperasi untuk melayani permintaan tersebut.
* Kita dapat melakukan *update* dan *maintenance* aplikasi dengan nol *downtime*. *Update* dapat dilakukan secara bergilir dan teratur antara setiap mesin layanan web. Kita juga dapat memiliki dua versi aplikasi yang berbeda secara bersamaan (*canary release*). Kita dapat menentukan berapa banyak pengguna acak yang mendapatkan versi aplikasi yang lebih baru. Layanan web *stateless* memberikan kemudahan dalam maintenance.
* Dengan membuat server web *stateless*, itu tidak hanya meningkatkan *scalability* dan *performance* tetapi juga ketersediaan. *Downtime* nol dan *maintenance* mudah adalah alasan mengapa *availability* dapat meningkat.

Semua manfaat dari *web service* yang *stateless* di atas membutuhkan stategi *deployment* yang tepat sehingga bisa mendukung *availability* yang tinggi. Strategi *deployment* yang akan digunakan pada penelitian ini digambarkan pada gambar III.3 sebagai berikut:



Gambar III.3 Gambaran strategi deployment sistem E-Learning di Cloud

Strategi yang akan dilakukan adalah dengan memasukkan sistem LMS Moodle dalam kontainer berupa Moodle Pod yang berisi aplikasi LMS Moodle dan *web server*. Moodle Pod ini berlaku sebagai mesin *web server* yang berada di dalam *Virtual Machine* (VM). Untuk menjaga agar moodle pods ini tetap *stateless* maka setiap penyimpanan file pendukung yang sifatnya dinamis dan bertambah seperti dokumen, gambar dan database dilakukan di luar Pod. Dalam hal ini kita akan menggunakan layanan *cloud* dari GCP untuk penyimpanan file berupa Cloud FileStore sedangkan untuk database layanannya bernama CloudSQL. Moodle pod ini diorkestrasi oleh Kubernetes *engine* yang akan mengatur *cluster* pods yang ada di VM dengan fitur *scale up* dan *scale out*.

Banyaknya *concurrent user* yang mengakses sistem *E-Learning* bisa mengakibatkan pemakaian *resource* CPU dan memory di dalam pod moodle menjadi *overload*, maka untuk menghindari hal tersebut kita bisa melakukan setting di Kubernetes *engine* jika resource CPU di Pod di atas 50% maka Kubernetes akan melakukan *scale up* pod dengan membuat pod baru sehingga dapat melayani banyaknya *request* dari user yang mengakses sistem E-*Learning*. Sebaliknya jika di level pod pemakain rata-rata CPU dan memory dibawah limit yang ditentukan misal di bawah 50%, maka kita bisa mengkonfigurasi sehingga sistem melakukan *scale down.*  Hal ini juga berlaku di level cluster VM, jika *resource* RAM dan CPU di cluster mencapai titik limit yang telah ditentukan, maka *cluster* akan secara otomatis *scale out (horizontal scalling)* dengan membuat VM baru ataupun sebaliknya melakukan *scale down* dengan menghapus kelebihan VM. Disinilah proses apa yang disebut a*uto-scalling* agar *availability* tetap terjaga dengan baik dan juga memberikan efisiensi terhadap biaya.

### REST Constraint

Arsitektur awal mengikuti REST *Constraint* yang dibuat oleh pendirinya, Fielding (Thomas Fielding, 2000). *Constraint* dasar dalam REST adalah *client-server*. Client membuat permintaan dan server merespons permintaan. Arsitektur ini memisahkan logika sisi *client* dan logika sisi *server*. Logika sisi *server* mengekspos *Application Programming Interface* (API) dan menggunakan spesifikasi HTTP sebagai protokol komunikasi.

Gaya arsitektur yang akan dirancang berdasarkan *constraint* REST adalah adalah penggunaan *uniform interface*. Ini berisi tiga elemen: metode, jenis media, dan sintaks pengidentifikasi *resource*. *Controller* sistem *E-Learning* ini akan menggunakan standar URI untuk menyatakan di mana data sedang ditransfer ke atau dari. Ini juga berupa URL karena kita dapat menggunakannya sebagai pengidentifikasi resource dan menerapkan metode di atasnya. Di bawah ini adalah sintaks umum URI yang akan digunakan dalam arsitektur awal (Erl et al., 2012):

*{scheme}://{authority}{path}?{query}*

Contoh URI yang menggunakan semua komponen adalah “*https://client.eduprovider.co/tenant/tenant-name/school/2?user=3”*

