

**RANCANG BANGUN SISTEM *E-LEARNING* BERBASIS
MICROSERVICES DAN *DOMAIN DRIVEN DESIGN*
(STUDI KASUS PROBISTEK UIN MAULANA
MALIK IBRAHIM MALANG)**

SKRIPSI

Oleh:
LIA KHOIRUNNISA'
NIM. 14650045



**JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2019**

**RANCANG BANGUN SISTEM *E-LEARNING* BERBASIS *MICROSERVICES*
DAN *DOMAIN DRIVEN DESIGN* (STUDI KASUS PROBISTEK UIN
MAULANA MALIK IBRAHIM MALANG)**

SKRIPSI

**Diajukan kepada:
Fakultas Sains dan Teknologi
Universitas Islam Negeri (UIN)
Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)**

**Oleh:
LIA KHOIRUNNISA'
NIM. 14650045**

**JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2019**

LEMBAR PERSETUJUAN**RANCANG BANGUN SISTEM *E-LEARNING* BERBASIS *MICROSERVICES*
DAN *DOMAIN DRIVEN DESIGN* (STUDI KASUS PROBISTEK UIN
MAULANA MALIK IBRAHIM MALANG)****SKRIPSI**

Oleh :

LIA KHOIRUNNISA'
NIM. 14650045

Telah Diperiksa dan Disetujui untuk Diuji

Tanggal : 21 Mei 2019

Dosen Pembimbing I

Dr. Suhartono, M.Kom
NIP. 19680519 200312 1 001

Dosen Pembimbing II

M. Ainul Yaqin, M.Kom
NIP. 19761013 200604 1 004

Mengetahui,

Ketua Jurusan Teknik Informatika

Fakultas Sains dan Teknologi

Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian

NIP. 19740424 200901 1 008

LEMBAR PENGESAHAN

**RANCANG BANGUN SISTEM *E-LEARNING* BERBASIS
MICROSERVICES DAN DOMAIN DRIVEN DESIGN (STUDI KASUS
PROBISTEK UIN MAULANA MALIK IBRAHIM MALANG)**

SKRIPSI

Oleh :

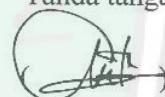
LIA KHOIRUNNISA'
NIM. 14650045

Telah Dipertahankan di Depan Dewan Penguji
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Pada Tanggal Mei 2019

Susunan Dewan Penguji

- | | | |
|-----------------------|---|--|
| 1. Penguji Utama | : | <u>Ajib Hanani, M.T</u>
NIDT. 19840731 20160801 1 076 |
| 2. Ketua Penguji | : | <u>Supriyono, M.Kom</u>
NIDT. 19841010 20160801 1 078 |
| 3. Sekretaris Penguji | : | <u>Dr. Suhartono, M.Kom</u>
NIP. 19680519 200312 1 001 |
| 4. Anggota Penguji | : | <u>M. Ainul Yaqin, M.Kom</u>
NIP. 19761013 200604 1 004 |

Tanda tangan






Mengetahui,

Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang



Dr. Cinyo Crysdian
NIP. 19740424 200901 1 008

HALAMAN MOTTO

"Mimpi tidak dilahirkan saat kita tidur. Tetapi mimpi adalah sesuatu yang membuat kita tidak bisa tidur."

"Tentukan pilihan hidupmu, sebelum pilihan akan menentukan hidupmu"

HALAMAN PERSEMBAHAN

الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ

Saya persembahkan karya ini dengan segala kerendahan hati, semoga skripsi ini bermanfaat dunia dan akhirat. Aamiin.

Dengan segala hormat, saya persembahkan karya ini kepada :

Kedua orang tua saya Bapak Mugianto dan Ibu Insi Mardliyah, yang telah mendidik dengan penuh kasih sayang dan memberikan dukungan serta do'a untuk menuntut ilmu setinggi-tingginya.

Guru diskusi saya Muhammad H. Syaifulloh yang telah meluangkan waktu, tenaga dan pikiran demi terselesainya skripsi ini.

Keluarga Teknik Informatika kelas B 2014, keluarga Biner (Teknik Informatika angkatan 2014), serta seluruh keluarga besar Teknik Informatika UIN Maulana Malik Ibrahim Malang

Teman-teman seperjuangan Mahfud, Yuli yang memberikan kritik, saran dan motivasi dalam pembuatan skripsi ini. Orang-orang yang saya sayangi, yang tak bisa saya sebutkan satu per satu.

Saya ucapkan terimakasih yang luar biasa. Semoga ukhwah kita tetap terjaga dan selalu diridhoi Allah SWT. Allahumma Aamiin.



PERNYATAAN KEASLIAN TULISAN

Saya yang bertandatangan di bawah ini:

Nama : Lia Khoirunnisa'
 NIM : 14650045
 Fakultas/ Jurusan : Sains dan Teknologi/ Teknik Informatika
 Judul Skripsi : **Rancang Bangun Sistem *e-Learning* Berbasis *Microservices* dan *Domain Driven Design* (Studi Kasus Prohistek UIN Maulana Malik Ibrahim Malang).**

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-nenar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 21 Mei 2019



Penulis

Lia Khoirunnisa'
 NIM. 14650045

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Segala puji bagi Allah SWT, karena atas rahmat, hidayah serta karunia-Nya, penulis dapat menyelesaikan skripsi yang berjudul “Rancang Bangun Sistem *e-Learning* Berbasis *Microservices* dan *Domain Driven Design* (Studi Kasus Prodistek UIN Maulana Malik Ibrahim Malang)” sebagai salah satu syarat untuk memperoleh gelar sarjana pada Program Studi Teknik Informatika jenjang Strata-1 Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Salawat serta salam senantiasa terlimpahkan kepada Nabi Muhammad SAW, keluarga dan para sahabat yang telah membimbing umat dari gelapnya alam jahiliyah menuju cahaya islam yang diridoi Allah SWT.

Penulis menyadari adanya banyak keterbatasan yang penulis miliki, sehingga ada banyak pihak yang telah memberikan bantuan baik moril, materiil, nasihat dan semangat dalam menyelesaikan penelitian ini. Maka dari itu dengan segenap kerendahan hati penulis mengucapkan terimakasih kepada:

1. Prof Dr H Abd. Haris, M.Ag selaku rektor UIN Maulana Malik Ibrahim Malang.
2. Dr. Sri Harini, M.Si. selaku Dekan Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
3. Dr. Cahyo Crysdiyan selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

4. Dr. Suhartono, M.Kom dan M. Ainul Yaqin, M.Kom, selaku pembimbing yang senantiasa meluangkan waktu untuk membimbing, mengarahkan penulis, dan memberi masukan.
5. Ajib Hanani, M.T dan Supriyono, M.Kom selaku penguji yang telah membimbing dan memberikan masukan pada skripsi ini.
6. Seluruh Dosen Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang yang telah memberikan ilmu dan pengetahuan serta pengalaman.
7. Segenap civitas akademik Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
8. Kedua orang tua serta seluruh keluarga besar penulis yang senantiasa mendukung.
9. Sahabat-sahabat seperjuangan Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

Penulis menyadari dalam karya ini masih banyak kekurangan. Oleh karena itu penulis selalu menerima segala kritik dan saran dari pembaca. Semoga karya ini bermanfaat bagi seluruh pihak.

Malang, 21 Mei 2019

Penulis

DAFTAR ISI

HALAMAN COVER	ii
LEMBAR PERSETUJUAN	iii
LEMBAR PENGESAHAN	iv
HALAMAN MOTTO	v
HALAMAN PERSEMBAHAN	vi
PERNYATAAN KEASLIAN TULISAN	viii
KATA PENGANTAR.....	ix
DAFTAR ISI.....	xi
DAFTAR GAMBAR.....	xiii
DAFTAR TABEL	xvi
ABSTRAK	xvii
ABSTRACT	xviii
المخلص.....	xix
BAB 1 PENDAHULUAN	1
1.1 Latar Belakang.....	1
1.2 Identifikasi Masalah.....	6
1.3 Tujuan Penelitian	6
1.4 Manfaat Penelitian	6
1.5 Batasan Masalah	7
BAB II KAJIAN PUSTAKA.....	8
2.1 Penelitian Terkait	8
2.2 <i>E-Learning</i>	10
2.2.1 Konsep <i>E-Learning</i>	11
2.2.2 Komponen <i>E-Learning</i>	12
2.3 <i>Microservices Architechture</i>	13
2.3.1 Pengertian <i>Microservices Architecture</i>	13
2.3.2 Aspek dalam <i>Microservices</i>	15
2.3.3 Keuntungan <i>Microservices</i>	17
2.4 <i>API Gateway</i>	20
2.5 <i>RESTful API / REST API</i>	20
2.5.1 Pengertian <i>RESTful API</i>	26
2.5.2 Komponen <i>RESTful API</i>	26
2.5.3 Cara Kerja <i>RESTful API</i>	26

2.6	<i>Domain Driven Design</i>	25
2.6.1	Pengertian <i>Domain</i>	26
2.6.2	Pengertian <i>Domain Driven Design</i>	26
2.6.3	<i>Subdomain</i> dan <i>Bounded Context</i>	26
2.7	<i>NoSQL</i>	28
2.8	<i>MongoDB</i>	29
BAB III ANALISIS DAN PERANCANGAN SISTEM		30
3.1	Analisis Sistem	30
3.1.1	Gambaran Umum Sistem	26
3.1.2	Prosedur Penelitian	32
3.2	Pemodelan Sistem	35
3.2.1	<i>Domain Driven Design</i>	36
3.2.2	<i>Microservices Architecture</i>	38
3.3	Analisis Proses Bisnis	41
3.3.1	Pengertian <i>Microservices Architecture</i>	41
3.3.1.1	Analisis Proses Bisnis	41
3.3.2	Identifikasi dan Analisis Kebutuhan	44
3.4	Desain Sistem	48
3.4.1	Pemodelan Sistem	48
3.4.2	Desain <i>Interface</i>	56
BAB IV HASIL DAN PEMBAHASAN		63
4.1	Implementasi Sistem	63
4.2	Implementasi <i>Microservices</i> dan <i>Domain Driven Design</i>	64
4.3	Implementasi Antarmuka Aplikasi	67
4.4	Pengujian Sistem	81
4.4.1	Pengujian <i>Multiple Servers</i>	81
4.4.2	Pengujian Fungsionalitas	87
4.4.3	<i>Unit Test</i> dan <i>Code Coverage</i>	89
4.4.4	<i>Integration Test</i>	91
BAB V PENUTUP		95
5.1.	Kesimpulan	95
5.2.	Saran	96
DAFTAR PUSTAKA		97

DAFTAR GAMBAR

Gambar 2.1 <i>Microservices</i>	14
Gambar 2.2 <i>API Gateway Pattern</i>	15
Gambar 2.3 Contoh <i>Polyglot Persistence</i>	18
Gambar 2.4 <i>REST Response</i> dan <i>REST Request</i>	20
Gambar 2.5 <i>Domain, Subdomain</i> dan <i>Bounded Context</i>	28
Gambar 3.1 <i>Flowchart</i> Prosedur Penelitian	32
Gambar 3.2 Sistem <i>e-Learning</i> Probistek	35
Gambar 3.3 Desain <i>DDD</i>	37
Gambar 3.4 <i>auth-service</i>	39
Gambar 3.5 <i>class-service</i>	39
Gambar 3.6 <i>test-service</i>	40
Gambar 3.7 <i>material-service</i>	40
Gambar 3.8 <i>Context Diagram</i>	48
Gambar 3.9 <i>DFD Level 1</i>	49
Gambar 3.10 <i>DFD Level 2</i>	50
Gambar 3.11 <i>CDM e-Learning</i>	51
Gambar 3.12 <i>PDM e-Learning</i>	52
Gambar 3.13 <i>BPMN Login</i>	52
Gambar 3.14 <i>BPMN create user</i>	53
Gambar 3.15 <i>BPMN create class</i>	53
Gambar 3.16 <i>BPMN update class</i>	53
Gambar 3.17 <i>BPMN delete class</i>	54
Gambar 3.18 <i>BPMN create material</i>	54
Gambar 3.19 <i>BPMN update material</i>	54
Gambar 3.20 <i>BPMN delete material</i>	55
Gambar 3.21 <i>BPMN create test</i>	55
Gambar 3.22 <i>BPMN update test</i>	55
Gambar 3.23 <i>BPMN delete test</i>	56
Gambar 3.24 <i>BPMN doing test</i>	56
Gambar 3.25 <i>Form Login</i>	57
Gambar 3.26 <i>Form Registrasi</i>	57
Gambar 3.27 <i>Form Program Keahlian</i>	57

Gambar 3.28 <i>Form Materi</i>	58
Gambar 3.29 <i>Form Assignment</i>	58
Gambar 3.30 <i>Form Soal</i>	58
Gambar 3.31 <i>List Soal</i>	59
Gambar 3.32 Data Program Keahlian <i>Admin</i>	59
Gambar 3.33 Data Program Keahlian <i>Teacher</i>	60
Gambar 3.34 Data Program Keahlian <i>Student</i>	60
Gambar 3.35 Detail Program Keahlian	60
Gambar 3.36 Data Materi Admin	61
Gambar 3.37 Data Materi.....	61
Gambar 3.38 Data Soal Admin	61
Gambar 3.39 Data Soal	62
Gambar 3.40 <i>Test</i>	62
Gambar 4.1 <i>structure auth-service</i>	64
Gambar 4.2 Autentikasi <i>API Gateway</i>	65
Gambar 4.3 Struktur <i>class-service</i>	65
Gambar 4.4 Komunikasi <i>class-service</i> ke <i>auth-service</i>	66
Gambar 4.5 Struktur Kode <i>material-service</i>	66
Gambar 4.6 Struktur Kode <i>test-service</i>	67
Gambar 4.7 Halaman <i>Login</i>	68
Gambar 4.8 Autentikasi <i>Login</i>	69
Gambar 4.9 <i>Generate Token</i>	69
Gambar 4.10 Halaman <i>Logout</i>	70
Gambar 4.11 Halaman Tambah Siswa.....	70
Gambar 4.12 Halaman List Siswa.....	71
Gambar 4.13 Halaman Tambah Guru	71
Gambar 4.14 Halaman List Guru	72
Gambar 4.15 Halaman Tambah Kelas	73
Gambar 4.16 Halaman List Kelas	73
Gambar 4.17 Halaman Edit Kelas.....	73
Gambar 4.18 Halaman Detail Kelas.....	74
Gambar 4.19 Halaman Kelas Pengajar	74
Gambar 4.20 Halaman Kelas Siswa.....	75
Gambar 4.21 Halaman Join Kelas Siswa	75

Gambar 4.22 Halaman Tambah <i>Assignment</i>	76
Gambar 4.23 Halaman List <i>Assignment</i>	76
Gambar 4.24 Halaman Edit <i>Assignment</i>	77
Gambar 4.25 Halaman Tambah Soal	77
Gambar 4.26 Halaman List Soal	78
Gambar 4.27 Halaman Detail Soal	78
Gambar 4.28 Halaman Edit Soal	79
Gambar 4.29 Halaman Hapus Soal	79
Gambar 4.30 Halaman <i>Assignment</i>	80
Gambar 4.31 Halaman Test Siswa	80
Gambar 4.32 Halaman Lihat Profil	80
Gambar 4.33 <i>List Server</i>	81
Gambar 4.34 Konfigurasi <i>Server Class</i>	82
Gambar 4.35 Ketiga <i>Server</i> dengan Kondisi <i>Online</i>	82
Gambar 4.36 <i>class-service</i> pada <i>Server</i> Pertama Kondisi <i>Offline</i>	83
Gambar 4.37 <i>class-service</i> pada <i>Server</i> Kedua Kondisi <i>Offline</i>	83
Gambar 4.38 <i>class-service</i> pada <i>Server</i> Ketiga Kondisi <i>Offline</i>	84
Gambar 4.39 <i>class-service</i> pada <i>Server</i> 1 dan 2 Kondisi <i>Offline</i>	85
Gambar 4.40 <i>class-service</i> pada <i>Server</i> 1 dan 3 Kondisi <i>Offline</i>	85
Gambar 4.41 <i>class-service</i> pada <i>Server</i> 2 dan 3 Kondisi <i>Offline</i>	86
Gambar 4.42 Layanan <i>class-service</i> dapat diakses	86
Gambar 4.43 Grafik Uji Fungsionalitas <i>Role Administrator</i>	87
Gambar 4.44 Grafik Uji Fungsionalitas <i>Role Teacher</i>	88
Gambar 4.45 Grafik Uji Fungsionalitas <i>Role Student</i>	88
Gambar 4.46 <i>unit test</i>	90
Gambar 4.47 <i>code coverage</i>	90
Gambar 4.48 Integrasi <i>auth-service</i> ke <i>class-service</i>	91
Gambar 4.49 Integrasi <i>auth-service</i> ke <i>material-service</i>	92
Gambar 4.50 Integrasi <i>auth-service</i> ke <i>test-service</i>	93
Gambar 4.51 Integrasi <i>material-service</i> ke <i>class-service</i>	94

DAFTAR TABEL

Tabel 2.1 Terminologi *SQL* dan *MongoDB*29

Tabel 3.1 Analisis Proses Bisnis43

Tabel 3.2 Data Siswa.....44

Tabel 3.3 Data Guru44

Tabel 3.4 Data Program Keahlian.....44

Tabel 3.5 Data Materi45

Tabel 3.6 Data Test45

Tabel 3.7 Identifikasi dan Analisis Kebutuhan Non-fungsional.....46

ABSTRAK

Khoirunnisa', Lia. 2019. **Rancang Bangun Sistem E-Learning Berbasis Microservices dan Domain Driven Design (Studi Kasus Probistek UIN Maulana Malik Ibrahim Malang)**. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Pembimbing: (I) Dr. Suhartono, M.Kom
(II) M. Ainul Yaqin, M.Kom

Kata kunci: *Microservices, Bounded Context, DDD, e-learning.*

Microservices merupakan konsep arsitektur perangkat lunak dimana sistem terdiri dari layanan-layanan kecil yang saling bekerjasama dan bersifat otonom, *deployable*, *scalable* yang dimodelkan berdasarkan *bounded context*. *Microservices* mengatasi masalah terfokus pada layanan yang memerlukan penanganan, tanpa mempengaruhi layanan yang lain. Dalam implementasi *microservices* menggunakan *bounded context* dari *Domain Driven Design (DDD)* untuk menentukan batasan layanan. Hasil dari pembagian sistem dengan *bounded context* adalah 4 services yaitu *auth-service* (sebagai *API gateway*), *class-service*, *material-service* dan *test-service* yang masing-masing dibangun dengan *Node.js*. Sedangkan untuk penyimpanan data menggunakan DBMS *MySQL* dan *MongoDB*. Masing-masing layanan berkomunikasi dengan menggunakan *REST*. Pengujian dilakukan dengan *multiple servers* yang menggunakan algoritma *round robin* untuk mendapatkan sistem yang *scalable* dan *resilient*. *Functional testing* menguji fungsional perangkat lunak menggunakan *balckbox* dengan hasil semua fitur dapat berjalan sesuai fungsinya. *Integration test* menggunakan *Postman* untuk menguji integrasi antar *services* dengan hasil antar *service* dapat berkomunikasi dan melakukan pertukaran data. *Unit test* dan *code coverage* yang terdiri dari *positive test* dan *negative test* untuk menguji aplikasi berjalan sesuai dengan skenario *test* yang telah dibuat.

ABSTRACT

Khoirunnisa', Lia. 2019. **Designing e-Learning Systems based Microservices and Domain Driven Design (Case Study of Probistek UIN Maulana Malik Ibrahim Malang)**. Informatics Department of Science and Technology Faculty. The State Islamic University Maulana Malik Ibrahim Malang.

Promotor: (I) Dr. Suhartono, M.Kom

(II) M. Ainul Yaqin, M.Kom

Keyword: *Microservices, Bounded Context, DDD, e-learning.*

Microservices is a software architecture concept where the system consists of small services that work together and are autonomous, deployable, and scalable which is modeled based on the bounded context. Microservices overcome problems focused on services that require handling, without affecting other kites. In microservices implementation use the bounded context of Domain Driven Design (DDD) to determine services limits. The result of dividing the bounded context system are 4 services, namely auth-service (as an API gateway), class-service, materials-services and test services, each of which is build with Node.js. While for storing data using MySQL and MongoDB DBMS. Each service communicates using REST. Testing has done with multiple servers that use round robin algorithms to get a system that is scalable and resilient. Functional testing tests the functional software using black-box with the results that all features can function according to their functions. The integration test uses Postman to test the integration between services with the test result that the service can communicate and exchange data. Unit tests and code coverage consisting of positive tests and negative tests to test applications run according to the test scenario that has been made.

الملخص

خير النساء، ليا. 2019. تصميم أنظمة التعليم الإلكتروني على أساس الخدمات الصغرى والتصميم الموجة بالمجال (دراسة حالة **Probistek** جامعة مولانا مالك إبراهيم الإسلامية الحكومية). قسم هندسة المعلوماتية بكلية العلوم والتكنولوجيا بجامعة مولانا مالك إبراهيم الإسلامية الحكومية.

المشرف: (1) الدكتور سوهارتونو. (2) عين اليقين الماجستير.

كلمات مفتاحية: الخدمات الصغرى، تحديد السياق، التصميم الموجة بالمجال، التعليم الإلكتروني.

الخدمات الصغرى (Micro Service) من مفهوم هندسة البرمجيات حيث يتكون نظامها من خدمات صغيرة تعمل بعضهم بعضاً، وتكون مستقلة، وقابلة للنشر والتطوير التي قد تمت تصميمها بالتحديد السياقي. تحل الخدمات الصغرى على الخدمات التي تتطلب المعالجة، دون التأثير على الخدمات الأخرى. تستخدم الخدمات الصغرى تحديد السياق من التصميم الموجة بالمجال لتحديد حدود الخدمة. أما نتائج تقسيم النظام بتحديد السياق تتكون من أربع خدمات، هي: مصادقة الخدمة (كبوابة واجهة برمجة التطبيقات)، طبقات الخدمة، مواد الخدمة، واختبار الخدمة التي تمت تصميم كل منها باستخدام Node.js. بينما لتخزين البيانات باستخدام DBMS MySQL و MongoDB وكل تلك الخدمات يتصل باستخدام REST. يتم الاختبار مع متعددة الخدمات التي تستخدم بـ Algoritma Round Robin للحصول على النظام قابل للتطوير والمرونة. يختبر الاختبار الوظيفي البرامج الوظيفية باستخدام خزانة الاسرار (black box) مع النتائج أن تعمل جميع النظام وفقاً لوظائفها. يستخدم اختبار التكامل Postman لاختبار التكامل بين الخدمات مما يؤدي إلى إمكانية اتصال الخدمة وتبادل البيانات. تتكون وحدة الاختبار و رمز التغطية من اختبار الإيجابي واختبار السلبي لاختبار البرمجيات التي تعمل وفقاً مع تخطيط الاختبار الذي تم إجراؤه.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang atau lebih dikenal dengan sebutan UIN Maliki Malang merupakan salah satu perguruan tinggi Islam terbaik di Indonesia. UIN Maliki Malang sukses dalam mengembangkan model pendidikan yang memadukan tradisi antara universitas dan pesantren. Selain unggul dalam mengantar sivitas akademika terampil berbahasa Arab dan Inggris, juga berprestasi dalam mencetak mahasiswa penghafal al-Qur'an yang menuju *World Class University*. Dan menjadi pilihan terbaik bagi mahasiswa yang berasal lebih dari 16 negara. Selain terdiri dari beberapa fakultas, UIN Maliki Malang juga mempunyai program bisnis teknologi (Probistek).