Tabel III.1 memperlihatkan perincian dari setiap komponen URI

Tabel III.1Sintaks umum URI

|  |  |
| --- | --- |
| **Bagian URI** | **Penggunaan** |
| *https* | *scheme/methods* |
| *client.eduprovider.co* | *Authority* |
| /tenant/tenant-name/ | Path yang menunjukkan informasi list sekolah yang dimiliki *tenant-name* tertentu |
| *2* | Identifier resource mengacu pada sekolah nomor 2 yang dimiliki *tenant-name* tertentu |
| *?user=3* | Query untuk melihat detail user nomor 3 pada sekolah nomor 2 |

Tabel III.1 menunjukkan penggunaan yang berbeda di setiap bagian URI dan fungsinya untuk mendukung arsitektur. URI yang sama dapat digunakan beberapa kali untuk melayani permintaan yang berbeda dengan menggunakan metode HTTP. URI di bawah ini menunjukkan metode HTTP yang akan dirancang untuk list sekolah di RESTful API. URI nya adalah:

*“https: //client.eduprovider.co/tenant/tenant-name".*

Tabel III.2 Penggunaan pola URI pada core controller

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **URI** | **Result** |
| GET | *https: //client.eduprovider.co/tenant/tenant-name* | Melihat semua daftar sekolah |
| POST | *https: //client.eduprovider.co/tenant/tenant-name* | Membuat list sekolah baru |
| PUT/  PATCH | *https: //client.eduprovider.co/tenant/tenant-name/3* | Update list sekolah dengan primary key 3 |
| DELETE | *https: //client.eduprovider.co/tenant/tenant-name/3* | Delete list sekolah dengan primary key 3 |

Tabel III.2 menunjukkan arti dari setiap metode HTTP di API *E-Learning* Controller. Identifier *“tenant”* adalah konstanta. Sedangkan identifier *“tenant-name”* adalah variable yang bisa berubah sesuai dengan nama *tenant* tertentu. *Identifier resource* yang digunakan adalah *primary key* dengan nilai 3 yang menunjukkan list sekolah nomor 3 yang dimiliki oleh *tenant-name* tertentu. Arsitektur ini sebagian besar akan menggunakan JSON untuk mengambil dan mentransfer data. Arsitektur ini menggunakan arsitektur sistem berlapis sebagai salah satu *REST constraint*.

### SOA Design Pattern

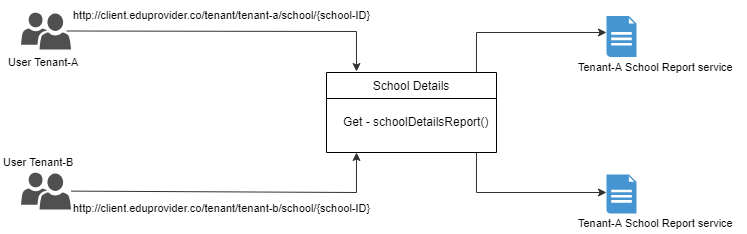
SOA *design* *pattern* yang diilhami REST akan diterapkan untuk penelitian ini. Pola desain ini sudah dioptimalkan untuk gaya arsitektur REST (Erl et al., 2012). Pola desain pertama yang digunakan adalah negosiasi konten dan terkait dengan *media type* dalam *uniform interface*. Beberapa *resource* memiliki beberapa presentasi untuk menyediakan data ke *client-side-logic* yang berbeda. Arsitektur ini menyesuaikan beberapa API agar mampu mengembalikan tidak hanya di JSON tetapi XML berdasarkan kebutuhan *client*. Implementasi *Accept* di *Header Request* adalah cara yang paling sering digunakan dan direkomendasikan untuk menentukan *media type.*

**Accept:** application/json, text/plain

Kemudian, *server* menggunakan HTTPS *Responses Header* untuk menentukan media types. Nilainya harus sama dengan *Accept* di *Request Header* client.

**Accept:** application/json, text/plain

*Design Pattern* berikutnya adalah *reusable contract*. Tujuan dari arsitektur ini adalah untuk melayani banyak *tenant* lembaga pendidikan dan memusatkan sistem. *Reusable contract* terkait dengan batasan kontrak yang seragam. Satu kontrak harus tersedia untuk semua *tenant*. Juga, masing-masing *tenant* memiliki banyak sekolah yang tersebar di lokasi yang berbeda. Harus ada kontrak yang juga dibagikan secara internal diantara masing-masing *tenant*.