Program Bisnis Teknologi (Probistek) merupakan program inovasi bisnis dan teknologi berlandaskan Ulul Albab yang berada di bawah naungan UIN Maulana Malik Ibrahim Malang. Berdiri sejak tahun 2006 yang diketuai oleh Dr. Suhartono M.Kom. Probistek mempunyai jenjang pendidikan selama 1 dan 2 tahun yang setara dengan Diploma 1 dan Diploma 2. Di Probistek dikenal peserta didik yaitu seorang yang menempuh pendidikan di sana dan staff pengajar yaitu seorang yang memberi didikan kepada peserta didiknya. Probistek bertujuan mempersiapkan peserta didiknya menjadi tenaga profesional dan berjiwa *entrepreneur* serta memiliki jalur karier yang spesifik pada bidang keahliannya. Didalamnya terdapat 6 program keahlian pilihan, yaitu Komputer Akuntansi Perpajakan, *Web Development*, Desain Grafis Multimedia, Film & Televisi,

English for Tourism, dan *Mobile Development / Android*. Program keahlian yang lainnya akan dikembangkan untuk memenuhi kebutuhan akan SDM yang memiliki karakter kuat, berketampilan khusus di bidangnya, dan siap untuk bersaing di dunia kerja.

Agar ada perubahan dalam proses belajar mengajar pada Probistek maka diperlukan *e-learning*. Dengan adanya *e-learning* proses belajar mengajar tidak hanya mendengar uraian materi dari *guru* namun siswa juga dapat melakukan hal lain seperti mengamati, melakukan, mendemonstrasikan dan lain-lain sehingga dapat meng-*improve* pengetahuan dari siswa.

Sebuah sistem berkembang karena kebutuhan bisnis, dimana perangkat lunak disusun berdasarkan arsitektur monolitik karena merupakan cara termudah dan tercepat untuk membangun perangkat lunak. Akan tetapi setelah bertahun-tahun untuk menambahkan fitur baru menjadi lebih sulit karena kompleksnya sistem dan struktur dari perangkat lunak yang dibuat. Sehingga merubah monolitik menjadi *microservices* memungkinkan *software* menjadi lebih otonom sehingga sistem dapat bekerja khusus dalam tugas tertentu. Munculnya *microservices* dapat menggantikan aplikasi monolitik yang kompleks (Nadareishvili, dkk 2016). Di dalam aplikasi yang kompleks, *code base* yang dihasilkan juga besar sehingga akan memakan waktu yang lama untuk melakukan *maintenance* atau *track* baris *code* yang bermasalah. Selain itu *microservices* sendiri muncul karena didorong oleh beberapa faktor diantaranya untuk mengatasi keterbatasan gaya SOA (Pahl & Jamshidi, 2016). *E-learning* merupakan program dengan *request* tinggi dan butuh sistem yang stabil dengan model yang terstruktur, sehingga *microservices* cocok digunakan untuk membangun *elearning*.

Microservices dapat mengelola sistem menjadi lebih adaptif terhadap perubahan (*requirement change*) (Munawar & Hodijah, 2018), yang cocok diterapkan pada sistem *e-learning* karena *e-learning* merupakan sistem yang dinamis yang sering dilakukan *maintenance* guna menunjang pendidikan yang lebih baik. Ditambah lagi *microservices* dapat memberikan manfaat yang signifikan, diantaranya untuk merancang, mengembangkan, menguji dan merilis layanan dengan sangat gesit (Francesso, et al, 2019).

Microservices hadir sebagai suatu arsitektur dalam pengembangan suatu perangkat lunak di mana suatu sistem akan dipecah menjadi beberapa *service* kecil yang independen dan dapat dijalankan serta dapat berinteraksi antara *service* satu sama lain. *Service* tersebut akan menjalankan suatu proses tertentu, selain itu *service* tersebut juga dapat saling berkomunikasi untuk menjalankan suatu proses bisnis tertentu.

Integrasi *microservices* dengan ayat Qur'an berkaitan dengan pengorganisasian atau *shaff* dalam surat *Al-Shaff* ayat 4 :

إِنَّ اللَّهَ يُحِبُّ الَّذِينَ يُقَاتِلُونَ فِي سَبِيلِهِ صَفًّا كَانَهُمْ بُنِينَ مَرَصُوصٍ

Artinya: “Sesungguhnya Allah mencintai orang-orang yang berperang di jalan-Nya dalam barisan yang teratur, seakan-akan mereka seperti suatu bangunan yang tersusun kokoh”. **QS. Al-Shaff (4).**

Dalam tafsir *As Sa'di* karya Syaikh Abdurrahman bin Nashir as-Sa'di ayat tersebut mengandung anjuran dari Allah SWT terhadap hambanya untuk berjihad di jalan-Nya, dan juga memberitahukan apa yang seharusnya dilakukan di medan perang, yaitu menyusun barisan (*shaff*) yang kokoh, kuat, rapi, lurus dan tanpa

celah antara satu dengan yang lainnya. Dengan barisan yang teratur dan rapi akan menjadi sebuah barisan kelompok (*shaff*) yang kokoh, memunculkan rasa kebersamaan dan membuat musuh merasa takut.

Sewaktu di medan perang Nabi Muhammad SAW menyusun barisan untuk strategi perang. Kemudian beliau meletakkan barisan-barisan itu pada tempat yang telah diatur nabi. Diantara barisan yang satu dengan barisan yang lain fokus pada tugas masing-masing dan tidak bergantung antara satu dengan yang lain. Dengan cara yang seperti ini maka akan membuat pekerjaan menjadi sempurna.

Suatu pekerjaan apabila dilakukan secara teratur dan terarah akan menghasilkan yang baik, maka dalam pengorganisasian barisan kelompok baik proses juga dilakukan secara terarah dan teratur. Hal ini konsepnya mirip dengan *microservices* dimana pada satu sistem terdapat beberapa *services* yang masing-masing *service* diatur dan dikelola berdasarkan batasan (*bounded context*) atau fungsi dari masing-masing *services*. Seperti halnya perang, dalam sistem juga ada yang mengatur barisan-barisan (*services-services*) yaitu berupa *api-gateway*. Namun *services-services* tersebut merupakan suatu kesatuan yang mempunyai satu tujuan. *Services-services* diatur dengan rapi dan teratur agar menjadi sistem yang kokoh atau mempunyai kualitas yang baik.

Namun *microservices* sendiri memiliki kelemahan dalam memisahkan *service* yang ada, yaitu tidak adanya aturan pasti mengenai *design* suatu sistem. Oleh karena itu sebuah *microservices* harus bekerja sama dengan suatu desain *pattern*. Salah satu contoh desain *pattern* adalah *Domain Driven Design (DDD)*.

Domain Driven Design bertindak sebagai suatu aturan pelengkap dalam mengembangkan suatu sistem yang kompleks.

Pola untuk mengembangkan suatu sistem yang kompleks di mana *developer* akan menempatkan perhatian pada inti dari suatu aplikasi serta fokus pada kompleksitas suatu *domain* bisnis (Evans, 2004). *Domain Driven Design* berfokus pada tiga prinsip utama, yaitu fokus pada *domain* inti dan *domain logic*, desain kompleks dasar pada model *domain*, bekerja sama dengan *domain experts* untuk memperbaiki model aplikasi dan menyelesaikan masalah terkait *domain* yang muncul (Powell & Morse, 2017). Jadi dalam penerapan *Domain Driven Design* harus ada *domain expert* yang sekaligus sebagai analisis yang mengetahui bagian-bagian detail dari pengembangan aplikasi yang akan dibuat.

Pada studi kasus pembangunan aplikasi *e-learning* Probistek Universitas Islam Negeri Maulana Malik Ibrahim Malang, penulis menggunakan *Arsitektur Microservices* dengan *Node.js*, sedangkan untuk *client* menggunakan bahasa pemrograman *PHP* dan untuk penyimpanan datanya menggunakan *DBMS MongoDB* dan *MySQL*. *Arsitektur Microservices* digunakan untuk melakukan setiap *service* yang akan fokus pada satu fungsionalitas tertentu.

Oleh karena itu pada studi kasus pembangunan aplikasi *e-learning* Probistek Universitas Islam Negeri Maulana Malik Ibrahim Malang, penulis menggunakan *Arsitektur Microservices* untuk manajemen arsitektur pada sistem sehingga dapat meminimalkan waktu serta proses *development*, *deployment* maupun *maintenance* yang terfokus pada *service* yang mengalami masalah. Selain itu dengan penerapan *Domain Driven Design* sistem diharapkan mempunyai struktur kode yang baik, terstruktur serta *testable*.

1.2 Identifikasi Masalah

Berdasarkan latar belakang tersebut, maka diidentifikasi masalah sebagai berikut:

1. Bagaimana membangun aplikasi sistem *e-Learning* Probistek UIN Maulana Malik Ibrahim Malang menggunakan *Microservices* dan *Domain Driven Design*?
2. Bagaimana pengaruh *Microservices* dan *Domain Driven Design* terhadap sistem *e-Learning* Probistek UIN Maulana Malik Ibrahim Malang ?

1.3 Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut :

1. Membangun sistem *e-Learning* pada Probistek UIN Maulana Malik Ibrahim Malang.
2. Untuk mengetahui pengaruh *Microservices* dan *Domain Driven Design* dalam membangun sistem *e-Learning* Probistek UIN Maulana Malik Ibrahim Malang.

1.4 Manfaat Penelitian

Manfaat yang diperoleh pada penelitian ini sebagai berikut :

1. Pembuatan sistem *e-learning* untuk Probistek sehingga dapat mempermudah kegiatan belajar mengajar,
2. Mengenalkan kepada para *software developer* mengenai arsitektur *microservices* dan *design pattern Domain Driven Design*.

3. Dengan menggunakan *Microservices* akan menjadikan arsitektur sistem yang baik, mudah dalam proses *development* dan *deployment*.
4. Sistem terdiri dari kode yang terstruktur dengan menggunakan *design pattern Domain Driven Design*.

1.5 Batasan Masalah

Batasan masalah pada penelitian ini adalah :

1. Pembahasan difokuskan pada pemodelan sistem berbasis *Microservices* dan *Domain Driven Design* sebagai *design pattern*.
2. Implementasi dilakukan dengan teknik pemrograman berbasis objek.
3. Sistem *e-learning* yang dikerjakan berbasis *website*.
4. *Stack* yang digunakan adalah *Node.js*, *PHP* dan *DBMS MongoDB* dan *MySQL*.
5. Komunikasi antar *service* menggunakan *REST*
6. Data diambil dari Probistek
7. Soal yang tersedia dalam sistem *e-learning* berupa pilihan ganda
8. *Microservices* pada *server 1* sama dengan *Microservices* pada *server 2* dan *3*

BAB II KAJIAN PUSTAKA

2.1 Penelitian Terkait

Penelitian tahun 2016 (Purnama, 2016) menerapkan *Microservices* pada aplikasi pengelolaan skripsi di STMIK AKAKOM Yogyakarta. Aplikasi yang dibangun mengimplementasikan arsitektur *Microservices* sehingga dapat menghasilkan layanan secara terpisah yang hubungan ke *private cloud* menggunakan *Docker*. Selain itu *service-service* yang ada dijalankan dan berkomunikasi satu sama lain melalui *http* atau *tcp*. Hasil dari penelitian ini antara lain :

- Layanan secara terpisah dapat dibangun menggunakan *Microservices*
- Sistem yang dibangun dapat melakukan pencarian skripsi berdasarkan kata kunci dari abstrak ataupun dari judul.
- Sistem diterapkan dalam *private cloud* dengan menggunakan *Docker*.
- *Service* saling berkomunikasi antara satu dengan yang lain maupun berdiri sendiri
- Digunakan teknologi *docker* untuk melakukan pengujian aplikasi berjalan atau tidak

Pada tahun 2017 (Suryotrisongko, 2017) melakukan penelitian berupa penyusunan model dan pembuatan *proof of concept* dari modifikasi arsitektur *software* yang terdistribusi yang berbasis *microservices* dan *Docker-container* untuk *Resilient Information Systems*. Penelitian ini menunjukkan implementasi *microservice* untuk studi kasus sistem manajemen asosiasi/keanggotaan yang digunakan untuk membuktikan adopsi model *Microservices Migration Patterns*

yang kemudian mengevaluasi model di Asosiasi Sistem Informasi Indonesia (AISINDO). Model atau desain arsitektur sistem informasi/*software* yang telah dikembangkan menunjukkan aspek kualitas resiliensi, dengan pendekatan *Microservices – Docker container* yang telah dikembangkan dengan system *back-end Spring Boot* dan *front-end Angular2*. Serta *proof of concept* dari model atau desain arsitektur yang diusulkan telah dibuat dengan menulis ulang program atau *refactoring Software Open Source* manajemen asosiasi / keanggotaan. Aspek kualitas relisiensi dapat dicapai dengan cara mengamati ketangguhan system ketika salah satu *instance microservices* mengalami gangguan.

Perbedaan penelitian yang sedang dilakukan peneliti dengan penelitian pertama yaitu :

1. Penelitian yang dilakukan peneliti menerapkan *design pattern* yaitu *domain driven design* sedangkan pada penelitian sebelumnya tidak.
2. Penelitian dilakukan dengan menerapkan *docker* namun penelitian yang dilakukan peneliti tidak.
3. Penelitian menggunakan 1 DBMS sedangkan penelitian yang dilakukan peneliti menggunakan 2 DBMS.
4. Pengujian hanya dilakukan apakah aplikasi berjalan atau tidak sedangkan penelitian yang dilakukan peneliti melakukan pengujian pada *client* maupun *server* dan dengan scenario pengujian *microservices*.
5. Penelitian yang dilakukan peneliti menjelaskan pengaruh dari penerapan *microservces* pada sistem sedangkan penelitian sebelumnya hanya menjelaskan bahwa *microservices* dapat digunakan untuk membangun layanan secara terpisah.

Sedangkan perbedaan penelitian yang sedang dilakukan peneliti dengan penelitian kedua yaitu :

1. Penelitian yang dilakukan peneliti menerapkan *design pattern* yaitu *domain driven design* sedangkan pada penelitian sebelumnya tidak.
2. Penelitian yang dilakukan focus pada aspek resiliensi sedangkan penelitian yang dilakukan peneliti lebih melebar.
3. Penelitian dilakukan dengan menggunakan sistem yang sudah ada atau *migration pattern* sedangkan penelitian yang dilakukan peneliti membangun sistem dari awal.
4. Penelitian dilakukan dengan menerapkan *docker* namun penelitian yang dilakukan peneliti tidak.
5. Penelitian menggunakan 1 DBMS sedangkan penelitian yang dilakukan peneliti menggunakan 2 DBMS.

2.2 E-Learning

Sistem Pembelajaran Elektronik (Hartanto & Purwo, 2002) atau *e-learning* merupakan suatu teknologi informasi pada instansi pendidikan yang tersambung pada jaringan internet. Onno W. Purbo menjelaskan istilah “e” merupakan kepanjangan dari elektronik. Sedangkan *learning* dalam bahasa Indonesia berarti pembelajaran. Sehingga *e-learning* merupakan bentuk teknologi yang berguna mendukung kegiatan belajar mengajar yang tersambung dalam internet. Sedangkan Marc J. Rosenberg (Rosenberg, 2001) mengemukakan *e-learning* merujuk pada penggunaan teknologi internet untuk meningkatkan pengetahuan dan keterampilan.

Tiga kriteria dasar dalam *e-learning*, yaitu :

1. *E-Learning* bersifat jaringan.
2. *E-Learning* disediakan untuk *client* dengan teknologi internet.
3. *E-Learning* merupakan solusi pembelajaran konvensional.

Uraian diatas menunjukkan bahwa *e-learning* memanfaatkan teknologi internet yang berguna untuk memperkuat model pembelajaran konvensional.

2.2.1 Konsep *E-Learning*

Pengajaran konvensional (Budiarto, 2009) yang terjadi di sekolah-sekolah atau yang disebut pengajaran tradisional merupakan pengajaran yang kurang efektif jika dibanding dengan pengajaran yang sudah modern. Salah satu ciri pengajaran modern adalah sudah menggunakan teknologi internet. Dengan teknologi ini muncullah pembelajaran seperti *e-learning* yang diharapkan dapat membantu atau menambah materi pengajaran yang belum diberikan pada pengajaran konvensional, sehingga peran *e-learning* bukan untuk menggantikan tetapi membantu baik untuk siswa maupun guru dalam hal menunjang pendidikan lebih tinggi.

Elemen yang ada di dalam *e-learning* adalah sebagai berikut :

1. Soal-soal : Seperti halnya pada pengajaran konvensional, *e-learning* juga disediakan soal-soal baik itu sebelum atau sesudah diberikan modul materi. Soal disini biasanya untuk mengukur sejauh mana kemampuan siswa dalam memahami materi yang sedang atau telah diberikan.
2. Komunitas : komunitas, kumpulan belajar atau grup dalam pembelajaran *online* dapat membantu siswa dalam memperoleh informasi maupun untuk bertukar pendapat dalam memecahkan masalah.

3. Pengajar *online* : dalam hal ini pengajar juga mendapat pemberitahuan dari *e-learning* sehingga guru dan siswa dapat saling berkomunikasi.
4. Kesempatan bekerja sama : karena pengguna dapat menggunakan *e-learning* dimana dan kapan saja, hal ini dapat memungkinkan mereka melakukan diskusi secara bersamaan tanpa terkendala jarak.
5. Multimedia : Berbeda dengan pengajaran konvensional, dengan pengajaran modern materi ataupun soal yang ditampilkan dapat berupa suara, gambar atau video yang akan memperjelas atau sekedar memberikan daya tarik untuk siswa.

2.2.2 Komponen E-Learning

Beberapa komponen yang membentuk *e-learning* adalah :

1. Infrastruktur *e-Learning*

Adalah hal mendasar untuk mengakses *e-learning*, yaitu berupa *personal computer* (PC), jaringan internet dan perlengkapan lain yang dibutuhkan untuk mendukung pembelajaran modern.

2. Sistem dan Aplikasi *e-Learning*

Sistem *e-learning* yang digunakan dalam pembelajaran sering disebut dengan *Learning Management System (LMS)*, seperti *Moodle*, *Dokeos*, *Atutor*, dll. Di dalam penelitian ini penulis membangun aplikasi *e-Learning* sendiri dari awal yang lebih sederhana.

3. Konten *e-Learning*

Materi atau bahan pengajaran dapat berupa multimedia atau berupa tulisan text biasa seperti halnya yang ada pada buku pelajaran biasa. Materi atau bahan ajar dapat diakses sewaktu-waktu oleh pengguna kapan dan dimana saja.

Aktor atau pelaku dalam pelaksanaan *e-Learning* dapat dikatakan sama dengan proses belajar mengajar konvensional, yaitu :

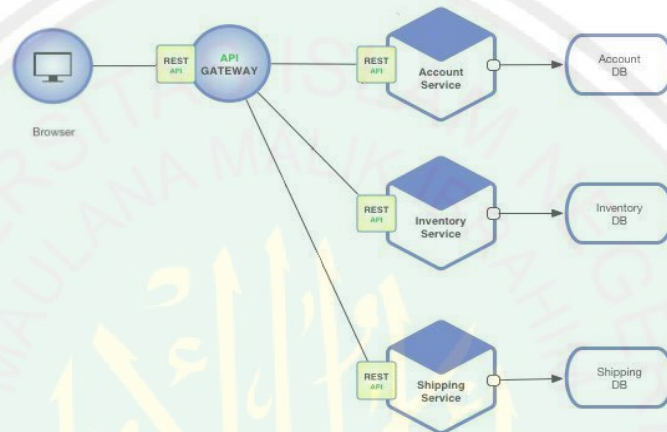
1. Dosen, guru (instruktur) yang memberi arahan, bimbingan, tugas, dll.
2. Siswa, mahasiswa yang menerima bimbingan dari guru ataupun langsung dari sistem.
3. Administrator yang mengelola, mengatur pembelajaran *e-learning*, seperti membuat kelas, registrasi user, dll.

2.3 *Microservices Architecture*

2.3.1 *Pengertian Microservices Architecture*

Microservices adalah komponen kecil perangkat lunak yang khusus dalam satu tugas tertentu dan bekerja sama untuk mencapai tugas tingkat tinggi (Gonzalez, 2016). Aplikasi diatur sedemikian rupa sehingga saling terpisah menjadi *service-service* kecil yang independen, berfungsi spesifik (*high cohesion*) dan tidak saling bergantung pada komponen program lainnya (*loose coupling*), dengan antarmuka *API (Application Programming Interface)* (Newman, 2015). Secara sederhana arsitektur aplikasi *Microservices* menggunakan desain yang memecah aplikasi berdasarkan fungsinya secara spesifik. Tidak sekedar memisahkan berdasarkan *user-role* atau *subdomain* saja, tetapi aplikasi akan di *breakdown* lebih rinci dari segi fungsionalitasnya. Aplikasi dirancang agar setiap fungsi bekerja secara independen. Dan setiap fungsi dapat menggunakan teknologi *stack* sesuai dengan kebutuhan yang berarti akan ada teknologi yang berbeda dalam satu aplikasi besar (Triwahyudhi, 2016). *Microservices* mendapat banyak perhatian di akhir ini. *Microservices* itu sendiri muncul karena didorong oleh beberapa faktor diantaranya untuk mengatasi keterbatasan gaya SOA (Pahl

& Jamshidi, 2016). Hal ini merupakan langkah selanjutnya dalam evolusi aplikasi perusahaan. *Microservices* muncul dari *domain driven design*, *continuous delivery*, virtualisasi berdasarkan permintaan, infrastruktur otomatisasi, dan kebutuhan akan tim kecil yang mengelola keseluruhan dari siklus produk (Livora, 2016). Adapun aplikasi terdiri dari beberapa layanan seperti 2.1 di bawah ini :



Gambar 2.1 Microservices

Dari gambar 2.1 diatas terlihat bahwa konsep dari *microservices* yaitu *service-service* kecil yang independen, *service* dapat berjalan sendiri dan dapat mengelola sendiri *service* yang ada tanpa berkaitan dengan *service* lain.