Gambar III.4 SOA Design Pattern Core Controller LMS berupa Reusable Contract

Gambar III.4 menjelaskan bagaimana *Core Controller* *E-Learning* menangani beberapa *tenant* pendidikan dengan menggunakan *Reusable contract*. *Report* detail sekolah berbeda pada setiap client dan memerlukan layanan yang berbeda. Kedua *tenant* mengakses kontrak yang sama yaitu *GET - schoolDetailsReport().* Permintaan diteruskan ke layanan yang berbeda dengan kemampuan menampilkan laporan detail terkait sekolah tertentu yang dimiliki oleh *tenant* A dan *tenant* B. Satu-satunya yang membedakan permintaan adalah URL dasar. *Tenant* A menggunakan URL dasar yakni *http://client.eduprovider.co/tenant/tenant-a/* dan tenant B menggunakan URL dasar *http://client.eduprovider.co/tenant/tenant-b/*

Implementasi *reusable contract* ini selain diterapkan di *core controller* juga akan diterapkan pada *engine* LMS Moodle di setiap *tenant* sekolah. *Pattern* untuk menampilkan *report* nilai hasilpembelajaran di LMS moodle nantinya memiliki *pattern* URL sebagai berikut:

* *http://school1.tenant-a.edu/grade/report/user/index.php?id=3* untuk menampikan report nilai siswa nomor 3 di sekolah 1 pada *tenant* A.
* *http://school1.tenant-b.edu/grade/report/user/index.php?id=3* untuk menampikan report nilai siswa nomor 3 di sekolah 1 pada *tenant* B.

## Rencana Evaluasi

### Skenario dan Pertanyaan Desain

Dari rancangan arsitektur yang diajukan, maka rencana evaluasi penelitian ini akan berfokus apada *scalability* dan *performance* system. Tabel III.3 menunjukkan template untuk membuat skenario umum yang diperoleh berdasarkan penelitian yang dibuat oleh Costa (Costa et al. 2014, 2016).

Tabel III.3 Skenario Umum Atribut Kualitas

|  |  |  |  |
| --- | --- | --- | --- |
| **Atribut Kualitas (AK)** | **Skenario Umum AK** | **Rekomendasi untuk skenario kongkret** | **Pertanyaan Design** |
| *Scalability* | **SC1 -** Logic di sisi server terjamin *stateless* dan *scalable*/terukur | Apa yang membuat sistem *stateless* harus didefinisikan dengan jelas. | Mekanisme security apa untuk *service consumers* untuk melakukan akses pada *resources*? |
| **SC2** - Logic di sisi server melakukan penyimpanan data seperti dokumen, gambar, video dan data pembelajaran lainnya pada storage terpisah di luar mesin web server untuk mendukung skalabilitas ecara horizontal | Di mana data yang disimpan harus disebutkan. | Apa saja komponen lainnya yang bergantung pada layanan REST? |
| **SC3** - Website tenant dan layanan API dibuat dengan mulus dengan tanpa konfigurasi apapun di server. | Dalam skenario nyata, ketika klien ingin menggunakan sistem E-Learning, konfigurasi harus mudah. | Apa saja komponen lainnya yang bergantung pada layanan REST? |
| **SC4 –** Aplikasi REST dapat secara otomatis *scale up* dan *scale out* ketika ambang CPU mencapai tingkat tertentu. | Harus ditentukan di level berapa utilisasi CPU sehingga sistem dapat melakukan scale up dan scale out | Bagaimana layanan dikemas dan dipasang (*deploy*)? |
| *Performance* | **P1 –** Pembuatan tenant dapat dilakukan dengan mudah, cepat dan tidak memperlambat keseluruhan system | Harus ditentukan seberapa cepat (berapa detik). | Bagaimana layanan dikemas dan dipasang (*deploy*)? |
| **P2 –** Sistem dikunjungi oleh sejumlah tenant pada waktu yang terukur dan tidak mengurangi *user experiences.* | Harus ditentukan berapa banyak permintaan tenant yang datang ke server | Bagaimana cara melindungi server Web dari permintaan yang berlebihan? |
| **P3** – Sistem dikunjungi oleh sejumlah permintaan yang bersamaan pada waktu yang terukur dan tidak mengurangi *user experiences* | Harus ditentukan berapa banyak permintaan bersamaan yang datang ke server. | Apakah ada replikasi layanan REST saat *runtime*? |

Dari gambaran tabel III.3 di atas, intinya *performance* diperlukan untuk memastikan pembuatan *tenant* baru berjalan dengan cepat dan meningkatnya jumlah *tenant* tidak mengurangi *user experiences*. Adapun *scalability* diperlukan untuk memastikan sistem mampu *scale up* atau *scale down* ketika jumlah pengguna meningkat atau berkurang secara drastis.

Meskipun pada penelitian ini untuk evaluasi akan fokus ada kedua atribut kualitas di atas, namun tentunya atribut kualitas di atas memiliki keterkaitan dengan atibut kualitas lainnya. Sebagai contoh *scalability* terkait dengan *availability*. *Availability* berarti aplikasi harus tetap tersedia bagi pengguna setiap saat. Dari pengumpulan persyaratan, misal kita ingin sistem *E-Learning* dapat menangani 1000 *user* secara bersamaan dan memberikan kelonggaran penurunan waktu respons adalah 50%. Tabel 3.3 menunjukkan contoh skenario umum REST dalam *availability* .

Tabel III.4 Skenario Umum REST dalam availability

|  |  |  |  |
| --- | --- | --- | --- |
| **Atribut Kualitas** | **Skenario umum Atribut Kualitas** | **Rekomendasi untuk Skenario kongkret** | **Pertanyaan Disain** |
| *Availability* | **AV1** - Web server dimana layanan REST berjalan dibanjiri dengan sejumlah request N persen lebih tinggi dari normal dan services tetap responsif | Skenario konkret selanjutnya dapat menentukan penurunan yang diizinkan pada waktu respons. | 1. Komponen apa lagi yang bergantung pada layanan REST? |
| 1. Bagaimana cara melindungi server Web dari permintaan yang berlebihan? |
| **AV2 -** Kontainer Web tempat layanan REST berjalan dimatikan untuk pemeliharaan dan *service* tetap tersedia untuk melayani konsumen. | Skenario konkret dapat menunjukkan kondisi dan tindakan yang khusus untuk cluster atau teknologi grid yang akan digunakan untuk replikasi layanan. | 1. Komponen apa lagi yang bergantung pada layanan REST? |

### Skenario Kongkret

Skenario kongkret atau skenario riil diturunkan dari skenario atribut kualitas dan menjadi tolok ukur untuk hasil evaluasi. Tabel III.5 menggambarkan tentang langkah evaluasi dan hasil yang diharapkan.

Tabel III.5 Gambaran Skenario Kongret yang hendak dicapai

|  |  |
| --- | --- |
| **Attribut Kualitas** | **Skenario Kongkret** |
| *Scalability* (SC1) | Permintaan GET dikirim ke API materi pembelajaran (*Course*) di tenant ‘A’. API *Course* adalah *resource* yang dilindungi dan perlu otentikasi untuk mengakses. Permintaan harus mengandung token akses dalam bentuk JWT untuk diautentikasi dengan benar dan tidak ada status session yang disimpan di server atau database. |
| *Scalability* (SC2) | Permintaan GET dikirim ke API profil user di tenant ‘A’. Data gambar profil user berasal dari storage lain. Sistem tidak akan menyimpan data user pada storage yang sama dengan storage web server. |
| *Scalability* (SC3) | Tenant dibuat dan dikonfigurasi melalui panel admin tanpa memerlukan konfigurasi teknis terkait server. Setelah tenant dibuat, sistem secara otomatis menyebarkan basis data tenant, situs web (logika sisi-klien) dan API *endpoint* tenant (logika sisi-server). |
| *Scalability* (SC4) | API tingkat sekolah dalam tenant ‘A’ dikunjungi dengan sejumlah user secara bersamaan dalam satu menit. Jumlah *concurrent* user meningkat di setiap iterasi dan berhenti hingga mencapai 250 concurrent user. Kubernetes Pod akan diduplikasi jika total penggunaan CPU Pod mencapai 50%. Jika ukuran total Cluster tidak cukup untuk membuat duplikat Pod, Kubernetes Cluster akan melakukan *scale out* untuk meningkatkan daya komputasi. |
| *Performance* (P1) | Tenant dibuat dan dikonfigurasi melalui panel admin tanpa memerlukan konfigurasi teknis atau server. Tenant baru akan memiliki nama sekolah, logo, skema warna situs web, dan kredensial untuk admin untuk login dan memasukkan data sekolah. Tenant baru dibuat dan disebar dalam waktu kurang dari 5 menit. |
| *Performance* (P2) | 25 *concurrent user* mengakses sejumlah tenant secara acak dalam satu menit. Jumlah *tenant* meningkat di setiap pengulangan dan berhenti hingga mencapai 30 tenant. Availability sistem tetap 98% dan rata-rata waktu respons di bawah 2 detik. |
| *Performance* (P3) | API tingkat sekolah dikunjungi dengan sejumlah *concurrent* user dalam satu menit. Jumlah *concurrent* user meningkat di setiap iterasi dan berhenti hingga mencapai 250 concurrent user. Availability sistem tetap 98% dan rata-rata waktu respons di bawah 2 detik. |

Tabel III.5 di atas berisi standar minimum skenario kongret yang hendak dicapai untuk sistem *E-Learning* ini. Secara umum atribut kualitas *scalability* ingin menjaga agar sistem dapat mengatasi permintaan layanan dari user pada waktu yang memerlukan beban puncak seperti ketika waktu *deadline* pengumpulan tugas dan pengaksesan grading pada akhir semeseter. Selama waktu itu banyak pengguna secara bersamaan mengakses aplikasi sehingga sistem harus secara otomatis melakukan *scale up* agar performance layanan dapat terjaga dengan baik. Adapun ketika waktu beban menurun misal ketika liburan maka sistem akan *scale down* sehingga bisa menurunkan biaya.