Dalam *microservices* menurut penulis Mike Amundsen, Irakli Nadareishvili, Ronnie Mitra, dan Matt McLarty (Slocum, 2018) mengemukakan sifat aplikasi *microservices* adalah sebagai berikut :

1. Ukurannya kecil
2. Dibatasi oleh konteks
3. Berkembang secara otonomi
4. Dapat di *deploy* secara independen
5. Desentralisasi
6. Dibangun dan dirilis dengan proses otomatis

2.3.2 Aspek dalam *Microservices*

Aspek dalam *microservices* dibedakan menjadi 3 (Weir & Wunderlich, 2016), yaitu :

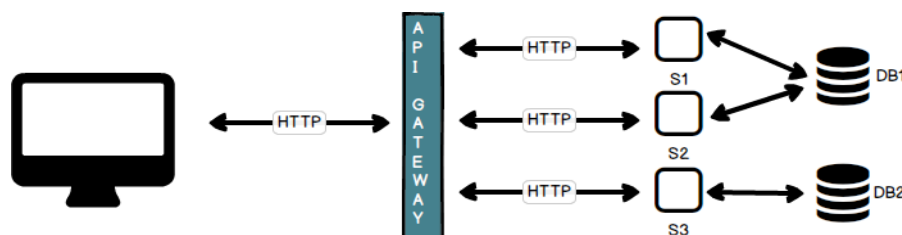
1. *Architectural*, terdiri dari :

a. *Bounded Context*

Arsitektur *microservices* yang dibangun berdasarkan konsep *share-as-little-as-possible* memanfaatkan konsep dari *domain driven design* yang disebut *bounded context*. *Bounded context* merupakan bagaimana mengelompokkan *service*. Secara arsitektural *bounded context* mengacu pada bagaimana modul menjadi *service* berdasarkan fungsionalnya. Komponen *service* yang dirancang pada dasarnya independen berdasarkan batasan yang telah didefinisikan. Keuntungan memanfaatkan *bounded context* yaitu menjaga *service* menjadi lebih mudah karena dengan dijadikan satu yang mempunyai fungsional yang sama akan mengurangi modul yang ada dan memungkinkan *service* untuk berkembang tanpa mempengaruhi *service* yang lainnya.

b. *API gateway*

API Gateway merupakan gerbang dari beberapa *API* yang bertugas menangani berbagai *request*, mengatur *request*, manajemen *API*, *authentication API* dan lainnya (Polladino, 2018). Berikut *pattern* dari *API Gateway* dapat dilihat pada gambar 2.2 di bawah ini :



Gambar 2.2 *API Gateway Pattern*

Dari gambar 2.2 diatas dapat dilihat bahwa *API Gateway* mengolah *request* baik dari *client-server* maupun *request* dari *server-server*.

c. *Polyglot*

Di dalam *microservices*, *service* satu dengan yang lain dapat dibangun dengan bahasa pemrograman dan *database* yang berbeda. Sehingga programmer dapat menggunakan teknologi terbaru pada sistem yang akan dibangun.

d. *Choreographed*

Maksud dari *choreography* disini artinya koordinasi antar *service* tanpa perantara pusat. Dengan *choreography* satu *service* memanggil *service* yang lain dan *service* yang dipanggil memanggil *service* lainnya dan seterusnya.

2. *Technical*, terdiri dari :

a. *Deployed independently*

Setelah *service* selesai dikerjakan dapat langsung di *deploy* tanpa menunggu *service* yang lain selesai.

b. *Scales independently*

Dapat mengembangkan *service* yang ada tanpa mempengaruhi *service* yang lain.

3. *Organizational*, terdiri dari :

a. *Product not project*

Kumpulan dari beberapa *service* disebut sebagai produk, bukan sebagai proyek.

b. *Small teams*

Tidak memerlukan tim yang besar untuk membangun *microservices*.

c. *Decentralised governance*

Karena *service* bersifat otonom maka dapat mengatur dan mengolah sendiri *service* yang ada tanpa berpengaruh terhadap *service* yang lain.

2.3.3 Keuntungan *Microservices*

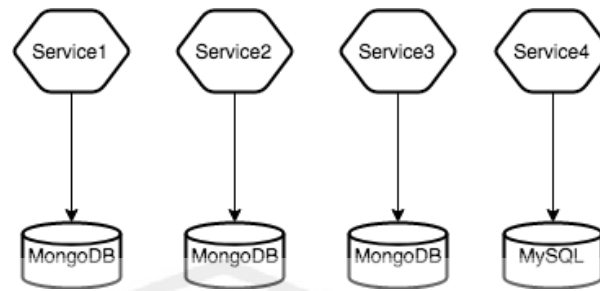
Berikut keuntungan yang dapat diambil dari *microservices* yang berkonsep sistem terdistribusi dan arsitektur berorientasi layanan (Newman, 2015) :

a. Heterogenitas Teknologi

Konsep *microservices* yang terdiri dari beberapa *service* membuat *developer* dapat memutuskan untuk menggunakan teknologi yang berbeda. Hal ini membuat kita membuat kita memilih *tool* yang tepat. Jika salah satu bagian dari sistem perlu peningkatan kinerja, *developer* dapat memutuskan untuk menggunakan *stack* teknologi yang berbeda yang mampu mencapai tingkat kinerja yang dibutuhkan.

Selain itu dengan *microservices*, *developer* dapat mempelajari atau menggunakan teknologi baru yang dapat membantu pekerjaan. Salah satu penghalang besar untuk belajar teknologi baru adalah resikonya. Dengan aplikasi *monolithic*, jika ingin mencoba bahasa pemrograman, *database* atau kerangka baru perubahan apapun akan berdampak pada seluruh sistem. Dengan sistem *microservices*, kita dapat memilih *service* yang dirasa beresiko paling rendah. Dengan belajar teknologi baru *developer* dapat mengetahui teknologi yang dapat memberikan keuntungan bagi mereka.

Untuk penyimpanan data, *database* yang digunakan tidak harus sama dengan *database* yang digunakan layanan sebelumnya namun juga dapat menggunakan *database* yang sama sekali berbeda dengan layanan lainnya. Hal ini yang dinamakan *polyglot persistence* (Fowler, 2011).



Gambar 2.3 Contoh *Polyglot Persistence*

Dari gambar 2.3 diatas dapat dilihat bahwa setiap *service* menerapkan *database* yang berbeda-beda. Dimana *service* 1, 2 dan 3 menggunakan *database MongoDB* sedangkan untuk *service* 4 menggunakan *database MySQL*.

b. Otonomi

Pada arsitektur *microservices*, layanan secara independen pada *engine* yang berbeda. Komunikasi di antara mereka dilakukan melalui jaringan. Hal ini sebenarnya dapat menambahkan beberapa *overhead* namun di sisi lain dapat mempertahankan modularitas, dan lebih mudah memahami bagian-bagian individual suatu sistem.

Dibanding aplikasi *monolithic* yang menggunakan *database* tunggal, dalam arsitektur *microservices* setiap layanan mengelola *instance* nya sendiri. Hal ini membantu menjaga *database* tetap dalam keadaan konsisten karena tidak ada layanan lain yang bisa langsung mengakses data. Layanan yang bertanggung jawab atas data mengenal semantiknya dengan baik dan berfungsi sebagai *gateway* untuk layanan lain yang ingin menggunakan data tersebut.

c. Ketahanan

Kunci dalam teknik ketahanan adalah sekat. Batas *service* inilah yang menjadi sekat. Apabila salah satu komponen sistem mengalami kegagalan maka tidak akan berdampak secara langsung pada sistem yang lain, sehingga *developer*

dapat mengatasi masalah tersebut secara bertahap dan sistem masih dapat berjalan. Berbeda dalam *service* yang *monolithic*, jika terdapat kegagalan maka keseluruhan sistem akan berhenti bekerja.

d. Mudah Dikembangkan (*Scalable*)

Pada sebuah *service* besar yang berbasis *monolithic*, *developer* harus mengembangkannya secara bersamaan. Apabila sistem tersebut memerlukan pengembangan, maka seluruh tatanan sistem akan mengalami perubahan. Sedangkan sistem dengan konsep *microservices* apabila memerlukan pengembangan, maka *developer* dapat memilih *service* mana yang akan dikembangkan tanpa memberikan pengaruh kepada *service* yang lain.

e. Mudah di *Deploy*

Apabila terdapat perubahan satu baris kode di aplikasi *monolithic* akan berpengaruh kepada semua sistem. Sedangkan pada *microservices* perubahan dapat dibawa ke layanan tunggal dan diterapkan secara independen dari seluruh sistem sehingga dapat mengatur baris kode lebih cepat. Apabila terjadi masalah, juga dapat diatasi dengan segera tanpa mengganggu *service* yang lain.

f. Keselarasan Organisasi

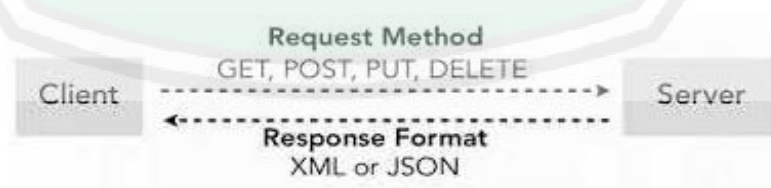
Banyak masalah yang terjadi terkait tim yang besar serta *codebase* yang besar. Masalah akan semakin buruk apabila tim sudah terdistribusi. Sedangkan untuk tim yang kecil yang mengerjakan *codebase* kecil pasti akan lebih produktif dari tim yang besar dan mengerjakan *task* yang besar. Disini *microservices* memungkinkan kita untuk menyesuaikan arsitektur dengan lebih baik ke organisasi yang ada sehingga membantu meminimalkan jumlah orang.

2.4 API Gateway

API Gateway adalah jenis layanan dalam arsitektur *microservices* yang menyediakan lapisan bersama dan *API* bagi *client* untuk berkomunikasi antar *service*. *API* dapat mengarahkan permintaan, mentransformasikan protokol, menggabungkan data dan menerapkan logika bersama seperti otentikasi. *API Gateway* berfungsi sebagai gerbang utama masuk ke layanan *microservices*. *Microservices* merupakan layanan dimana dapat menerapkan berbagai macam teknologi, sehingga tim yang membangun *microservices* dapat menggunakan berbagai bahasa pemrograman, *database* dan protokol berbeda untuk menerapkan *microservices*. Dari situlah disini peran *API Gateway* untuk menyediakan lapisan bersama untuk menangani perbedaan protokol layanan (Marton, 2017).

2.5 RESTful API / REST API

Mayoritas implementasi berfokus pada *REST* sebagai sarana untuk layanan *microservices* untuk berkomunikasi satu sama lain. Alasannya karena kesederhanaannya yaitu layanan berkomunikasi secara langsung dan serentak satu sama lain melalui *HTTP* tanpa memerlukan infrastruktur tambahan (Williams, 2015).



Gambar 2.4 REST Response dan REST Request

Dari gambar 2.4 diatas ditunjukkan bahwa *REST Request* dari *client* ke *server* mempunyai empat *method* yang sering digunakan yaitu *GET*, *POST*, *PUT*

dan *DELETE*. Sedangkan *response* yang dikirim dari *server* menuju *client* dapat berupa *XML* atau *JSON*.

2.5.1 Pengertian *RESTful API*

REST (*Representational State Transfer*) adalah standar komunikasi arsitektur berbasis website. *REST* pertama kali diperkenalkan oleh Roy Fielding pada tahun 2000. Umumnya *REST* menggunakan *HTTP* (*Hypertext Transfer Protocol*) sebagai protokol untuk komunikasi data. Tujuan dari *REST* yaitu agar sistem mempunyai performa yang baik sehingga mempercepat proses pertukaran atau komunikasi data.

Pada penerapannya *REST server* menyediakan *resources* (sumber daya/data) dan *REST client* mengakses dan menampilkan *resources* tersebut untuk penggunaan selanjutnya. Setiap *resources* diidentifikasi oleh *URIs* (*Universal Resource Identifiers*) atau *global ID*. *Resources* tersebut direpresentasikan dalam bentuk format teks, *JSON* atau *XML*. Pada umumnya format yang dipakai berupa *JSON* dan *XML* (Fediri, 2016).

2.5.2 Komponen *RESTful API*

Komponen penting dalam *RESTful API* (Haryandi, 2016), adalah :

a. *URL Design*

RESTful API diakses menggunakan protokol *HTTP*. Dalam memberi nama atau struktur *URL API* (*endpoint*) ada baiknya konsisten agar mudah dimengerti *developer* dan menghasilkan *API* yang baik. Contoh penamaan *URL / endpoint* adalah :

/group

/group/A1B12

/group/A1B12/algorithm

/group/A1B12/algorithm/12345

b. *HTTP Verbs*

Agar *server* menerima dan mengerti *request client* diperlukan metode.

Macam-macam metode yang ada pada *HTTP request* adalah : *GET*

GET adalah metode *HTTP request* yang digunakan untuk membaca atau mendapatkan data dari *resources*.

Contoh :

GET /group : mendapatkan data group

GET /group/A1B12 : mendapatkan data grup dengan ID A1B12

❖ *POST*

POST adalah metode *HTTP request* yang digunakan untuk membuat data baru dengan menyisipkan data dalam *body* saat *request* dilakukan.

Contoh :

POST /group : membuat data grup baru

❖ *PUT*

PUT adalah metode *HTTP request* yang biasanya digunakan untuk melakukan *update* atau perubahan data *resource*.

Contoh :

PUT /grup/A1B12 : melakukan perubahan data pada ID A1B12

❖ *DELETE*

DELETE adalah metode *HTTP request* yang digunakan untuk menghapus suatu data pada *resource*.

Contoh :

DELETE /group/A1B12 : menghapus data grup yang mempunyai ID A1B12

c. *HTTP Response Code*

HTTP Response Code adalah kode standarisasi untuk memberikan informasi hasil *request* kepada *client*. Macam-macam *response code* yang sering digunakan pada *REST*, yaitu :

- 200 *OK*

Response code yang menampilkan bahwa *request* berhasil.

- 201 *Created*

Response code ini menandakan bahwa *request* yang dilakukan berhasil dan data telah dibuat. Kode ini digunakan untuk mengkonfirmasi berhasilnya *request PUT* atau *POST*.

- 400 *Bad Request*

Response code ini memberikan tanda bahwa *request* yang dikirim salah atau data tidak ada.

- 401 *Unauthorized*

Response code ini memberikan tanda bahwa *request* yang dibuat membutuhkan *authentication* sebelum mengakses *resource*.

- 404 *Not Found*

Response code ini mempunyai arti bahwa *request* yang dipanggil tidak ditemukan.

- 405 *Method Not Allowed*

Response code ini memberi arti *request endpoint* ada namun metode *HTTP* yang digunakan tidak diperbolehkan.

- 409 *Conflict*

Response code ini berarti *request* yang dikirim sudah ada sebelumnya.

- 500 *Internal Server Error*

Response code ini berarti terdapat kesalahan pada sisi *server* atau *resource*.

d. *Format Response*

Respon dari *request* yang dihasilkan biasanya berupa data *XML* atau *JSON*. Setelah mendapat data tersebut, data dapat *diparsing* dan diolah sesuai kebutuhan. Pada penelitian ini penulis menggunakan *JSON* sebagai data *response*.

Contoh :

JSON

GET /group/A1B12

HTTP/1.1 200 OK

Content-Type: application/vnd.api+json

{

"id": "A1B12",

"first_name": "jhon",

"last_name": "doe",

"created": "2015-05-22T14:56:29.000Z",

"updated": "2015-05-22T14:56:29.000Z"

}

2.5.3 Cara Kerja *RESTful API*

Pengguna melakukan *request* data melalui *HTTP request* dan kemudian *server* merespon melalui *HTTP response* (Fediri, 2016).

Komponen *HTTP request* :

- ✓ *Verb*, *HTTP* method yang digunakan : *GET*, *POST*, *DELETE*, *PUT* dll.
- ✓ *Uniform Resource Identifier (URI)* : mengidentifikasi lokasi *resource* pada *server*.
- ✓ *HTTP Version*, versi dari *HTTP* yang digunakan, contoh *HTTP* v1.1.
- ✓ *Request Header*, berisi meta data untuk *HTTP Request*. Contoh, *type client/browser*, format yang didukung oleh *client*, format dari *body* pesan, *setting cache* dll.
- ✓ *Request Body*, konten dari data.

Komponen *HTTP response* :

- *Status/Response Code*, memberikan kode status server terhadap *resource* yang di *request*. misal : 404, artinya *resource* tidak ditemukan dan 200 *response OK*.
- *HTTP Version*, menunjukkan versi dari *HTTP* yang digunakan, contoh *HTTP* v1.1.
- *Response Header*, berisi meta data untuk *HTTP Response*. Contoh, *type server*, panjang *content*, tipe *content*, waktu *response*, dll.
- *Response Body*, konten dari data yang diberikan.

2.6 Domain Driven Design

Untuk membuat *software* yang harmonis dengan *domain* adalah dengan merefleksikan *domain* tersebut. Jadi konsep-konsep dan elemen-elemen yang ada pada *domain* harus terdapat pada *software* tersebut. Begitu juga dengan relasi antar elemen-elemennya.

2.6.1 Pengertian Domain

Untuk menentukan desain berbasis domain atau *domain driven design* harus menetapkan apa yang dimaksud dengan *domain* dalam konteks ini. Definisi *domain* menurut kamus adalah bidang pengetahuan atau aktivitas. Sedangkan yang dimaksud *domain* pada rekayasa perangkat lunak biasanya mengacu pada area subjek dimana aplikasi dimaksudkan untuk diterapkan. Dengan kata lain, selama pengembangan aplikasi, *domain* adalah bidang pengetahuan dan aktivitas seputar logika aplikasi.

2.6.2 Pengertian Domain Driven Design

Ketika *developer* menghadapi masalah yang kompleks, *developer* dapat memahami masing-masing kompleksitasnya. Dengan membongkar masalahnya, *developer* mengubahnya menjadi potongan yang lebih mudah dipahami dan dikelola. Teknik perancangan perangkat lunak yang dapat memahami dan mengatasi kompleksitas adalah *Domain Driven Design* (Sokhan, 2015). Definisi kamus umum dari *domain* adalah bidang pengetahuan atau aktivitas, yang diartikan dalam dunia rekayasa perangkat lunak mengacu pada area subjek dimana aplikasi dimaksudkan untuk diterapkan.

Domain Driven Design adalah metode pengembangan software menggunakan dan membangun berdasarkan gagasan dan ide *OOAD* (*Object Oriented Analysis and Design*). *Domain Driven Design* awalnya diperkenalkan dan dipopulerkan oleh Eric Evans dalam bukunya *Domain Driven Design: Tackling Complexity in the Heart of Software* (Evans, 2004). *Domain driven design* atau desain berbasis domain adalah perluasan dan penerapan konsep *domain*, karena berlaku untuk pengembangan *software*. Ini bertujuan untuk

memudahkan pengembangan *software* aplikasi yang kompleks dengan menghubungkan potongan *software* yang terkait menjadi model yang selalu berkembang (Powell & Morse, 2017).

Domain driven design berfokus pada 3 prinsip utama :

- Fokus pada *domain* inti dan *domain logic*.
- Desain kompleks dasar pada model *domain*.
- Terus berkolaborasi dengan *domain experts*, untuk memperbaiki model aplikasi dan menyelesaikan masalah terkait *domain* yang muncul.

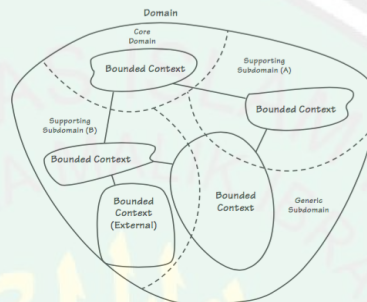
Istilah umum yang sering digunakan dalam *Domain driven design* :

- ❖ *Context* : pengaturana dimana sebuah kata atau pernyataan muncul yang menentukan maknanya. Pernyataan tentang model hanya bisa dipahami dalam konteks tertentu.
- ❖ *Model* : Suatu sistem abstraksi yang menggambarkan aspek-aspek tertentu dari *domain* dan dapat digunakan untuk memecahkan masalah yang terkait dengan *domain*.
- ❖ *Ubiquitous Language* : Bahasa yang terstruktur seputar *domain model* dan digunakan oleh semua anggota tim untuk menghubungkan semua aktifitas tim dengan *software*.
- ❖ *Bounded Context* : Deskripsi tentang batas (biasanya *subsistem*, atau pekerjaan tim tertentu) dimana model didefinisikan dan berlaku.

2.6.3 *Subdomain* dan *Bounded Context*

Subdomain merupakan pemecahan dari *domain* yang dibedakan berdasarkan *model*. Sistem yang kompleks dapat dipecah menjadi *subdomain* yang terpisah secara logis sesuai dengan fungsinya.

Sedangkan *bounded context* merupakan deskripsi tentang batasan (biasanya *subsistem*, atau pekerjaan tim tertentu) dimana *model* didefinisikan dan berlaku. Satu *bounded context* dapat terdiri dari berbagai *subdomain*. Berikut gambar lebih jelas mengenai *subdomain* dan *bounded context* dapat dilihat pada gambar 2.5 (Vernon, 2013) :



Gambar 2.5 Domain, Subdomain dan Bounded Context

2.7 NoSQL

Not Only SQL (NoSQL) merupakan konsep penyimpanan data non-relasional. Dengan kata lain NoSQL merupakan jenis basis data yang tidak menggunakan perintah SQL dalam memanipulasi basis data tersebut. Salah satu model penyimpanan NoSQL adalah *Document Database*. Dalam satu objek data disimpan dalam satu dokumen yang biasanya terdiri dari *key-value*, dan *value* sendiri bisa berupa *array* atau *key-value* bertingkat. Contohnya *MongoDB*.

Kelebihan database NoSQL :

- NoSQL dapat menampung data yang terstruktur, semi terstruktur dan tidak terstruktur secara efisien dalam skala besar (*big data / cloud*).
- Menggunakan OOP dalam pengaksesan atau memanipulasi data.
- Tidak adanya *schema* tabel yang kaku (*Dynamic Schema*), sehingga cocok untuk data yang tidak terstruktur.

- *Autosharding*, jika database NoSQL dijalankan di *cluster* server (*multiple* server) maka data akan tersebar secara otomatis dan merata keseluruh server.