### Output ATAM: Analisis Arsitektur

Evaluasi arsitektur menghasilkan output ATAM yang berupa hasil analitik dari skenario kongret. Analisis arsitektur adalah hasil evaluasi skenario kongkret. Berdasarkan skenario kongkret, adalah mungkin untuk menganalisis arsitektur, risiko, dan *tradeoff* dari berbagai atribut kualitas Contoh pertanyaan desain “Apa komponen lain yang bergantung pada layanan REST?" Dan "Bagaimana melindungi server web dari permintaan yang berlebihan?". Arsitektur harus menjawab kedua pertanyaan dalam analisis. Tabel III.6 menunjukkan hasil analitik dari contoh skenario kongkret. Dimungkinkan untuk menganalisis arsitektur tidak hanya berdasarkan atribut kualitas aslinya tetapi juga atribut kualitas lainnya.

Tabel III.6 Hasil analitik dari contoh skenario kongkret secara umum

|  |  |
| --- | --- |
| **Rangkuman Skenario** | 1000 user secara bersamaan (*concurrent*) melakukan pengisian forum diskusi dengan maksimum penurunan waktu respon adalah 50% |
| **Tujuan Bisnis** | 1000 user konkuren tidak terpengaruh sama sekali dalam hal user experience |
| **Atribut Kualitas** | *Availability, Performance, Scalability, Security* |
| **Analisis Arsitektur** | * Berdasarkan pertanyaan “Komponen apa yang bergantung pada layanan REST?”. Sistem ini sangat bergantung pada LoginAPI, CourseAPI, UploadAPI karena tanpaAPI tersebut laporan pengisian forum tidak dapat dihasilkan * Sistem akan menyelesaikan permintaan dalam 1 detik jika hanya diakses oleh 1 pengguna dengan upload file sebesar 100KB. Jadi toleransi maksimum adalah 1,5 detik.- Sistem LMS merespons dengan status HTPP 200. * Sistem masih mengelola untuk melayani 1000 pengguna bersamaan dengan waktu respons 1,5 detik. * Permintaan akan diantrikan oleh server web apache / nginx. |
| **Risiko** | Jika server web apache sedang down, maka perlu di-restart |
| **Tradeoffs** | Penggunaan microservices untuk sistem dapat meningkatkan performance sistem karena beban didistribusikan secara merata di setiap layanan microservices. Tetapi keamanan bisa berkurang karena mengandung lebih banyak layanan daripada arsitektur monolith (jumlah serangan potensial akan meningkat). |

Untuk membuat arsitektur mendukung horizontal scalling di Kubernetes, database dan data media lainnya harus ditempatkan keluar dari server ke penyimpanan lain. Tabel III.7 mengevaluasi penyimpanan dalam arsitektur untuk mendukung *horizontal scalling* berdasarkan skenario kongkret dalam *Scalability* (SC4) dan *Performace* (P3)

Tabel III.7 Contoh analitik dari contoh skenario kongkret SC2

|  |  |
| --- | --- |
| **Rangkuman Skenario** | Permintaan GET dikirim ke API materi pembelajaran (course) di tenant ‘A’. Data materi pembelajaran (misal file pdf, docx, pptx) berasal dari penyimpanan lain di luar web server. Sistem tidak akan menyimpan data pembelajaran pada penyimpanan yang sama dengan server. |
| **Tujuan Bisnis** | Membuat sistem arsitektur yang scalable |
| **Atribut Kualitas** | Scalability (SC2) |
| **Analisis Arsitektur** | Materi pembelajaran dikonfigurasi terpisah yakni berasal dari layanan Google Cloud Storage. Demikian juga database dikonfigurasi pada laynan Google Cloud SQL. |
| **Risiko** | Dapat meningkatkan biaya sewa penyimpanan di cloud. Konfigurasi yang tepat sangat penting dilakukan agar data sensitif tidak bisa diakses publik. |
| **Tradeoff** | Dengan memisahkan media penyimpanan, ini dapat meningkatkan scalability dan performance karena server tidak mengkloning data media ketika melakukan horizontal scalling (scale out). Namun ini menambha biaya untuk menyewa cloud storage dan ini harus dikonfigurasi denganbeanr untuk menghindari kebocoran data. |

Arsitektur harus terbukti dapat diskalakan dengan menerapkan otentikasi berbasis token dan penyimpanan eksternal. Interaksi yang stateless antara *client* dan *server* secara teoritis membuat sistem dapat melakukan *scale up* dan *scale out*.

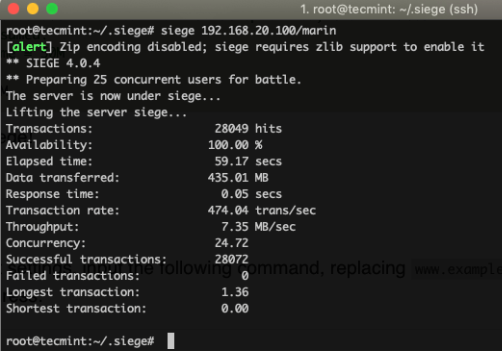
Karena arsitekturnya *multitenant*, kita harus mempertimbangkan penciptaan *tenant* baru. Untuk meningkatkan pengalaman pengguna, pembuatan tenant, konfigurasi, dan manajemen harus dilakukan melalui panel admin tanpa konfigurasi teknis di server.

Tabel III.8 Contoh hasil analitik skenario kongkret SC2 dan P1

|  |  |
| --- | --- |
| **Rangkuman Skenario** | Setelah tenant baru dibuat, sistem secara otomatis menggunakan database tenant, situs web (logika sisi-klien) dan API *endpoint* tenant (logika sisi-server). Tenant baru dibuat dan digunakan dalam waktu kurang dari 5 menit. |
| **Tujuan Bisnis** | Penempatan tenant mudah dan tidak mengurangi pengalaman pengguna |
| **Atribut Kualitas** | Scalability (SC2), Performance (P1) |
| **Analisis Arsitektur** | - Manajemen *tenant* dikelola melalui panel admin.  - Database tenant, situs web, dan titik akhir API yang digunakan secara instan dalam 15 detik  - Waktu penghapusan tenant tergantung pada data tenant |
| **Risiko** | Siapa pun yang memiliki akses yang valid untuk admin sistem dapat menggunakan dan menghapus tenant. |
| **Tradeoff** | *Tenant* dapat dibuat dengan mudah dan penempatan sistem LMS tenant dapat secara mudah dilakukan. Ini meningkatkan *performance* dan *scalability* daripada arsitektur tenant tunggal. Adapun jika proses ini tidak dilengkapi dengan verifikasi dua langkah, siapa pun yang memiliki akses dapat membuat dan menghapus *tenant* dan mengurangi *security*. |

Setelah persyaratan untuk membuat penskalaan horizontal terpenuhi, rencana berikutnya adalah mengevaluasi *scalability* arsitektur di Kubernetes. Skenario ini disebutkan dalam *Scalability* (SC4) dan *Performance* (P3). Untuk membuat skenario ini, spesifikasi Kubernetes akan ditentukan. Dalam skenario ini, Kubernetes berjalan di GCP dengan Google Kubernetes Engine (GKE). Nantinya Cluster berisi dua spesifikasi node atau disebut node pool. Spesifikasi node pool pertama adalah berupa dua Virtual Machine (VM) bertipe *n1-standard-1* dengan spesifikasi 1 vCPU dan 3,75 GB RAM. Kedua node pool melakukan *autoscaling* dengan ukuran minimum 2 VM dan maksimum 5 VM. Jadi spesifikasi minimum dari kumpulan node adalah 2 vCPU dengan 7.5 GB RAM. Pod *autoscaling* diatur ke utilisasi CPU 50% dengan Pod minimum 2 dan maksimum 50.

Dalam skenario ini, sejumlah *concurrent user* mencoba mengakses API LMS setiap detik selama periode satu menit. Jumlahnya dimulai dari 25 pengguna bersamaan dan meningkat di setiap iterasi hingga mencapai 250 pengguna bersamaan. *Software tester* Siege digunakan untuk membuat *stress testing* ke server dengan jumlah *concurrent user* yang telah ditentukan. Pengujian mengakses *resource* API LMS dengan tipe konten JSON. Jumlah *delay* dalam setiap permintaan adalah antara 0 dan 1 detik. Delay ini memungkinkan transaksi untuk dilakukan secara bertahap daripada dilakukan secara sekaligus. Gambar adalah tampilan proses *testing* dengan software Siege.