2.8 MongoDB

MongoDB adalah *open source database* yang menawarkan kinerja tinggi, *high availability*, dan *automatic scalling*. Dalam *MongoDB* dikenal *BSON serialization*, yaitu format penyimpanan data untuk *document*. *MongoDB* sering digunakan untuk aplikasi berbasis *Cloud*, *Grid Computing*, atau *Big Data*. Berikut contoh dari penulisan *document* :

```
{
  _id: ObjectId("549081be0dbcf82140041a7"),
  name: "Lia Khoirunnisa'",
  age: 26,
  groups: [
    "biner",
    "kelasB"
  ]
}
```

Berikut tabel terminologi dan konsep dari *SQL* dan *MongoDB* dapat dilihat pada tabel 2.1 :

Tabel 2.1 Terminologi SQL dan MongoDB

No	SQL	MongoDB
1.	<i>Database</i>	<i>Database</i>
2.	<i>Table</i>	<i>Collection</i>
3.	<i>Row</i>	<i>BSON Document</i>
4.	<i>Column</i>	<i>BSON Field</i>
5.	<i>Index</i>	<i>Index</i>
6.	<i>Join</i>	<i>Embeded document dan linking</i>
7.	<i>Primary key</i>	<i>Primary key (otomatis)</i>
8.	<i>Groub by</i>	<i>Aggregation</i>

BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

3.1.1 Gambaran Umum Sistem

Sistem yang akan dibangun adalah sistem pembelajaran elektronik atau *e-learning* yang akan digunakan di Probistek Universitas Islam Negeri Maulana Malik Ibrahim Malang untuk kegiatan belajar mengajar antara siswa dan guru.

Sistem ini dibangun menggunakan *MSA (Microservices Architecture)* yang diterapkan dengan *domain driven design* sebagai *design pattern* untuk penerapan *bounded context* dari *microservices*. Dari sisi *server* penulis membangun *API* dengan menggunakan *Node.js 10.13.0* sedangkan dari sisi *client* penulis membangun dengan bahasa pemrograman *PHP 7.1.17* dengan *framework Laravel 5.5.40* serta untuk komunikasi antar *service* menggunakan protokol *HTTP*. Untuk penyimpanan data menggunakan *MongoDB 3.4.10* sebagai *database NoSQL* dengan hasil berupa format *JSON* sebagai datanya dan *MySQL*. Aktivitas penelitian tidak terlepas dari data yang menjadi bahan baku informasi. Data yang ada di dalam sistem *e-learning* akan disimpan di *database* masing-masing *service*. Penanganan untuk permintaan dan pengelolaan data menggunakan *DBMS (Database Management System)* yaitu *MongoDB* dan *MySQL*. Data pada *MongoDB* disebut dengan *document*. Data yang diperoleh yaitu dari hasil wawancara dengan salah satu pegawai Probistek dan hasil dari observasi penulis.

Pada penelitian ini penulis memilih *MongoDB* dikarenakan data-data yang akan berkembang menjadi fleksibel terhadap adanya perubahan di masa

mendatang. Hal ini terjadi karena penyimpanan data pada *NoSQL* tanpa perlu mendefinisikan tipe data dan ukurannya terlebih dahulu. Selain itu performanya yang cepat sehingga untuk proses penanganan data menjadi lebih cepat. Selain itu penulis juga memilih menggunakan *MySQL* dikarenakan untuk kebutuhan bisnis yang membutuhkan relasional seperti untuk mengolah data mengenai perhitungan.

Agar sistem dapat berinteraksi antara *service* satu dengan yang lain, diperlukan komunikasi. Komunikasi antar *service* di dalam sistem ini dilakukan dengan menggunakan *HTTP Request* atau *URL API* atau yang biasa disebut dengan *endpoint*. Selain itu komunikasi antar *service* dan *client* menggunakan *Guzzle* 6.3.3

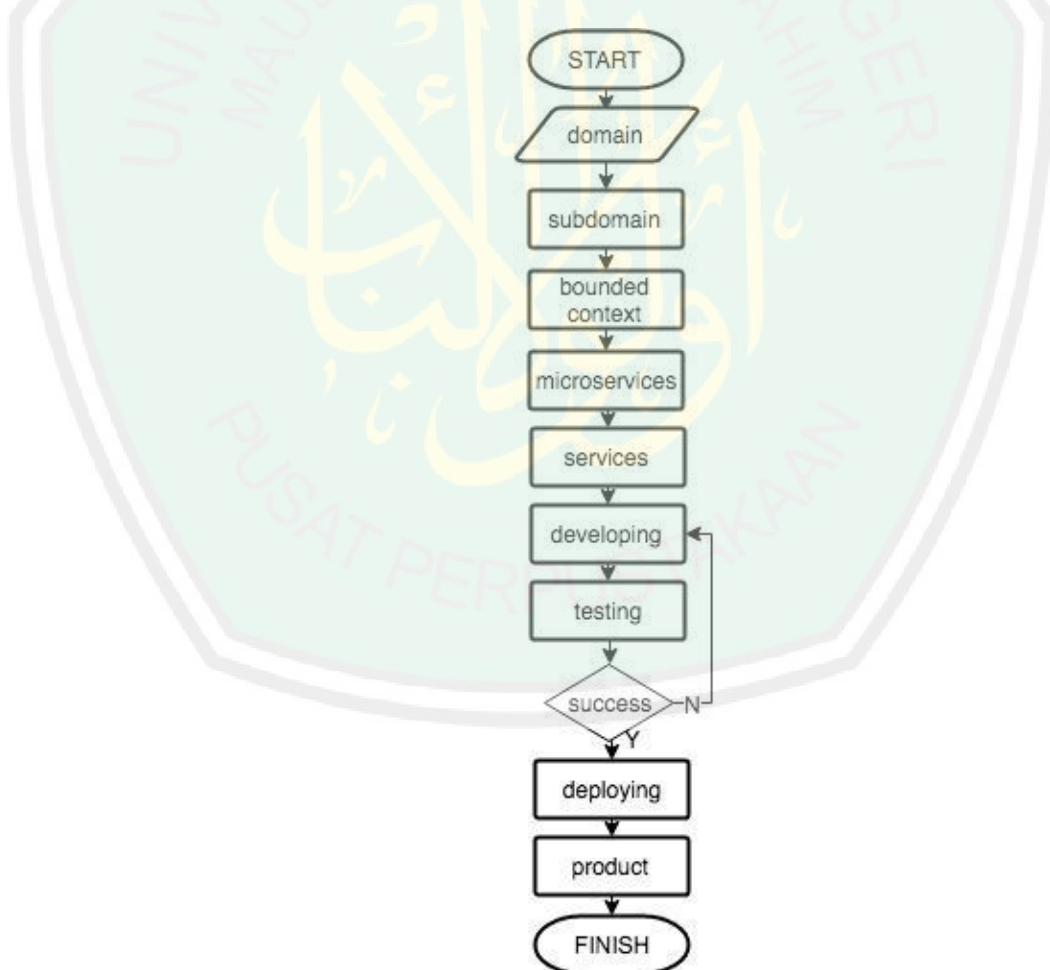
Aktivitas penelitian tidak terlepas dari data yang menjadi bahan baku informasi. Data yang ada di dalam sistem *e-learning* akan disimpan di *database* masing-masing *service*. Penanganan untuk permintaan dan pengelolaan data menggunakan *DBMS (Database Management System)* yaitu *MongoDB* dan *MySQL*. Data pada *MongoDB* disebut dengan *document*. Data yang diperoleh yaitu dari hasil wawancara dengan salah satu pegawai Probistek dan hasil dari observasi penulis.

Pada penelitian ini penulis memilih *MongoDB* dikarenakan data-data yang akan berkembang menjadi fleksibel terhadap adanya perubahan di masa mendatang. Hal ini terjadi karena penyimpanan data pada *NoSQL* tanpa perlu mendefinisikan tipe data dan ukurannya terlebih dahulu. Selain itu performanya yang cepat sehingga untuk proses penanganan data menjadi lebih cepat. Selain itu penulis juga memilih menggunakan *MySQL* dikarenakan untuk kebutuhan bisnis yang membutuhkan relasional seperti untuk mengolah data mengenai perhitungan.

Agar sistem dapat berinteraksi antara *service* satu dengan yang lain, diperlukan komunikasi. Komunikasi antar *service* di dalam sistem ini dilakukan dengan menggunakan *HTTP Request* atau *URL API* atau yang biasa disebut dengan *endpoint*. Selain itu komunikasi antar *service* dan *client* menggunakan *Guzzle 6.3.3*

3.1.2 Prosedur Penelitian

Prosedur dalam penelitian ini dilaksanakan dalam beberapa tahapan seperti yang dapat dilihat pada gambar 3.1 di bawah ini.



Gambar 3.1 Flowchart Prosedur Penelitian

Dari gambar 3.1 diatas berikut penjelasan masing-masing bagian dari prosedur penelitian.

1. *Domain*

Devinisi *domain* menurut kamus adalah bidang pengetahuan atau aktivitas. Sedangkan yang dimaksud *domain* pada rekayasa perangkat lunak biasanya mengacu pada area subjek dimana aplikasi dimaksudkan untuk diterapkan. Dengan kata lain, selama pengembangan aplikasi, *domain* adalah bidang pengetahuan dan aktivitas seputar logika aplikasi. Dalam hal ini *domain* dari penelitian ini adalah sistem pembelajaran atau *e-learning*.

2. *Subdomain*

Subdomain merupakan pemecahan dari *domain* yang dibedakan berdasarkan *model*. Sistem yang kompleks dapat dipecah menjadi *subdomain* yang terpisah secara logis sesuai dengan fungsinya. Dalam hal ini *subdomain* dari sistem *e-learning* adalah *user*, *student*, *teacher*, *materi*, *test*, dan program keahlian.

3. *Bounded Context*

Sedangkan *bounded context* merupakan deskripsi tentang batasan (biasanya *subsistem*, atau pekerjaan tim tertentu) dimana *model* didefinisikan dan berlaku. Satu *bounded context* dapat terdiri dari berbagai *subdomain*. Pada penelitian ini peneliti menggunakan *bounded context* terdiri dari 4 bagian, yaitu *auth-service* terdiri dari *subdomain user*, *student* dan *teacher*, *class-service* terdiri dari *subdomain* program keahlian, *material-service* terdiri dari *subdomain* materi, dan *test-service* terdiri dari *subdomain test*. Dari batasan ini *microservices* sudah mulai terbentuk.

4. *Microservices*

Merupakan *service-service* yang dibatasi dengan *bounded context* dimana setiap *service* bersifat independen dan melakukan tugas fungsionalnya. Dari keempat *bounded context* diatas, mulailah terlihat arsitektur *microservices* dimana terdapat 4 *service*, yaitu *auth-service*, *class-service*, *material-service*, dan *test-service*.

5. *Services*

Services merupakan bagian dari *microservices*. *Microservices* sendiri terdiri dari beberapa *services*. Dimana antara satu *service* dan *service* yang lain dapat berkomunikasi dengan menggunakan *HTTP Request*. Sifat dari masing-masing *service* ini sendiri bersifat otonom, yaitu mereka dapat memodifikasi secara bebas *service* mereka sendiri tanpa mempengaruhi *service* yang lain. Selain itu antara satu *service* dengan *service* lain dapat dibangun dengan bahasa pemrograman yang berbeda serta *database* yang berbeda.

6. *Developing*

Merupakan tahapan membangun sistem sesuai dengan rencana pembuatan proyek serta sesuai dengan tujuan.

7. *Testing*

Pengujian dilakukan untuk memastikan sistem telah sesuai dengan rancangan dan semua fungsi dapat digunakan tanpa ada *bug*.

8. *Deploying*

Dalam pembangunan perangkat lunak dibutuhkan *deployment* untuk mengetahui fitur-fitur apa saja yang telah berhasil dibangun dan dibuat sehingga nanti perangkat lunak yang dibangun sesuai dengan kebutuhan. Perangkat lunak

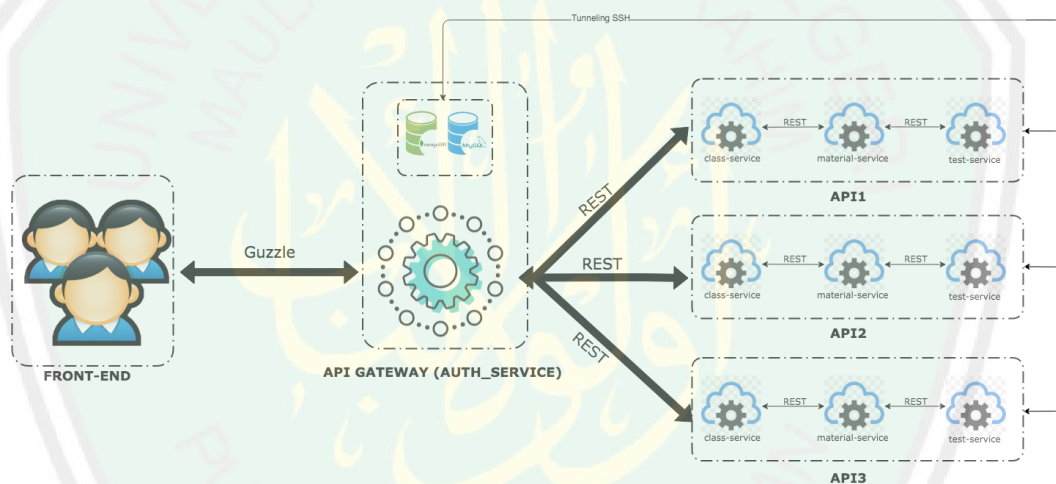
harus dijabarkan dan dipasang agar *bug* ataupun fitur yang tidak sesuai dengan kebutuhan dapat ditemukan.

9. Product

Di dalam konsep *microservices*, hasil akhir yaitu *service-service* yang ada disebut sebagai *product* bukan proyek.

3.2 Pemodelan Sistem

Berikut pemodelan sistem *e-learning* Probistek UIN Maulana Malik Ibrahim Malang dapat dilihat pada gambar 3.2 berikut.



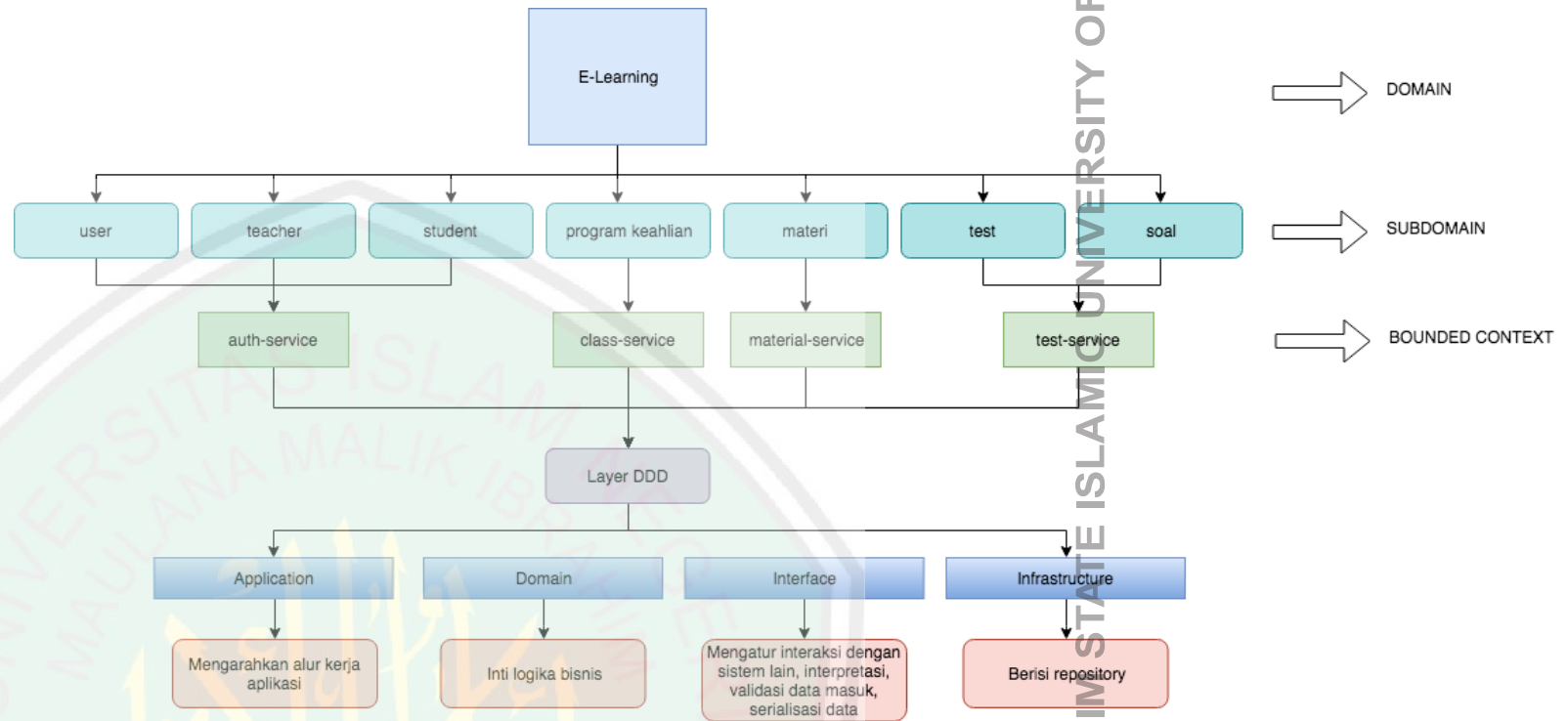
Gambar 3.2 Sistem *e-Learning* Probistek

Pada gambar 3.2 dapat dilihat bahwa *client* berkomunikasi dengan *server* menggunakan *Guzzle*. *Auth-service* berperan sebagai *API Gateway* yaitu *API* yang berfungsi sebagai gerbang utama masuk *microservices*, selain itu *API gateway* juga berperan untuk autentikasi dan manajemen *API*. Autentikasi *auth-service* adalah dengan menggunakan *JSON Web Token (JWT)*. Setelah berhasil melewati *API gateway* maka *request* dapat diteruskan ke *API* yang dituju, yaitu *material-service*, *class-service* dan *test-service*. Masing-masing *service* berkomunikasi

dengan menggunakan REST khususnya *protocol HTTP*. Sedangkan untuk mengakses ke *database*, *services* menggunakan *tunneling SSH*.

3.2.1 Domain Driven Design

Domain dalam rekayasa perangkat lunak biasanya mengacu pada area subjek dimana aplikasi dimaksudkan untuk diterapkan. *Domain* pada aplikasi ini merupakan *e-learning*, kemudian *domain* dipecah berdasarkan *model* menjadi *subdomain*. Sistem *e-learning* dipecah menjadi *subdomain* yang terpisah secara logis sesuai dengan fungsinya. Tahap selanjutnya adalah pembentukan *bounded context* dimana disinilah awal tahap pembuatan *microservices*. Karena konsep *bounded context* dalam *microservices* adalah bagaimana mengelompokkan *service*. Sam Newman membuat point penting (Nadareishvili dkk, 2016) bahwa *bounded context* mewakili domain bisnis otonom. Secara arsitektural *bounded context* mengacu pada bagaimana *modul* menjadi *service* berdasarkan fungsionalnya. Komponen *service* yang dirancang pada dasarnya independen berdasarkan batasan yang telah didefinisikan. Dimana kemudian satu bagian *bounded context* inilah yang menjadi *service* kecil dan awal terbentuknya *microservices*. Selanjutnya menentukan layer dari DDD. Untuk lebih jelasnya gambaran dari desain *domain driven design* dapat dilihat pada gambar 3.3 dibawah ini.



Gambar 3.3 Desain DDD

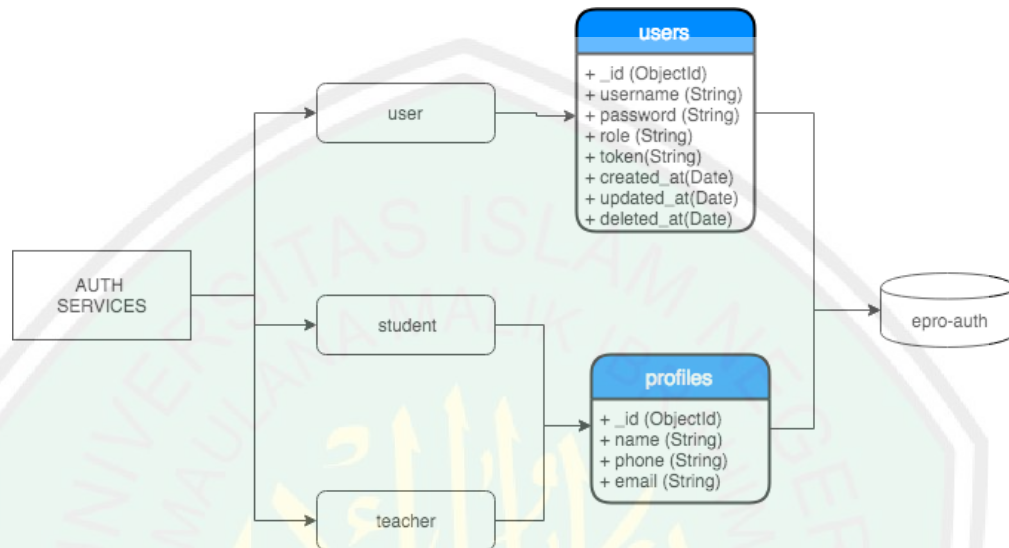
Desain pola *service* berarti *domain driven design* berperan dalam membagi 4 *services*, yaitu *auth-services*, *class-services*, *test-services* dan *material-services*. Sedangkan dari struktur kode *microservices* terbagi menjadi 2 bagian yaitu *logic* yang mengatur masalah algoritma dan *database* sedangkan *http* yang mengatur masalah *networking*. *Layer* pada *DDD* terdiri dari 4 *layer*, yaitu *application*, *domain* dan *infrastructure*. Bagian *logic* berada di dalam *layer domain* sedangkan *http* berada di dalam *layer interface*.

3.2.2 *Microservices Architecture*

Pada pemodelan arsitektur ini menjelaskan arsitektur dari *e-learning* yang menerapkan *Microservices Architecture* dimana di dalam *service* mengandung satu atau lebih *subdomain* yang dipisahkan oleh *bounded context*. Pada tahap ini penulis mendapatkan 4 *service*, yaitu *auth-service* dimana didalamnya mencakup *subdomain user*, *teacher*, dan *student*, *class-service* berisi *subdomain program keahlian*, *material-service* berisi *subdomain materi*, dan *test-service* berisi *subdomain test* dan soal. Setiap *service* dipisahkan antara *logic* dan *HTTP*. *Logic* berisi kumpulan modul yang berhubungan dengan algoritma dan *database* sedangkan *HTTP* berhubungan dengan *networking*. Pada *auth-service*, *class-service*, dan *materi-service* penulis menggunakan *database nosql* yaitu *MongoDB*, sedangkan untuk *test-service* penulis menggunakan *database MySQL* karena untuk kebutuhan bisnis yang membutuhkan relasional seperti untuk mengolah data mengenai perhitungan. Komunikasi antar *service* menggunakan *HTTP request* dimana *request* dan *response* baik dari *client* maupun *service* lain di

handle oleh API Layer. Berikut bagan setiap service mengenai e-learning microservices dapat dilihat pada gambar berikut ini.