Gambar III.5 Tampilan proses stress test pada software Siege

Adapun dari hasil stress test ini akan masukkan ke dalam Tabel III.9 sebagai berikut

Tabel III.9 Tabel evaluasi *stress test* aspek scalability cluster E-Learning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Concurrency**  **(user/detik)** | **vCPU/RAM (unit/GB)** | **Replika Pod** | **Availability**  **(%)** | **Average Response Time (detik)** |
| 25 | 2/7.5 | 4 |  |  |
| 50 | 2/7.5 |  |  |  |
| 75 | 2/7.5 |  |  |  |
| 100 | 3/11.25 |  |  |  |
| 125 | 3/11.25 |  |  |  |
| 150 | 3/11.25 |  |  |  |
| 175 | 4/15 |  |  |  |
| 200 | 4/15 |  |  |  |
| 225 | 5/18.75 |  |  |  |
| 250 | 5/18.75 |  |  |  |

Dari *stress test* di atas kita akan mengetahui di titik *concurency* berapa sistem *cluster* Kubernetes akan secara otomatis membuat replika pod baru beserta persentase *availability* dan rata waktu respons. Dari evaluasi ini, kita dapat melihat apakah jumlah *concurrent user* bisa meningkatkan latensi. Hasil evaluasi dari *stress test* di atas kemudian akan digunakan untuk output ATAM lainnya berupa hasil analitik skenario kongkret terkait dengan *Scalability* (SC4) dan *Performace* (P3).

Evaluasi selanjutnya adalah mengevaluasi kemampuan arsitektur dalam menangani beberapa *tenant* sekaligus. Skenario kongret dijelaskan dalam Performance (P2). 25 *concurrent user* mencoba untuk mencapai jumlah tenant URI yang berbeda setiap detik selama periode satu menit. Spesifikasi c*luster* Kubernetes sama dengan pengujian *scalability*. *Autoscaling* akan diatur ke replika Pods minimum 10 dan maksimum 50 Pods dengan ambang batas CPU diatur ke 50%. Tenant URI adalah berbeda di setiap iterasi, tetapi *resource* tetap sama yaitu *Core Controller* API. Jumlahnya mulai dari 3 *tenant* dan meningkat di setiap iterasi hingga mencapai 30 *tenant*. Rancangan evaluasi pada level *core controller* ini ada pada table III.10

Tabel III.10 Rencana tabel evaluasi untuk performance multitenant

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Jumlah Tenant** | **Transaksi Total** | **Availability** | **Average Response**  **Time (detik)** | **Average Transaction Rate**  **(transaction/detik)** | **Longest**  **Transaction**  **(detik)** | **Shortest**  **Transaction**  **(detik)** |
| 3 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |
| 21 |  |  |  |  |  |  |
| 24 |  |  |  |  |  |  |
| 27 |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |
| **Average** |  |  |  |  |  |  |

Dari hasil evaluasi di tabel III.10 kemudian akan digunakan untuk output ATAM berupa hasil analitik skenario kongkret terkait Performance (P2), dimana dari sini kita bisa mengetahui sejauh mana core controller E-*Learning* bisa menangani *concurrent user* dari *tenant* yang berbeda.

## Rencana Kegiatan

Tabel III.11 berikut merupakan tabel *timeline* rencana kegiatan yang akan dilakukan dalam penelitian ini.

Tabel III.11 Rencana Kegiatan Penelitian

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No | Kegiatan | Bulan | | | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | Studi Literatur |  |  |  |  |  |  |  |  |
| 2 | Latar Belakang dan Rumusan Masalah |  |  |  |  |  |  |  |  |
| 3 | Menentukan Tujuan dan Ruang Lingkup |  |  |  |  |  |  |  |  |
| 4 | Desain, Analisis, Evaluasi dan Optimalisasi Arsitektur Sistem E-Learning di komputasi awan. |  |  |  |  |  |  |  |  |
| 5 | Evaluasi Hasil |  |  |  |  |  |  |  |  |
| 7 | Penulisan Tesis |  |  |  |  |  |  |  |  |

# DAFTAR PUSTAKA

Adamczyk, P., Smith, P. H., Johnson, R. E., & Hafiz, M. (2011). REST and Web Service: In Theory and in Practice. In *REST: From Research to Practice*. Springer Science+Business Media, LLC.

Al-Ajlan, A., & Zedan, H. (2007). E-Learning (MOODLE) Based on Service Oriented Architecture. *Proceeding of the EADTU’s 20th Anniversary Conference*.

Al-Khanjari, Z., Al-Kindi, Z., Al-Kindi, E., & Kraiem, N. (2015). Developing educational mobile application architecture using SOA. *International Journal of Multimedia and Ubiquitous Engineering*, *10*(9), 247–254. https://doi.org/10.14257/ijmue.2015.10.9.25

Al-Khanjari, Z., Al-Roshdi, Y., & Kraiem, N. (2014). Developing virtual lab to support the computer science education in moodle. *International Journal of Software Engineering an Its Applications*. https://doi.org/10.1109/REV.2015.7087290

Barbacci, M., Clements, P., Lattanze, A., Northrop, L., & Wood, W. (2003). *Using the Architecture Tradeoff Analysis Method SM (ATAM SM ) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study*. Retrieved from http://www.sei.cmu.edu/publications/pubweb.html

Bass, L., Klein, M., & Moreno, G. (2001). *2001 Applicability of General Scenarios to ATAM*. Pittsburgh, PA.

Bianco, P., Kotermansk, R., & Merson, P. (2007). *Evaluating a Service-Oriented Architecture*.

Chen, M., Ma, Y., Liu, Y., Jia, F., Ran, Y., & Wang, J. (2013). Mobile learning system based on cloud computing. *Journal of Networks*, *8*(11), 2572–2577. https://doi.org/10.4304/jnw.8.11.2572-2577

Costa, B., Pires, P. F., Delicato, F. C., & Merson, P. (2014). Evaluating a Representational State Transfer (REST) architecture: What is the impact of REST in my architecture? *Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014*, 105–114. https://doi.org/10.1109/WICSA.2014.29

Costa, B., Pires, P. F., Delicato, F. C., & Merson, P. (2016). Evaluating REST architectures - Approach, tooling and guidelines. *Journal of Systems and Software*, *112*, 156–180. https://doi.org/10.1016/j.jss.2015.09.039

Darin E. Hartley. (2001). *Selling E-Learning*. American Society for Training and Development.

Davis, B., Carmean, C., & Wagner, E. D. (2009). *The Evolution of the LMS: From Management to Learning*.

Debashis De. (2016). *Mobile Cloud Computing: Architectures, Algorithms and Applications*. CRC Press.

Dvorak, R. (2011). *Moodle for Dummies*. New Jersey: Wiley Publishing, Inc.

Ejsmont, A. (2015). *Web Scalability for Startup Engineers*. McGraw Hill Education.

Erl, T. (2009). *SOA Design Patterns*. Prentice Hall.

Erl, T., Bennett, S. G., Gee, C., Laird, R., Manes, A. T., Schneider, R., … Venable, C. (2011). *SOA Governance\_ Governing Shared Services On-Premise and in the Cloud - Thomas Erl - (2011)*. Boston: Prentice Hall.

Erl, T., Carlyle, B., Pautasso, C., & Balasubramanian, R. (2012). *SOA with REST. Principles, Pattern & Constraints for Building Enterprise Solutions with REST*. New Jersey: Prentice Hall.

Fielding, R. T., Taylor, R. N., Erenkrantz, J. R., Gorlick, M. M., Whitehead, J., Khare, R., & Oreizy, P. (2017, August 2). *Reflections on the REST architectural style and “principled design of the modern web architecture” (impact paper award)*. 4–14. https://doi.org/10.1145/3106237.3121282

Garriga, M., Mateos, C., Flores, A., Cechich, A., & Zunino, A. (2016). RESTful service composition at a glance: A survey. *Journal of Network and Computer Applications*, *60*, 32–53. https://doi.org/10.1016/j.jnca.2015.11.020

Gorton, I. (2006). *Essential Software Architecture*. Berlin Heidelberg: Springer-Verlag.

Graham, I. (2008). *Requirements Modelling and Specification for Service Oriented Architecture*. John Wiley and Sons Inc.

Huertas, F., & Navarro, A. (2015). SOA support to virtual campus advanced architectures: The VCAA canonical interfaces. *Computer Standards and Interfaces*, *40*, 1–14. https://doi.org/10.1016/j.csi.2014.11.001

Internet World Stats. (2019). Retrieved April 9, 2019, from https://www.internetworldstats.com/top20.htm

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). *The Architecture Tradeoff Analysis Method*.

Khan, M. W., & Abbasi, E. (2015). Differentiating Parameters for Selecting Simple Object Access Protocol (SOAP) vs. Representational State Transfer (REST) Based Architecture. *Journal of Advances in Computer Networks*, *3*(1), 63–66. https://doi.org/10.7763/jacn.2015.v3.143

Kitanov, S., & Davcev, D. (2012). Mobile Learning in Mobile Cloud Computing Environment. *International Transactions on Systems Science and Applications*, *8*(December), 27–39.

Moodle Statistics. (2019). Retrieved April 9, 2019, from https://moodle.net/stats/

Nash, S. S., & Rice, W. (2018). *Moodle 3 E-Learning Course Development* (4th ed.). Packt Publishing Ltd.

Rao, N. M., C.Sasidhar, & Kumar, V. S. (2010). Cloud Computing Through Mobile-Learning. *International Journal of Advanced Computer Science and Applications*, *1*(6). https://doi.org/10.14569/ijacsa.2010.010607

Thomas Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine.

Zhang D. (2004). *Powering E-Learning In The New Millenium: An Overview of E-Learning and Enabling Technology, Information System Frontiers*. *5*(2), 201–212.