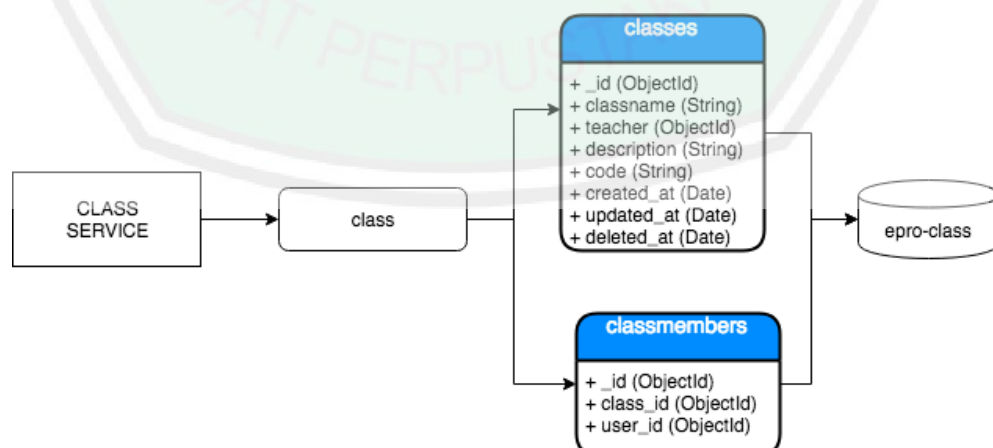
a. *auth-service*



Gambar 3.4 *auth-service*

Pada gambar 3.4 terlihat bahwa *auth-service* terdiri dari 3 subdomain, yaitu *user*, *student*, dan *teacher*.

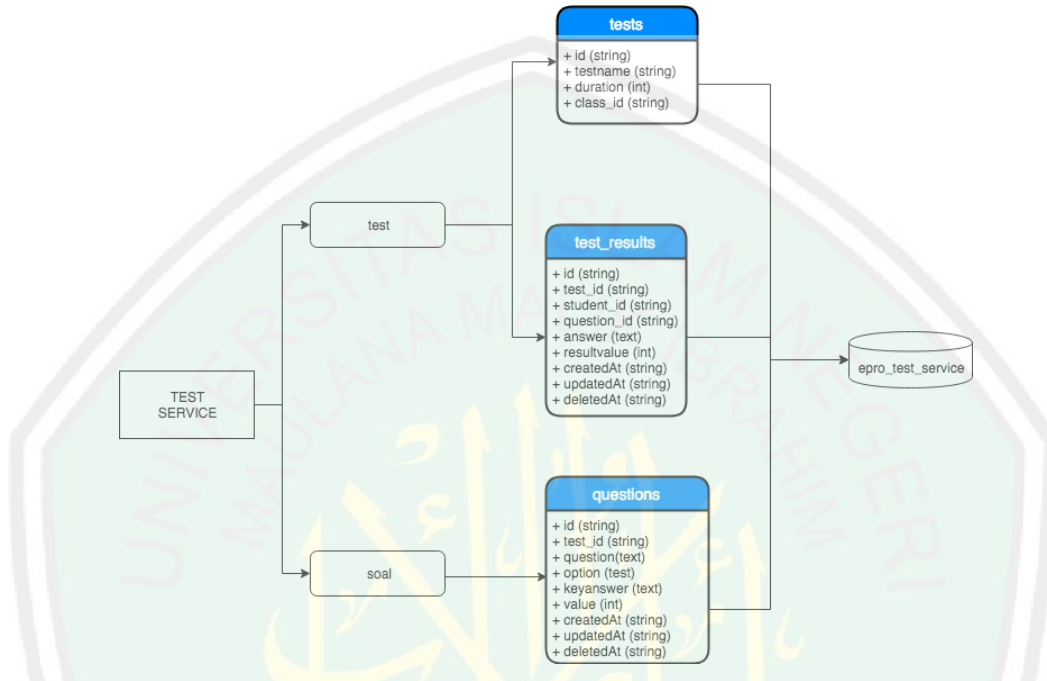
b. *class-service*



Gambar 3.5 *class-service*

Pada gambar 3.5 terlihat bahwa *class-service* terdiri dari 1 *subdomain*, yaitu *class*.

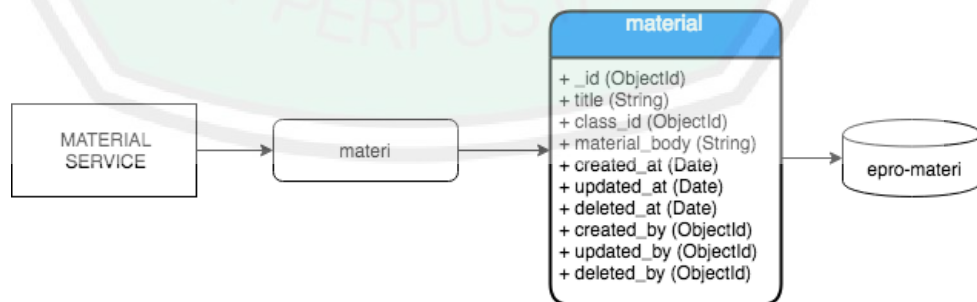
c. *test-service*



Gambar 3.6 test-service

Pada gambar 3.6 terlihat bahwa *test-service* terdiri dari 2 *subdomain*, yaitu *test* dan *soal*.

d. *material-service*



Gambar 3.7 material-service

Pada gambar 3.7 terlihat bahwa *material-service* terdiri dari 1 *subdomain*, yaitu *materi*.

3.3 Analisis Proses Bisnis

3.3.1 Identifikasi Proses Bisnis

Identifikasi proses bisnis bertujuan untuk mengidentifikasi kegiatan bisnis yang berhubungan dengan sistem yang dibangun. Identifikasi proses bisnis dalam sistem *e-learning* terdiri dari :

1. *Login* sistem
2. Pengolahan *user*
3. Pengolahan *class*
4. Pembuatan *test*
5. Pengolahan materi

3.3.1.1 Analisis Proses Bisnis

Merupakan cara untuk menilai efektivitas proses bisnis utama untuk bidang bisnis yang lebih spesifik dan tujuan bisnis. Analisa proses bisnis terdiri dari :

- a. Nama Proses Bisnis : merupakan nama proses bisnis di dalam pembangunan sistem *e-learning*.
- b. Siapa yang Terlibat : adalah pihak-pihak yang terkait dalam proses bisnis pembelajaran *e-learning*.
- c. Dimana Proses Bisnis Terjadi : merupakan tempat dimana proses bisnis berlangsung.
- d. Kapan Proses Bisnis Terjadi : merupakan waktu terjadinya proses bisnis *e-learning*.

- e. Bagaimana Proses Bisnis Dilakukan : merupakan cara proses bisnis *e-learning* dilakukan pihak yang terkait.

Dokumen yang Terkait : merupakan semua dokumen yang berkaitan dengan berlangsungnya proses bisnis yang terjadi.

Analisis proses bisnis dalam penelitian merujuk pada tabel 3.1 di bawah :



Tabel 3.1 Analisis Proses Bisnis

No	Nama Proses Bisnis	Siapa yang Terlibat	Dimana Proses Bisnis Terjadi	Kapan Proses Bisnis Terjadi	Bagaimana Proses Bisnis Dilakukan	Dokumen yang Terkait
1.	Login sistem	• Siswa • Guru • Admin	Menyesuaikan	Saat akan masuk sistem <i>e-learning</i>	▪ Siswa, guru dan operator memasukkan <i>username</i> dan <i>password</i> saat akan masuk sistem <i>e-learning</i>.	• <i>Fom Login</i>
2.	Pengolahan User	• Siswa • Guru • Admin	Ruang Admin Probistek	Saat ada perubahan user	▪ Calon siswa datang ke Probistek menuju tempat pendaftaran ▪ Admin memberikan formulir pendaftaran ▪ Calon siswa mengisi formulir ▪ Calon siswa membayar biaya daftar ulang ▪ Admin mengisi <i>form</i> registrasi	• Formulir pendaftaran • <i>Form</i> Registrasi
3.	Pengolahan Class	• Siswa • Guru • Admin	Menyesuaikan	Apabila terdapat program keahlian	▪ Admin mengelola program keahlian	• <i>Form</i> Program Keahlian
4.	Pengolahan Test	• Guru • Siswa	Saat diadakan <i>test</i>	Sesuai jadwal yang ditentukan	▪ Guru membuat <i>assignment</i> ▪ Guru membuat soal ▪ Siswa mengerjakan soal	• <i>Form Assignment</i> • <i>Form Soal</i>
5.	Manajemen materi	• Guru • Siswa	Menyesuaikan	Saat ada materi baru	▪ Guru membuat materi ▪ Siswa membaca materi	• <i>Form Materi</i>

3.3.2 Identifikasi dan Analisis Kebutuhan

Tahapan dalam identifikasi dan analisis kebutuhan dibedakan menjadi dua, yaitu :

1. Identifikasi dan analisis kebutuhan fungsional, yaitu penjelasan secara detail terkait kebutuhan sistem dan kegiatan yang dilakukan pihak yang terlibat dalam pembangunan sistem *e-learning*. Identifikasi kebutuhan fungsional dapat dilihat pada tabel-tabel berikut ini :

- a. Data Siswa

Tabel 3.2 Data Siswa

Calon Siswa	Pihak Admin	Kebutuhan Fungsional
Menerima formulir pendaftaran	Memberikan formulir pendaftaran	Menampilkan formulir pendaftaran
Mengisi formulir		Menerima <i>entry</i> data dalam formulir pendaftaran

- b. Data *Guru*

Tabel 3.3 Data *Guru*

Pihak Admin	Kebutuhan Fungsional
Membuat dan mengelola data <i>guru</i>	Menampilkan <i>form</i> data <i>guru</i>

- c. Data Program Keahlian

Tabel 3.4 Data Program Keahlian

Pihak Admin	Kebutuhan Fungsional
Membuat dan mengelola data program keahlian	Menampilkan <i>form</i> data program keahlian

d. Data Materi

Tabel 3.5 Data Materi

Pihak Guru	Pihak Admin	Kebutuhan Fungsional
Membuat dan mengelola data materi	Memonitor data materi	Menampilkan <i>form</i> data materi

e. Data Test

Tabel 3.6 Data Test

Pihak Guru	Pihak Operator	Kebutuhan Fungsional
Membuat dan mengelola data <i>assignment</i>	Memonitor data <i>test</i>	Menampilkan <i>form assignment</i>
Membuat dan mengelola data soal	Memonitor data soal	Menampilkan <i>form</i> soal

2. Identifikasi dan analisis kebutuhan nonfungsional, adalah informasi kebutuhan sistem dan komponen yang diperlukan yang terlibat dalam pengembangan sistem. Identifikasi dan analisis kebutuhan fungsional dapat dilihat pada tabel 3.7 di bawah ini :

Tabel 3.7 Identifikasi dan Analisis Kebutuhan Non-fungsional

Komponen SI	Spesifikasi	Siapa yang mengadakan	Kapan diadakan	Dimana diadakan	Bagaimana pengadaannya
Hardware					
<i>Server DBMS & Application</i>	- RAM 4 GB - SSD 25 GB	Peneliti	Akan melakukan pengujian sistem	<i>Google Cloud Platform</i>	Menentukan anggaran biaya
Software					
<i>Runtime</i>	<i>Node.js</i>	Peneliti	Awal pembuatan Sistem	Laptop Peneliti	<i>Download</i>
<i>DBMS</i>	<i>MongoDB</i> <i>MySQL</i>	Peneliti	Awal pembuatan Sistem	Laptop Peneliti	<i>Download</i>
<i>Pemodelan</i>	<i>Balsamiq Mock Up,</i> <i>Draw.io, Power Designer</i>	Peneliti	Awal Pembuatan Sistem	Laptop Peneliti	<i>Download</i>
<i>Editor</i>	<i>IntelIJ, Sublime Text</i>	Peneliti	Awal Pembuatan Sistem	Laptop Peneliti	<i>Download</i>
<i>Desain ERD dan DFD</i>	<i>Power Designer</i>	Peneliti	Awal Pembuatan Sistem	Laptop Peneliti	<i>Download</i>
<i>Browser</i>	<i>Google Chrome</i>	Peneliti	Awal Pembuatan Sistem	Laptop Peneliti	<i>Download</i>

Komponen SI	Spesifikasi	Siapa yang mengadakan	Kapan diadakan	Dimana diadakan	Bagaimana pengadaannya
<i>Sitem Operasi</i>	<i>Linux Ubuntu</i>	Peneliti	Awal pembuatan Sistem	Laptop Peneliti	<i>Download</i>
<i>Testing API</i>	<i>Postman</i>	Peneliti	Awal pembuatan Sistem	Laptop Peneliti	<i>Download</i>
<i>HTTP Reserve Proxy</i>	<i>NGINX</i>	Peneliti	Awal pembuatan Sistem	Laptop Peneliti	<i>Download</i>



3.4 Desain Sistem

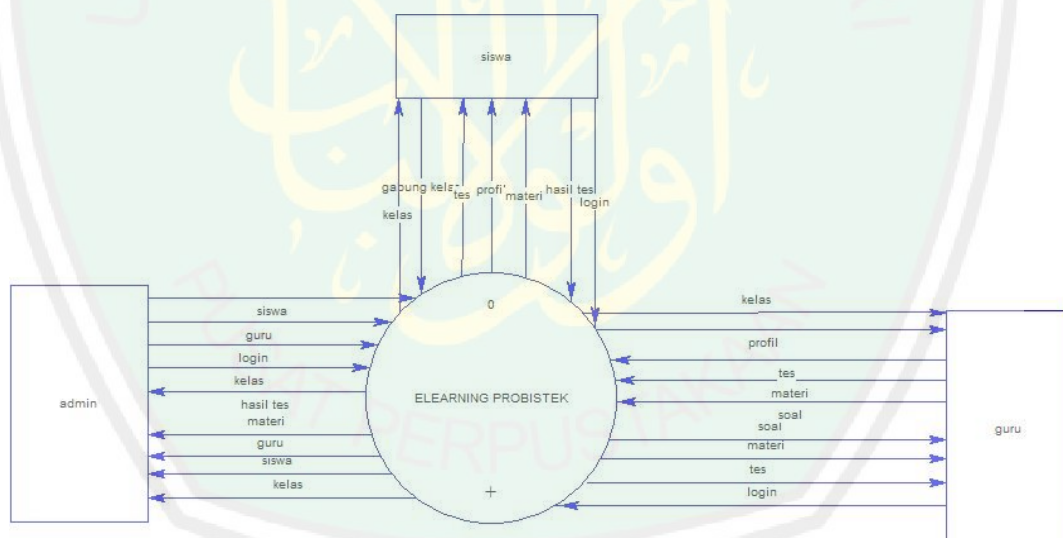
3.4.1 Pemodelan Sistem

Pemodelan proses bisnis merupakan bagian dari *business rules* dan alur bisnis. Pemodelan berfungsi agar bisnis berjalan sesuai dengan strategi dan target. Pemodelan sistem *e-learning* ini didesain dengan model, yaitu *DFD* (*Data Flow Diagram*), *CDM* (*Contextual Data Model*), *PDM* (*Physical Data Model*), *BPMN* (*Business Process Model and Notation*), dan *schema database*.

a. Data Flow Diagram (DFD)

1. Context Diagram

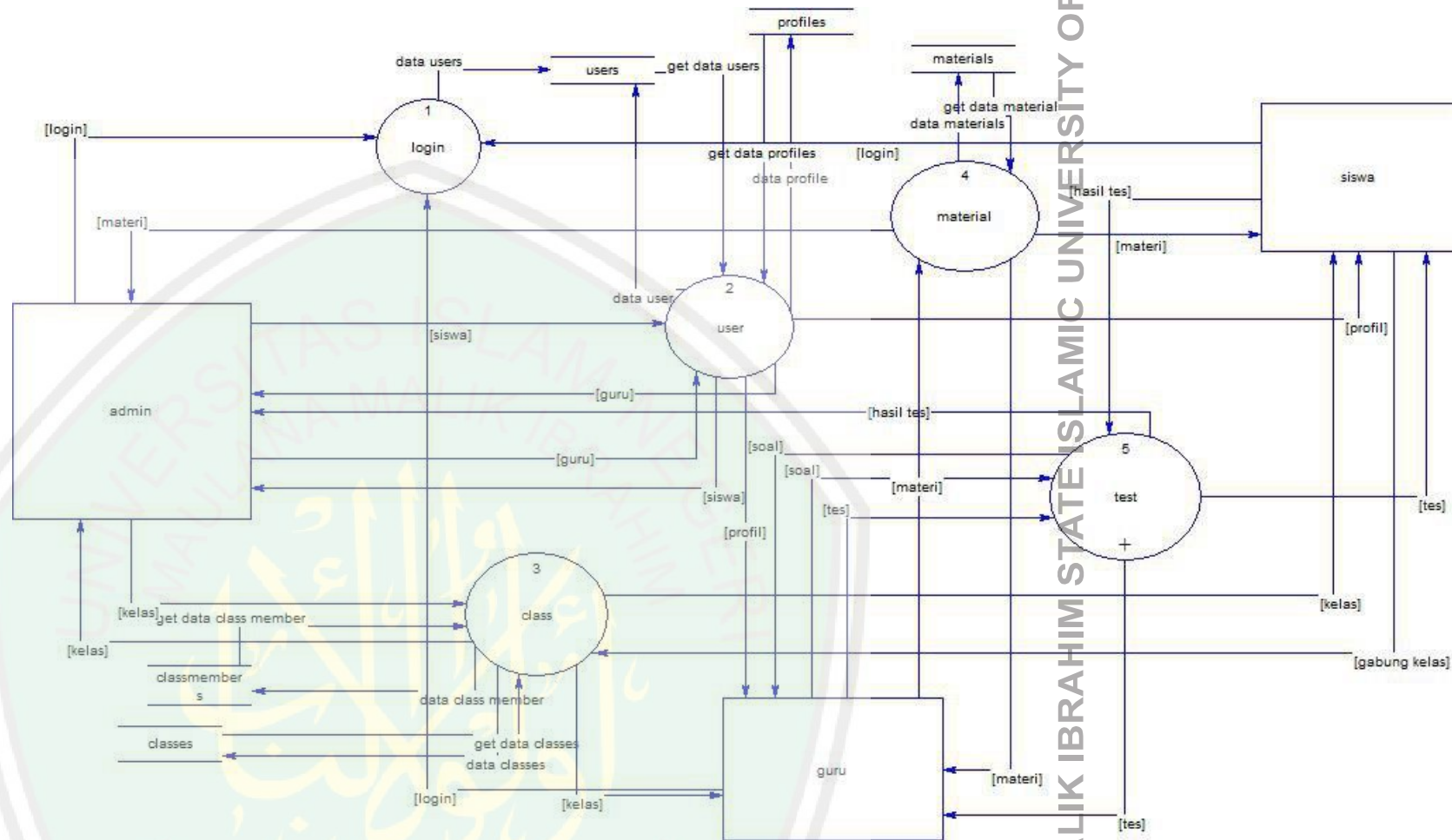
Context diagram dalam sistem *e-learning* ini mempunyai 3 *external entity*, yaitu admin, guru dan siswa



Gambar 3.8 Context Diagram

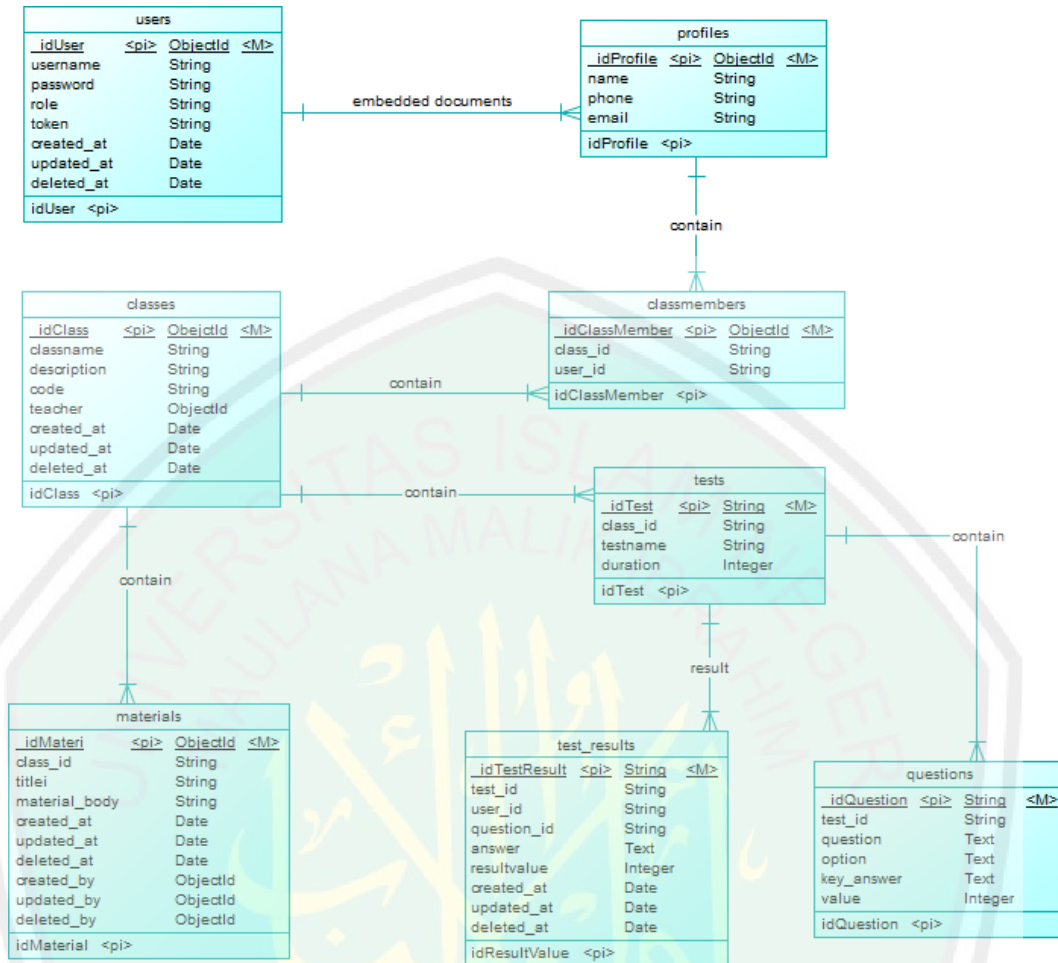
2. DFD Level 1

Merupakan hasil *decompose* dari *context diagram* pada gambar 3.8 dimana sistem di *breakdown* lebih rinci.



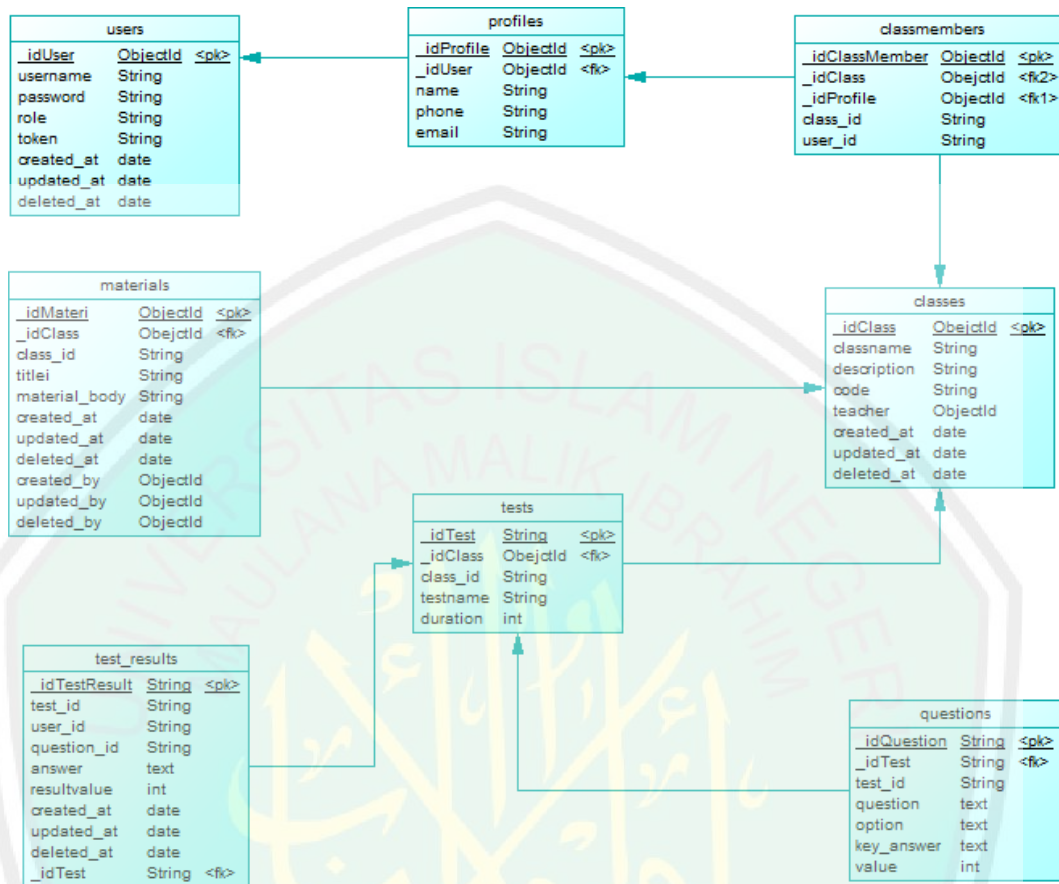
Gambar 3.9 DFD Level 1

Pada penelitian ini menggunakan 2 DBMS, yaitu *MongoDB* dan *MySQL*. *MongoDB* diterapkan pada *auth-service*, *class-service*, *material-service* karena *MongoDB* mempunyai performa yang ringan untuk kebutuhan akses data sehingga tidak memerlukan *response time* yang lama mengingat ketiga *services* tersebut sering diakses. Selain itu *field* pada *MongoDB* bersifat dinamis sehingga memudahkan *developer* apabila ada perubahan di kemudian hari. Namun *MongoDB* tidak mendukung *transaction* karena tidak adanya fitur *rolled back* data sehingga untuk *test-service* menggunakan *MySQL*.



Gambar 3.11 CDM e-learning

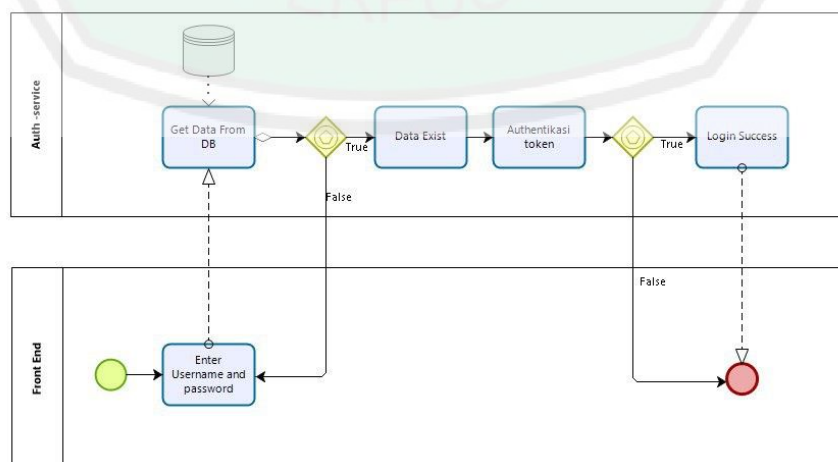
c. PDM (Physical Data Model)



Gambar 3.12 PDM e-learning

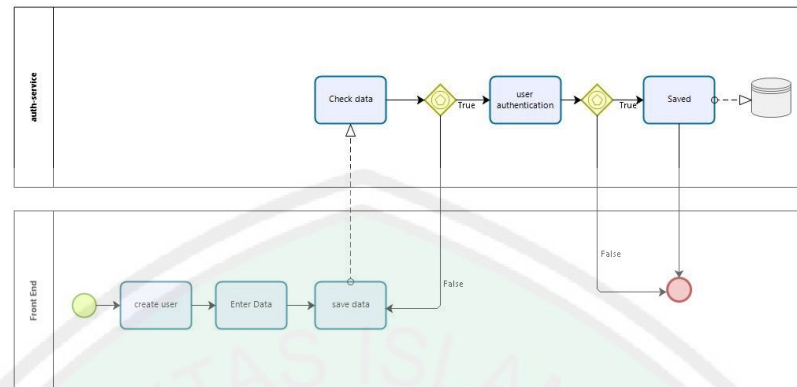
d. BPMN

- login



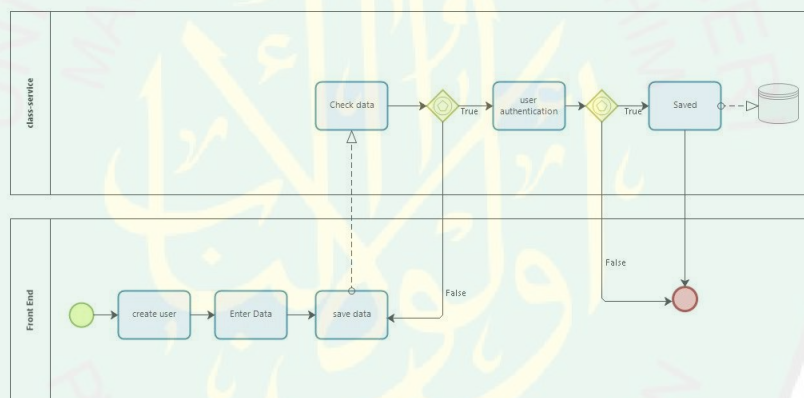
Gambar 3.13 BPMN login

- create user



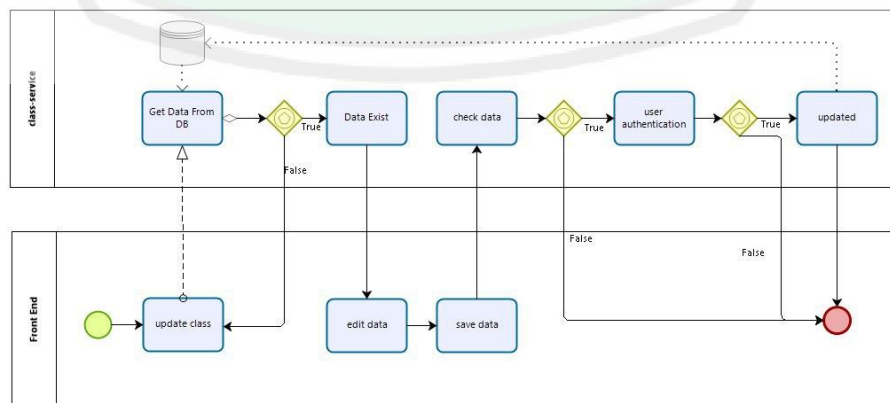
Gambar 3.14 BPMN create user

- create class



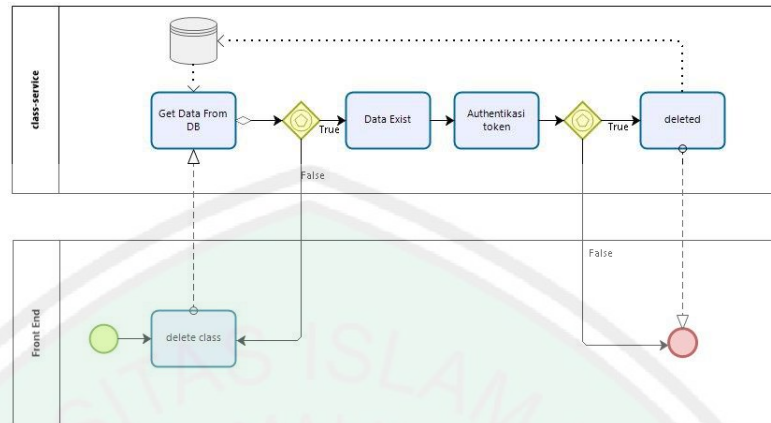
Gambar 3.15 BPMN create class

- update class



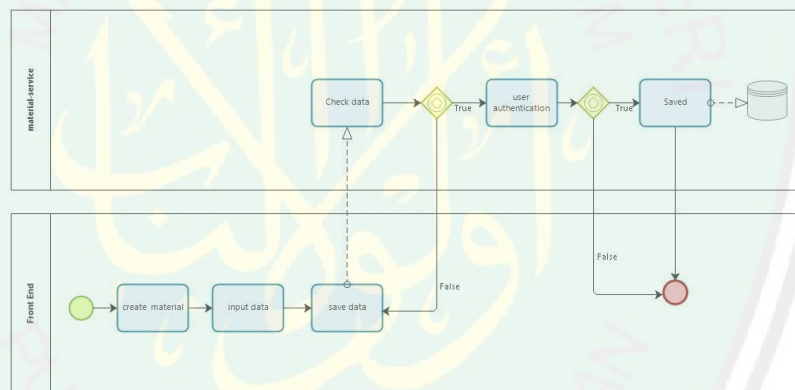
Gambar 3.16 BPMN update class

- delete class



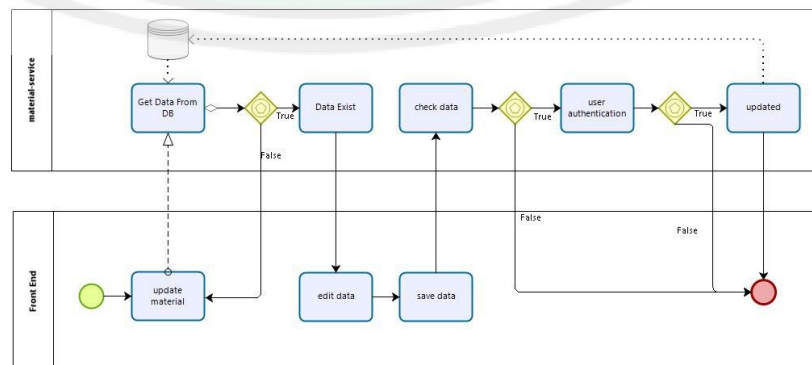
Gambar 3.17 BPMN delete class

- create material



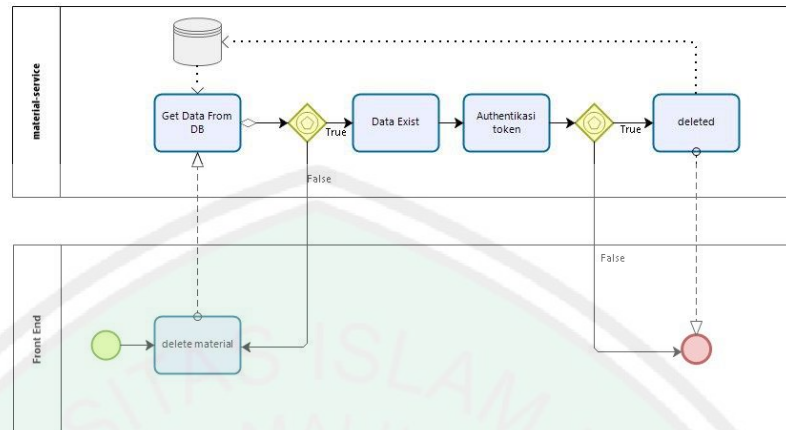
Gambar 3.18 BPMN create material

- update material



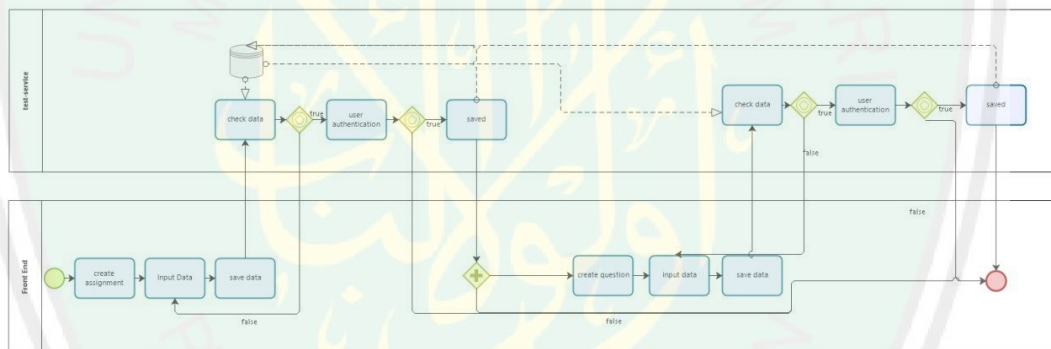
Gambar 3.19 BPMN update material

- delete material



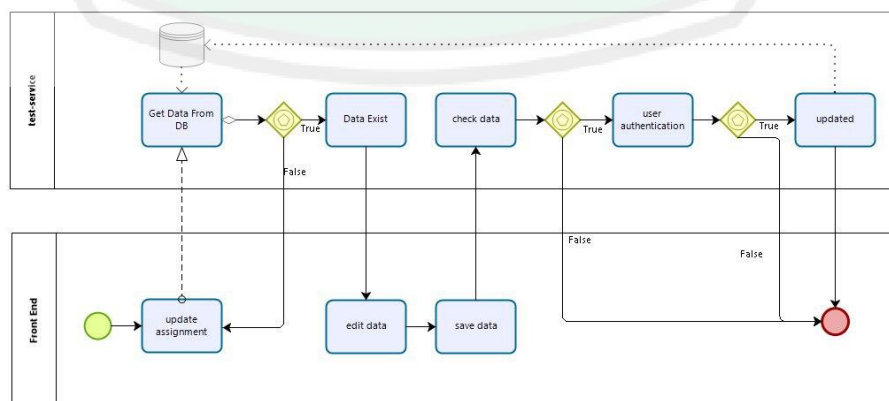
Gambar 3.20 BPMN delete material

- create test



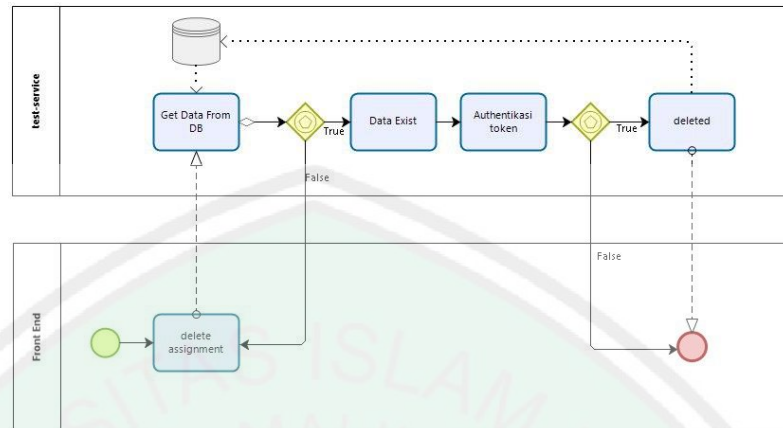
Gambar 3.21 BPMN create test

- update test



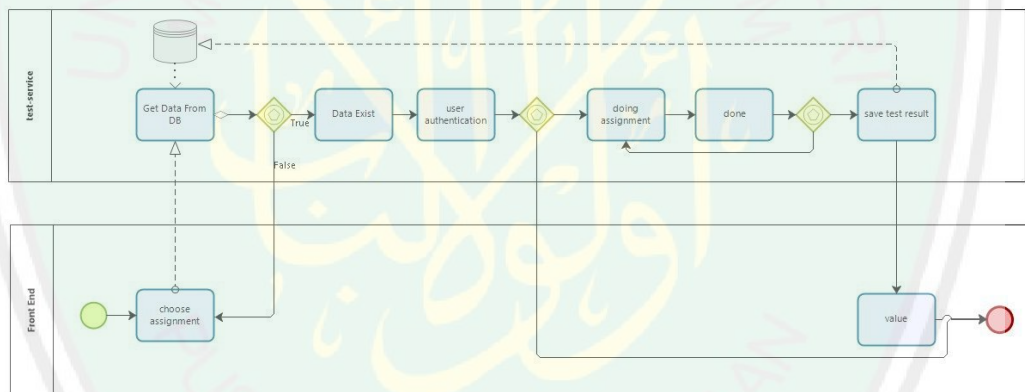
Gambar 3.22 BPMN update test

- delete test



Gambar 3.23 BPMN delete test

- doing test



Gambar 3.24 BPMN doing test

3.4.2 Desain Interface

1. Desain Input

a. Login Sistem

A Web Page
http://68.183.187.196/login

e-Pro
e-learning Probistek

Please Login First

Username

Password

Gambar 3.25 Form Login

b. Registrasi Siswa dan Guru

A Web Page
http://e-pro.ac.id/user/register

e-Pro
e-learning Probistek

Tambah User

Nama

Telepon

email

Username

Password

Gambar 3.26 Form Registrasi

c. Input Data Program Keahlian

A Web Page
http://e-pro.ac.id/program_keahlian

e-Pro
e-learning Probistek

Tambah Program Keahlian

Nama Program Keahlian

Pengajar

Deskripsi

Gambar 3.27 Form Program Keahlian

d. *Input Data Materi*

Gambar 3.28 Form Materi

e. *Input Data Assignment*

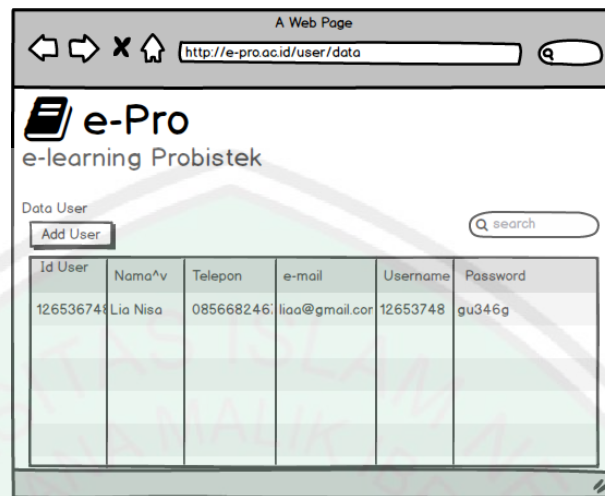
Gambar 3.29 Form Assignment

f. *Input Data Soal*

Gambar 3.30 Form Soal

2. Desain Output

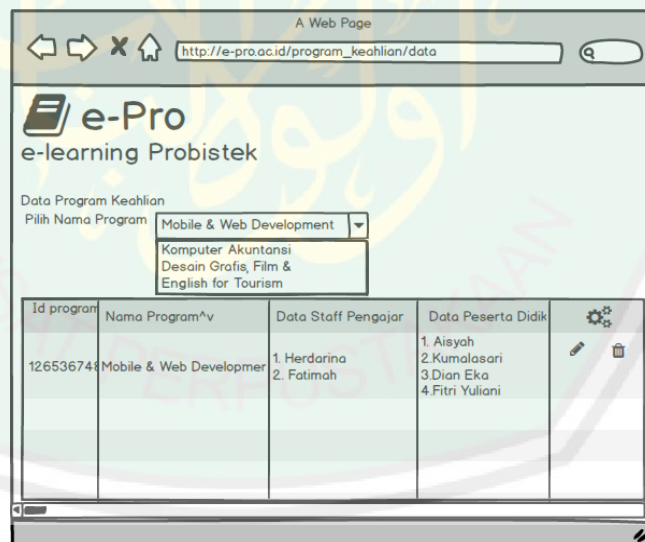
a. List User



Gambar 3.31 List User

b. List Data Program Keahlian

- Admin



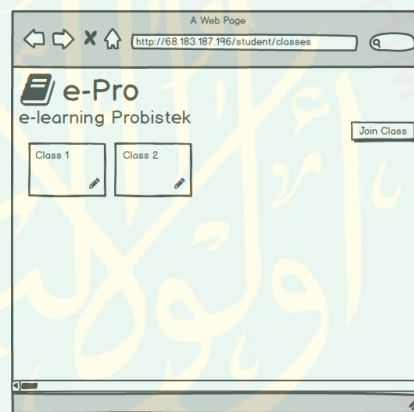
Gambar 3.32 Data Program Keahlian Admin

- Teacher



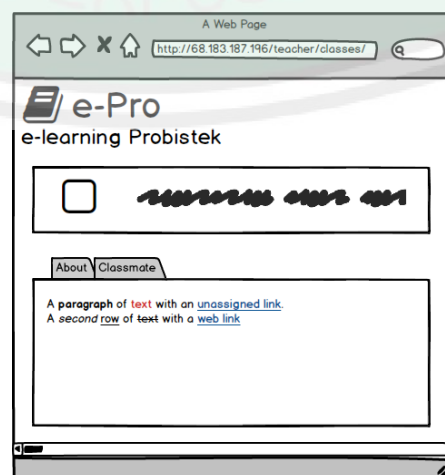
Gambar 3.33 Data Program Keahlian Teacher

- Student



Gambar 3.34 Data Program Keahlian Student

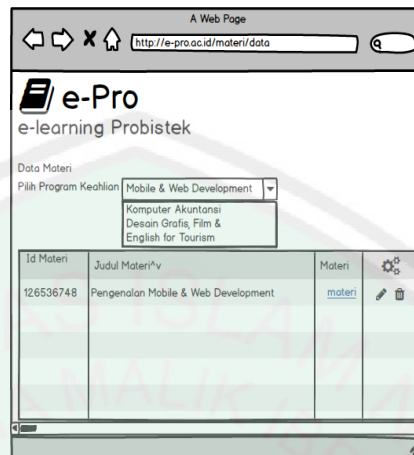
c. Detail Program Keahlian



Gambar 3.35 Detail Program Keahlian

d. *Output Data Materi*

- Admin



Gambar 3.36 Data Materi Admin

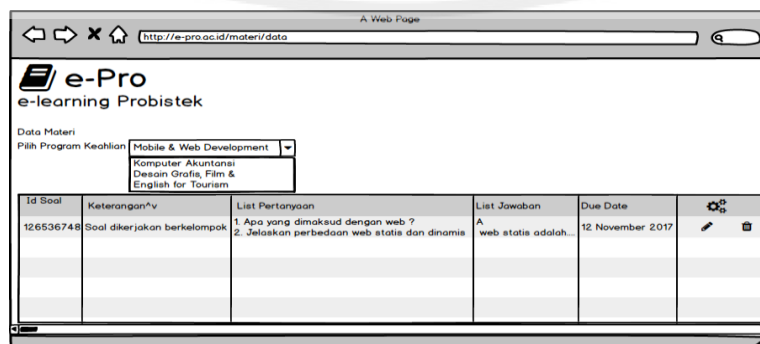
- Siswa dan Guru



Gambar 3.37 Data Materi

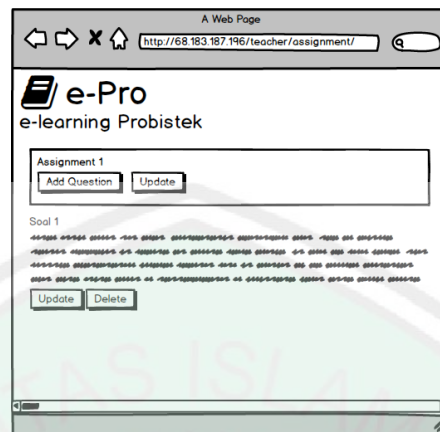
e. *Output Data Soal*

- Admin



Gambar 3.38 Data Soal Admin

- Siswa dan Guru



Gambar 3.39 Data Soal

- f. *Output Test*



Gambar 3.40 Test

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi Sistem

Adalah proses penerapan hasil rancangan ke dalam sistem yang dibangun sesuai dengan perancangan yang telah dibuat sebelumnya pada bab III. Teknologi dan sistem yang dibangun berbasis *website*. Berikut spesifikasi perangkat lunak (*software*) dan perangkat keras (*hardware*) untuk implementasi sistem :

A. Kebutuhan Hardware

1. *Processor : Intel core i3*
2. *RAM : 4 GB*
3. *HDD : 500 GB*

B. Kebutuhan Software

1. *OS : OS X El Capitan version 10.11.6 dan Microsoft Windows 10*
2. *Browser : Google Chrome version 68.0.3440.106.*
3. *IDE : IntelliJ version 2017.2, PhpStorm version 2018.2*
4. *Pemodelan : Balsamiq Mock Up, Draw.io, Power Designer*
5. *DBMS : MongoDB 3.4.10, MySQL 5.7*
6. *Uji coba REST : Postman, Insomnia*
7. *Tools Manajemen MongoDB : Robo 3T 1.1*
8. *Tools Manajemen MySQL : Sequel Pro 1.1.2*
9. *Cloud : Google Cloud Platform*

4.2 Implementasi *Microservices* dan *Domain Driven Design*

1. *auth-service*

Seperti yang telah dijelaskan pada bab III, *auth-service* terdiri dari *subdomain teacher*, *student* dan *user*. Kemudian *layers* DDD dibedakan menjadi 4 yaitu *Application*, *Domain* dan *Infrastructure* dan *Interface*. Berikut implementasi struktur *layer* dari *auth-service*.



Gambar 4.1 *structure auth-service*

Service ini juga berperan sebagai *api-gateway* yang bertugas menjadi gerbang lalu lintas *request* ke semua *service*. Sehingga *auth-service* mempunyai wewenang untuk memvalidasi semua *request*. Berikut proses *authentikasi* API Gateway. Semua *request* harus melalui *auth-service*, sehingga *route* untuk semua *services* harus terdaftar di *auth-service*.

```

import ...

const RouterInstance = require('restify-router').Router;
const router = new RouterInstance();

router.get("/", (req, res) => {
  res.json("API Gateway UP!");
});

/** Class routers ...*/
router.get("/admin/class*", [AdminMiddleware, ClassHandler]);
router.put("/admin/class*", [AdminMiddleware, ClassHandler]);
router.post("/admin/class*", [AdminMiddleware, ClassHandler]);

router.get("/student/class*", [StudentMiddleware, ClassHandler]);
router.post("/student/class*", [StudentMiddleware, ClassHandler]);

router.add("/teacher", require("./Teacher"));
router.get("/teacher/class*", [TeacherMiddleware, ClassHandler]);
router.put("/teacher/class*", [TeacherMiddleware, ClassHandler]);
router.post("/teacher/class*", [TeacherMiddleware, ClassHandler]);

/** Test routers ...*/
router.get("/admin/test*", [AdminMiddleware, TestHandler]);
router.put("/admin/test*", [AdminMiddleware, TestHandler]);
router.post("/admin/test*", [AdminMiddleware, TestHandler]);

router.get("/student/test*", [StudentMiddleware, TestHandler]);
router.put("/student/test*", [StudentMiddleware, TestHandler]);
router.post("/student/test*", [StudentMiddleware, TestHandler]);

router.get("/teacher/test*", [TeacherMiddleware, TestHandler]);
router.put("/teacher/test*", [TeacherMiddleware, TestHandler]);
router.post("/teacher/test*", [TeacherMiddleware, TestHandler]);

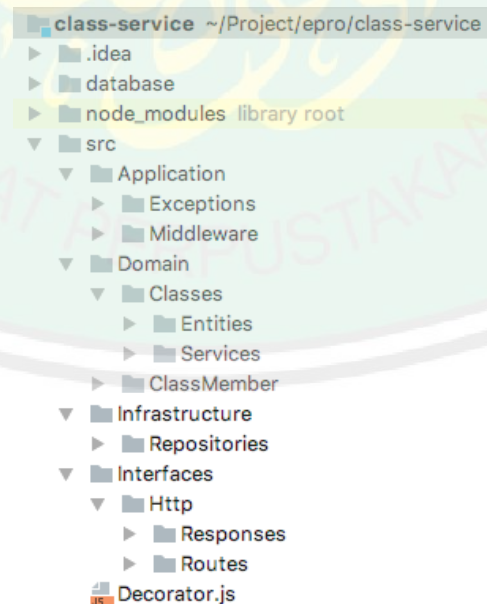
router.get("/teacher/question*", [TeacherMiddleware, TestHandler]);
router.put("/teacher/question*", [TeacherMiddleware, TestHandler]);

```

Gambar 4.2 Autentikasi API Gateway

2. class-service

Service ini memiliki 1 subdomain, yaitu subdomain class. Struktur kode class-service dapat dilihat pada gambar 4.3 dibawah ini.



Gambar 4.3 Struktur class-service

Class-service dapat berkomunikasi dengan service yang lain atau disebut dengan *cross service* melalui protokol *HTTP*. Berikut ini adalah contoh *class-service* berkomunikasi dengan *auth-service* untuk mendapatkan data guru.

```

/** Get teacher detail ...*/
async getTeacher(teacher_id) {
  const result = await rp(`${process.env.AUTH_SERVICE_URI}/teacher/${teacher_id}`, {
    json: true
  });
  return result.data;
}

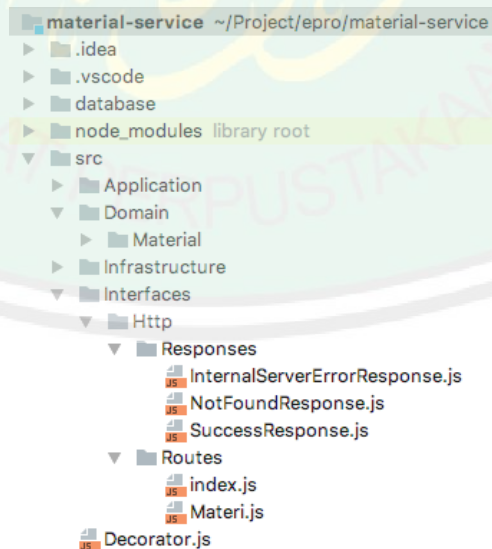
/** Get list teacher ...*/
async getAllTeacher() {
  const result = await rp(`${process.env.AUTH_SERVICE_URI}/teacher/all`, {
    json: true
  });
  return result.data;
}

```

Gambar 4.4 Komunikasi *class-service* ke *auth-service*

3. *material-service*

Material-service merupakan *service* untuk proses bisnis pengolahan materi dan memiliki 1 *subdomain*, yaitu materi. Berikut ini adalah struktur kode dari *material-service*.



Gambar 4.5 Struktur Kode *material-service*

4. test-service

Test-service terdiri dari 2 *subdomain*, yaitu *subdomain test* dan *soal*. *Subdomain test* berkaitan dengan pengolahan *test* atau *assignment*. Sedangkan *subdomain soal* berkaitan dengan pengolahan *soal* atau *question*. *Subdomain test* merupakan wadah atau induk dari *subdomain soal*. Sehingga sebelum membuat *soal* harus membuat *test* dahulu pada *subdomain test*. Berikut struktur kode dari *test-service*.



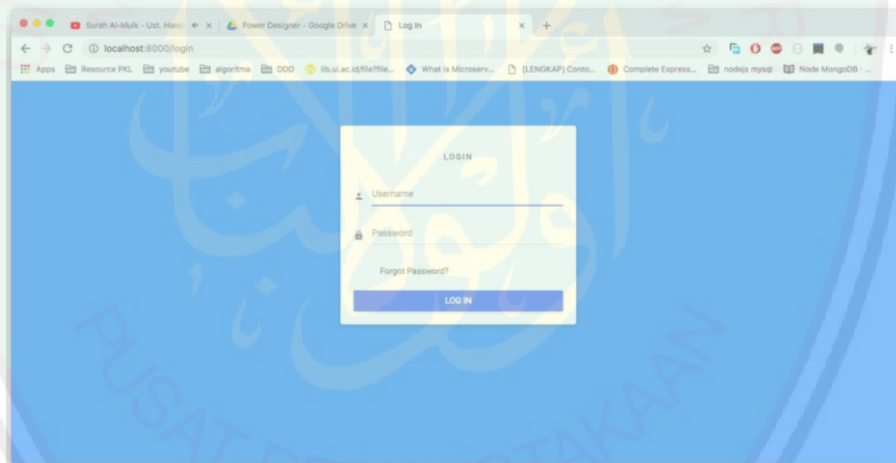
Gambar 4.6 Struktur Kode *test-service*

4.3 Implementasi Antarmuka Aplikasi

Pada implementasi antarmuka Sistem Informasi *e-Learning Probistek* ini berbeda-beda, disesuaikan dengan *role* pengguna. *Role* dibagi menjadi 3 bagian, yaitu *admin*, *teacher* dan *student*. Berikut implementasi *user interface* pada Sistem Informasi *e-Learning Probistek* :

a. Halaman *Login*

Halaman *login* merupakan halaman yang pertama muncul pada saat sistem diakses. Pada halaman ini terdapat dua *field*. *Field* pertama untuk *input username* dan yang kedua untuk *input password*. *View login* dapat dilihat pada gambar 4.7. *Username* dan *password* diproses pada *auth-service* yang akan divalidasi, kemudian apabila data valid akan menghasilkan token dan tampilan dialihkan ke halaman beranda sesuai dengan role *user*. Token berfungsi untuk menyimpan role sebagai *session* yang akan digunakan untuk mengakses data. Gambar 4.8 merupakan *authentikasi login* sedangkan gambar 4.9 merupakan *generate token*.



Gambar 4.7 Halaman *Login*

```
import ...

export class UserLogin {

  async login(username, password) {
    const user = await new UserRepository().findByCredential(username, password);
    const token = new TokenGenerator().generate({
      "id": user._id,
      "reference_id": (user[user.role.toLowerCase()] !== undefined) ? user[user.role.toLowerCase()]._id : null,
      "name": user.name,
      "role": user.role
    });

    this.storeToken(user._id, token);

    return {
      id: user._id,
      name: user.name,
      email: (user[user.role.toLowerCase()] !== undefined) ? user[user.role.toLowerCase()].email : null,
      phone: (user[user.role.toLowerCase()] !== undefined) ? user[user.role.toLowerCase()].phone : null,
      username: user.username,
      role: user.role,
      token: token
    };
  }

  storeToken(user_id, token) {
    UserLoginEntity.update(
      { user_id: user_id },
      { user_id: user_id, token: token },
      { upsert: true, setDefaultsOnInsert: true }
    );
  }
}
}
```

Gambar 4.8 Autentikasi Login

```
import JWT from "jsonwebtoken";

export class TokenGenerator {

  /**
   * TokenGenerator constructor
   */
  constructor() {
    /**
     * Load .env
     */
    require("dotenv").config({
      path: "../../.env"
    });
  }

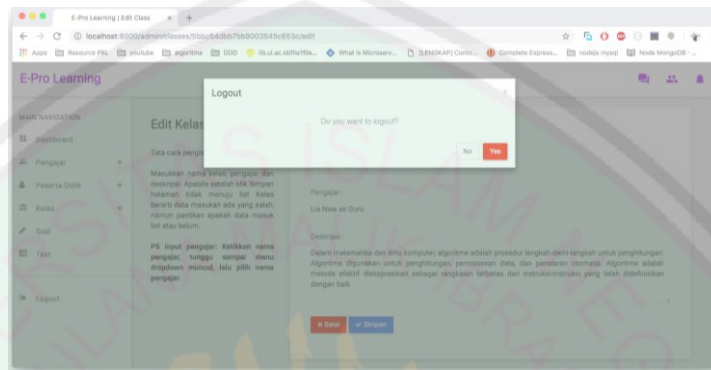
  /**
   * Generate token
   * @param data
   * @returns {*}
   */
  generate(data) {
    return JWT.sign({
      data: data
    }, process.env.JWT_SECRET, {
      expiresIn: "1d"
    });
  }
}
```

Gambar 4.9 Generate Token

Dalam pembuatan token telah dijelaskan bahwa *expiresIn*: "1d", yang berarti token hanya berlaku dalam waktu sehari, apabila lebih dari 24 maka pengguna harus *login* lagi untuk mendapatkan token yang baru agar bisa mengakses sisten.

b. Halaman *Logout*

Adalah tampilan saat pengguna ingin keluar dari sistem. Tampilan *logout* dapat dilihat pada gambar 4.10 dibawah. Apabila user mengklik Yes maka sistem akan keluar.



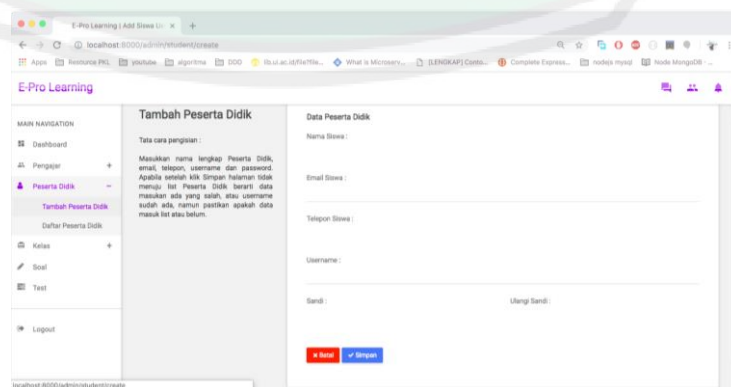
Gambar 4.10 Halaman *Logout*

c. Halaman *User*

Halaman ini hanya dapat diakses oleh administrator. Pada halaman ini terdapat fitur untuk *registrasi* dan *list* guru serta siswa .

- Tambah Siswa

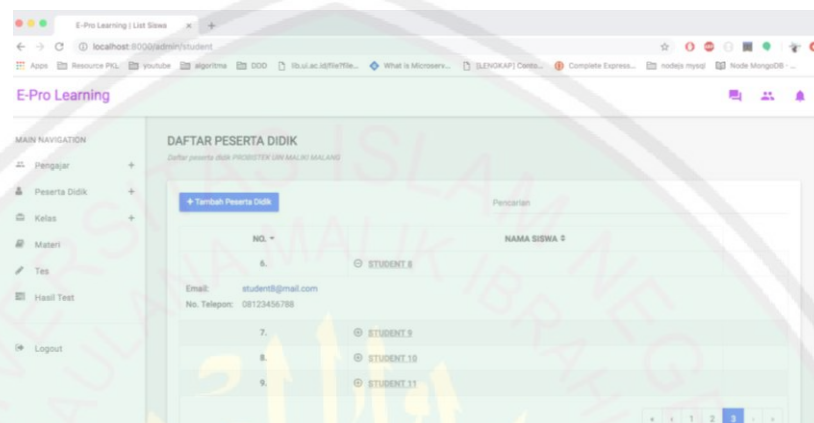
Setelah administrator berhasil melakukan registrasi siswa, tampilan akan *redirect* ke *list* siswa seperti gambar 4.12.



Gambar 4.11 Halaman Tambah Siswa

- List Siswa

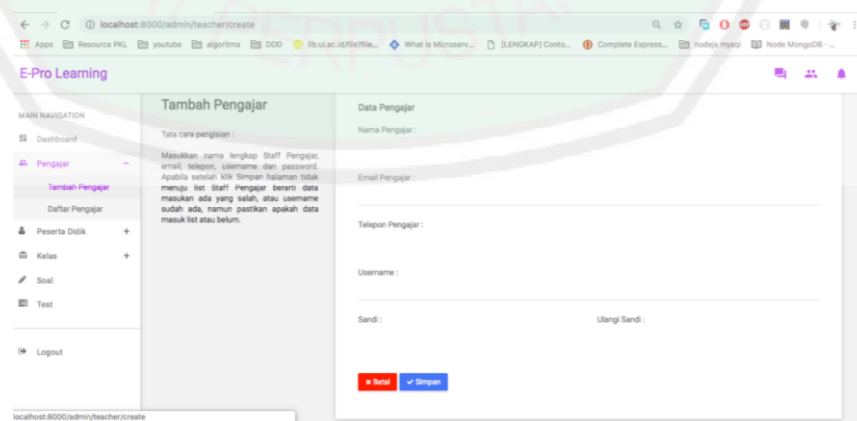
Administrator dapat melihat detail siswa dengan mengklik nama siswa pada gambar 4.12.



Gambar 4.12 Halaman List Siswa

- Tambah Guru

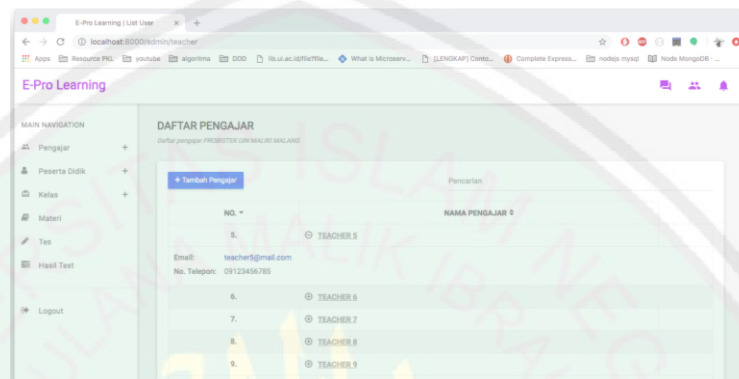
Registrasi guru pada dasarnya sama seperti *registrasi* pada siswa, setelah administrator berhasil melakukan *registrasi* tampilan akan *redirect* ke *list* siswa seperti gambar 4.13 di bawah ini.



Gambar 4.13 Halaman Tambah Guru

- List Pengajar

Administrator dapat melihat detail siswa dengan mengklik nama siswa pada gambar 4.14.



Gambar 4.14 Halaman List Guru

d. Halaman Program Keahlian / Kelas

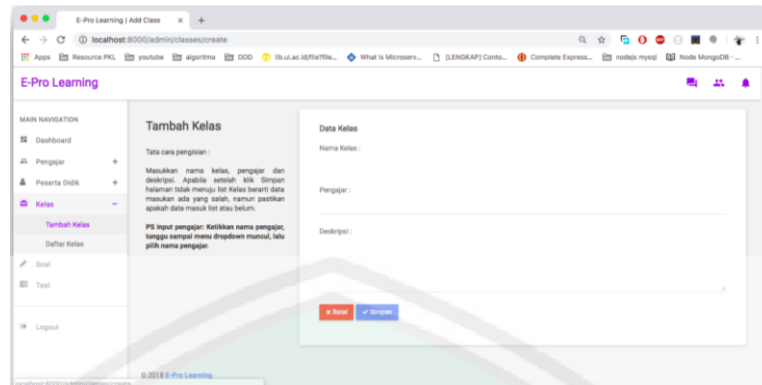
Program keahlian merupakan bagian dari *class-service*. Agar *client* dapat mengakses *class-service* maka perlu divalidasi terlebih dahulu oleh *auth-service*. *Auth-service* juga berperan sebagai *api-gateway* sehingga semua *request* dari *client* ke *service* lain harus melalui *auth-service* terlebih dahulu. *Authentikasi* dan *gateway* dari *auth-service* menuju *class-service* dapat dilihat pada gambar 4.2.

Halaman kelas ini terdapat 3 role yang berbeda dimana antara role *admin* dan guru atau siswa memiliki tampilan yang berbeda.

1. Admin

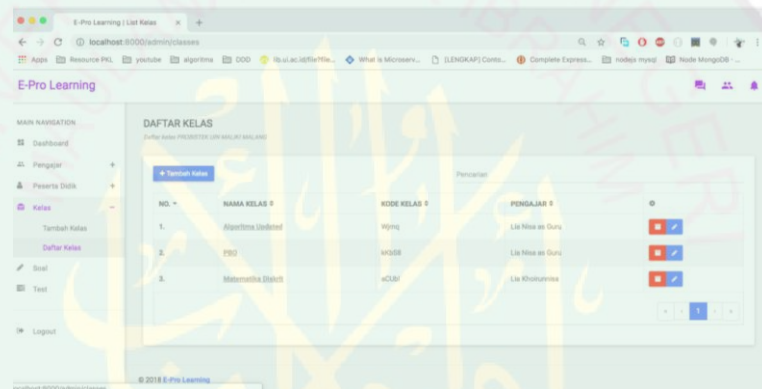
- Tambah Kelas

Halaman tambah kelas merupakan halaman untuk membuat kelas baru. Setelah berhasil membuat kelas baru, halaman menuju ke *list* kelas.



Gambar 4.15 Halaman Tambah Kelas

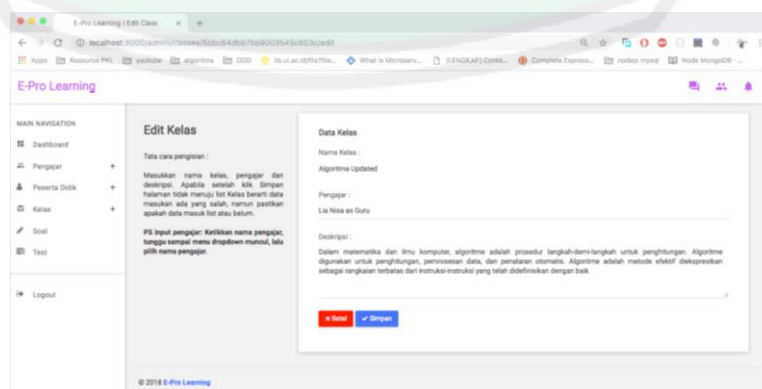
- List Kelas



Gambar 4.16 Halaman List Kelas

- Edit Kelas

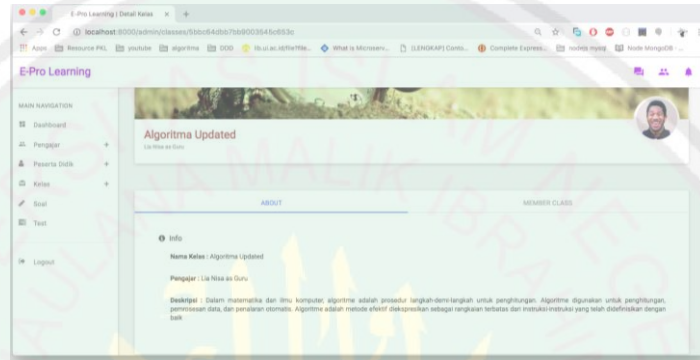
Apabila mengklik icon pencil yang terlihat pada gambar 4.16 maka akan menuju ke halaman edit kelas.



Gambar 4.17 Halaman Edit Kelas

- Detail Kelas

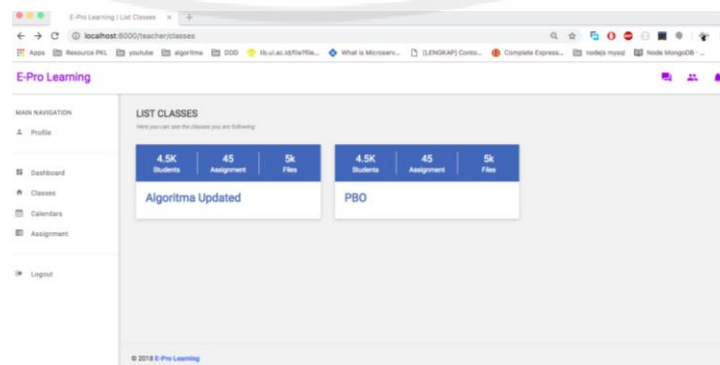
Apabila nama kelas pada list kelas gambar 4.16 di klik maka akan menuju ke halaman detail kelas. Di bagian detail kelas ada 2 menu tab, yaitu *about* dan *class member*. *About* berisi info kelas tersebut dan *member class* berisi tentang siswa yang tergabung dalam kelas tersebut.



Gambar 4.18 Halaman Detail Kelas

2. Guru

Pada halaman ini akan menampilkan kelas dimana guru tersebut tergabung dalam kelas tersebut. Apabila kelas di klik akan menampilkan detail kelas seperti detail kelas pada role administrator. Namun bedanya disini ada 3 tab, *about* yaitu berisi info dari kelas tersebut, *student* yaitu siswa yang tergabung dalam kelas tersebut dan *result assignment* yaitu hasil dari tugas yang dikerjakan siswa.

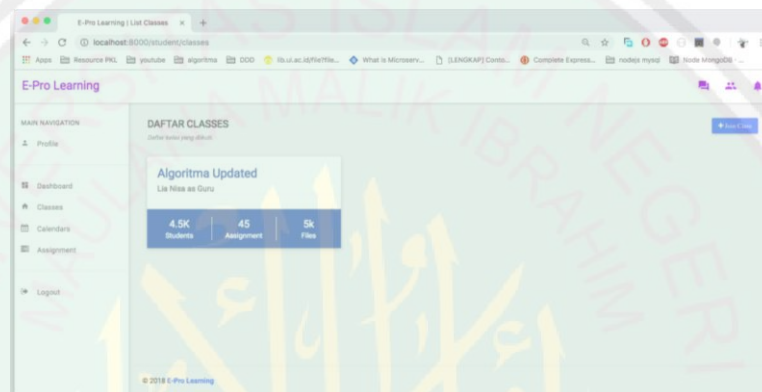


Gambar 4.19 Halaman Kelas Pengajar

Dari gambar 4.19 diatas terlihat bahwa akun guru tersebut tergabung dalam 2 kelas.

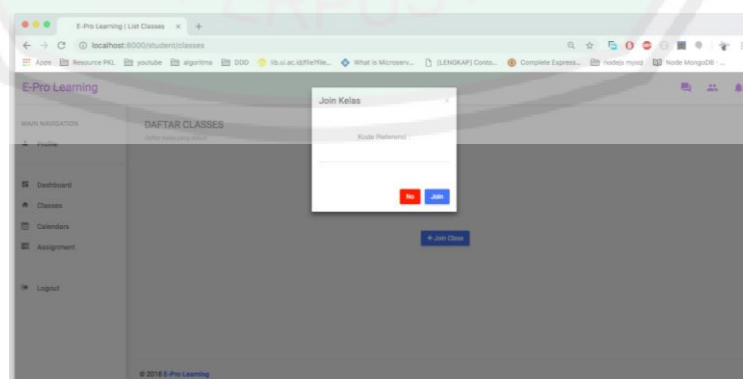
3. Peserta Didik

Pada halaman ini akan menampilkan kelas dimana siswa tergabung dalam kelas tersebut. Apabila kelas di klik akan menampilkan detail kelas seperti detail kelas pada role administrator.



Gambar 4.20 Halaman Kelas Siswa

Dari gambar 4.20 diatas terlihat bahwa akun siswa tersebut tergabung dalam 1 kelas. Apabila siswa belum tergabung dalam kelas maka siswa perlu melakukan *join class* dengan memasukkan *code kelas* ke dalam *pop up* menu *join kelas*.



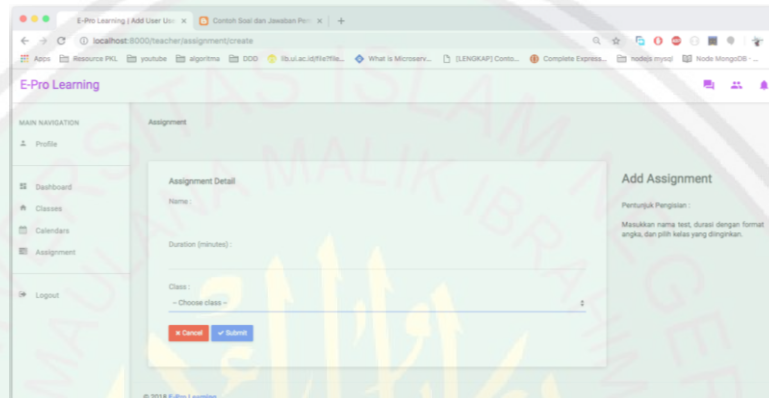
Gambar 4.21 Halaman Join Kelas Siswa

e. Halaman *Test*

1. Guru

- Tambah *Assignment*

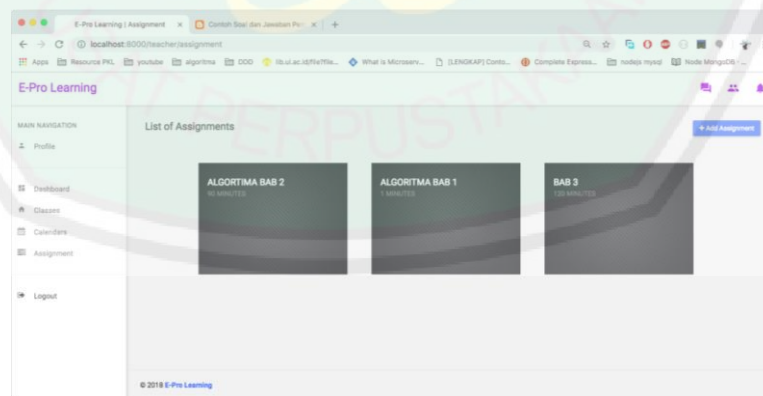
Merupakan halaman untuk membuat *test*. Halaman tambah *assignment* ini hanya dapat diakses oleh guru.



Gambar 4.22 Halaman Tambah *Assignment*

- List *Assignment*

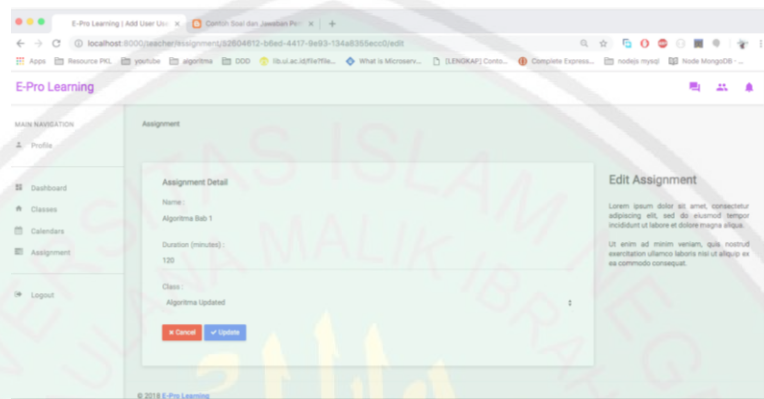
Merupakan *list* dari *assignment* yang telah dibuat oleh guru.



Gambar 4.23 Halaman List *Assignment*

- Edit Assignment

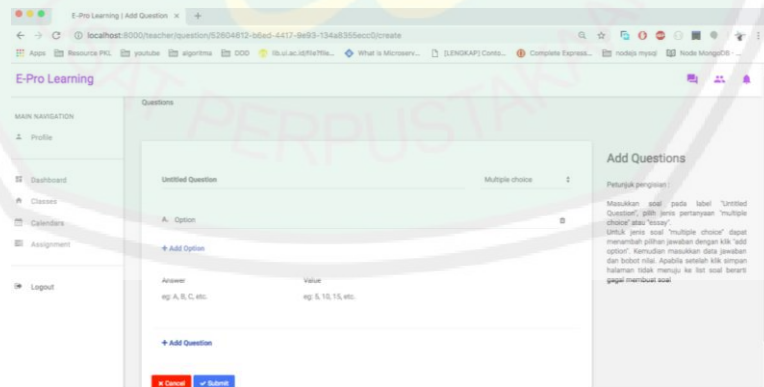
Halaman ini dapat diakses oleh guru yang ingin melakukan perubahan pada *assignment* yang telah dibuat. *Update assignment* dapat diakses dengan meng-klik ikon *pencil* pada item *assignment*.



Gambar 4.24 Halaman Edit Assignment

- Tambah Soal

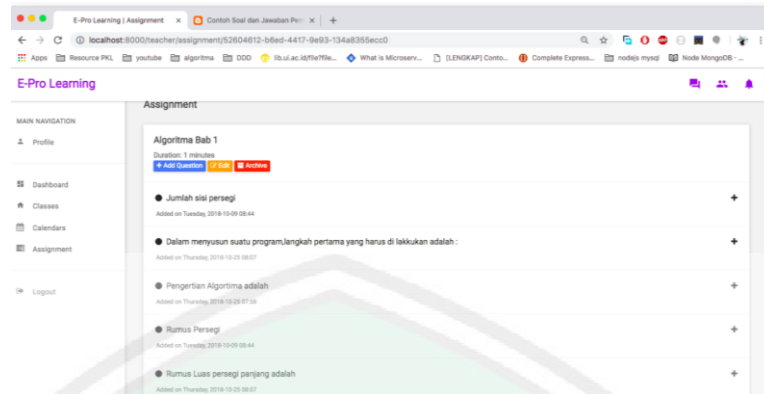
Setelah selesai membuat *assignment*, guru dapat membuat soal. Guru dapat membuat soal pada *assignment* yang telah dipilih. Soal dapat dibuat dalam jumlah banyak sekaligus dalam sekali *submit*.



Gambar 4.25 Halaman Tambah Soal

- List Soal

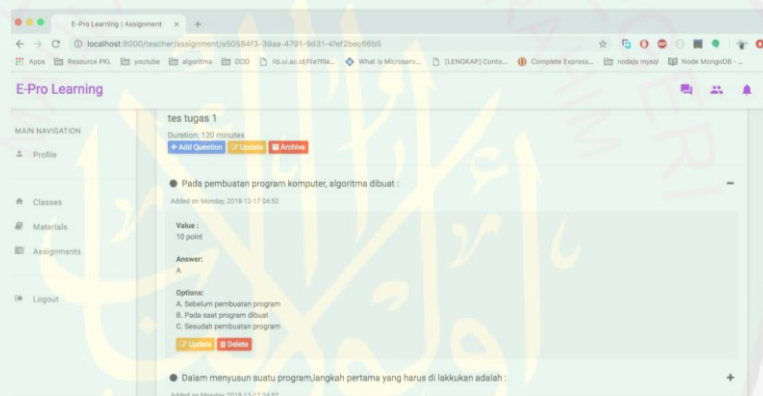
Selesai membuat soal maka sistem akan *redirect* ke *list* soal.



Gambar 4.26 Halaman List Soal

- Detail Soal

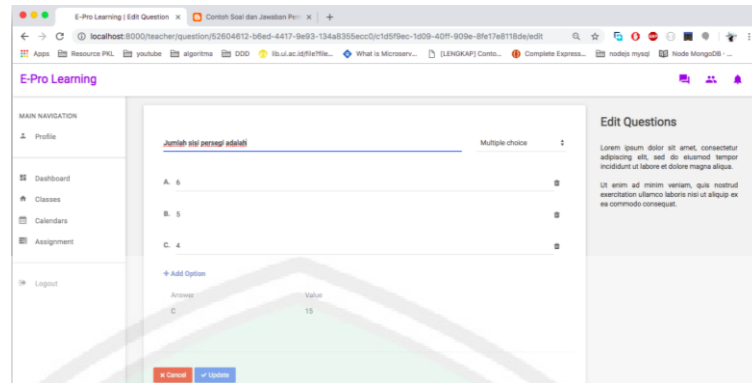
Pada halaman ini guru dapat melihat detail dari soal yang telah dibuat.



Gambar 4.27 Halaman Detail Soal

- Edit Soal

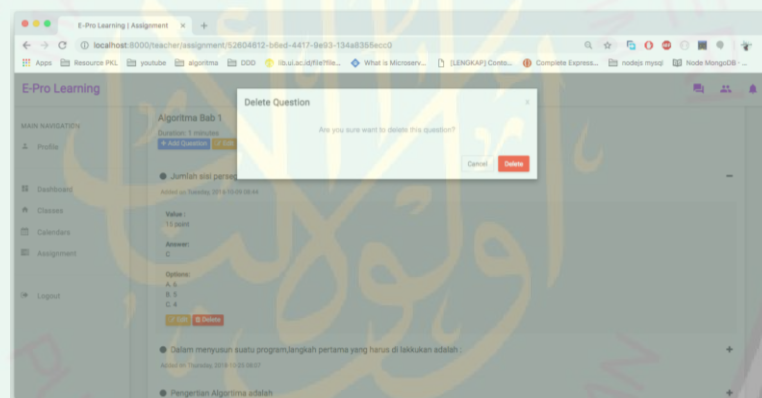
Halaman ini dapat diakses oleh guru yang ingin melakukan perubahan pada soal yang telah dibuat. Edit soal dapat diakses dengan meng-klik *button update* pada detail soal.



Gambar 4.28 Halaman Edit Soal

- Hapus Soal

Saat melihat detail soal gambar 4.27 terdapat *button* hapus yang berfungsi untuk menghapus soal.

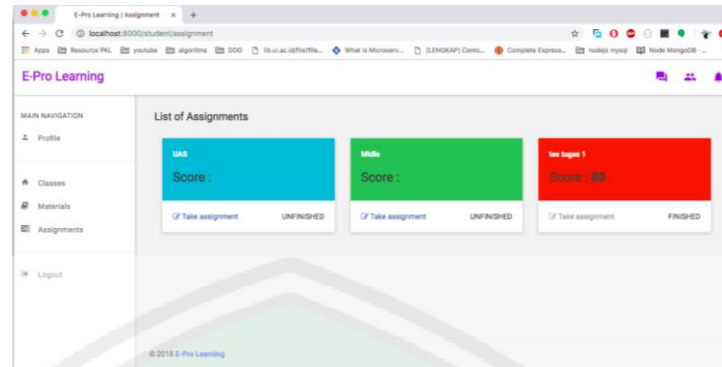


Gambar 4.29 Halaman Hapus Soal

2. Siswa

- *Assignment*

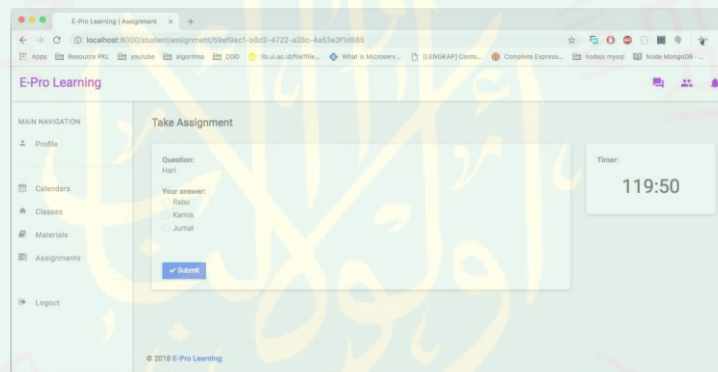
Saat siswa mengakses fitur *assignment* maka yang muncul adalah list dari *assignment*.



Gambar 4.30 Halaman Assignment

Apabila siswa telah mengerjakan *test* maka siswa dapat melihat *score* yang didapat dan tidak dapat mengakses *test* tersebut.

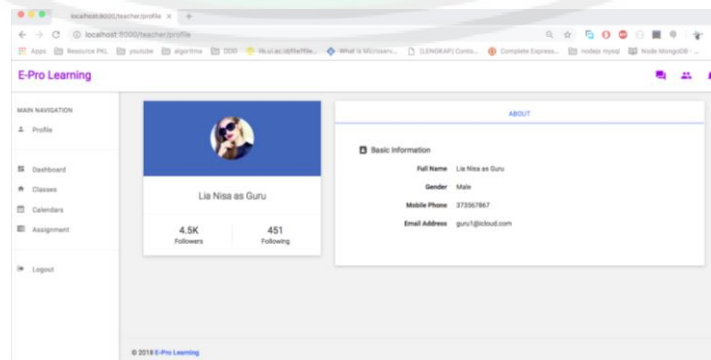
- Test



Gambar 4.31 Halaman Test Siswa

f. Halaman Profil

Halaman ini menampilkan data diri *user*.



Gambar 4.32 Halaman Lihat Profil

4.4 Pengujian Sistem

4.4.1 Pengujian dengan *Multiple Server*

Untuk memastikan sistem yang dihasilkan sesuai dengan kebutuhan maka perlu dilakukan *testing* atau pengujian sistem. Pengujian pada penelitian ini dilakukan dengan melakukan *cloning microservices* pada 3 VPS atau server yang berbeda. *Virtual Private Server* yang digunakan peneliti dari *Google Cloud Platform*. Instance pertama dengan IP address 35.240.196.35 bernama API-1, instance kedua dengan IP address 35.240.135.69 bernama API-2, dan instance ketiga dengan IP address 35.187.232.113 bernama API-3. Selain *microservices* penulis juga menyediakan server untuk *API-Gateway* dan *Front End*. Berikut ini adalah list *instance* yang akan digunakan :

Nama ^	Zona	Rekomendasi	Sedang digunakan oleh	IP internal	IP eksternal
✓ api-1	asia-southeast1-b	🔦 Tambah perform.		10.148.0.5 (nic0)	35.240.196.35 ↗
✓ api-2	asia-southeast1-b	🔦 Tambah perform.		10.148.0.6 (nic0)	35.240.135.69 ↗
✓ api-3	asia-southeast1-b	🔦 Tambah perform.		10.148.0.7 (nic0)	35.187.232.113 ↗
✓ api-gateway	asia-southeast1-b			10.148.0.4 (nic0)	35.240.240.163 ↗
✓ epro-frontend	asia-south1-c			10.160.0.2 (nic0)	35.200.224.237 ↗

Gambar 4.33 List Server

Virtual server melakukan distribusi beban dengan menggunakan *load balancer* yang menerapkan algoritma *scheduling*. Algoritma yang digunakan adalah *round robin*, yaitu algoritma yang akan membagi rata beban kepada semua server yang menjadi anggota *cluster*. Berikut adalah contoh konfigurasi server yang pada *class-service*

```

import _ from "lodash";
import rp from "request-promise";
import servers from "../../config/servers";

let current = 0;

module.exports = (req, res) => {
  const available_servers = _.filter(servers[process.env.APP_ENV].class, server => {
    return server.status === true;
  });
  console.log("class on Server " + available_servers[current].url);

  const endpoint = req.url.replace(/\/admin\/\student\/\teacher\/gi, "");
  const options = {
    method: req.method,
    url: available_servers[current].url + endpoint,
    headers: {
      "user-id": req.credential.data.id,
      "role": req.credential.data.role,
      "id": req.credential.data.reference_id,
      "token": req.headers.token
    },
    body: req.body,
    json: true
  };

  if (available_servers.length >= 1) {
    current++;
    if (current === available_servers.length) current = 0;

    rp(options).then(result => {
      res.json(result);
    }).catch(error => {
      res.status(error.statusCode);
      res.json(error.error);
    });
  }
};

```

Gambar 4.34 Konfigurasi Server Class

Gambar 4.35 dibawah menunjukkan ketiga server dalam kondisi *online* dan sistem dijalankan pada *browser*. Karena semua server dalam kondisi *online* maka sistem dapat diakses. Dapat dilihat pada terminal di sebelah kiri bahwa urutan penggunaan server adalah dimulai dari API-1, API-2 API-3 sesuai dengan algoritma *round robin*.

The screenshot displays a terminal window with three distinct sections, each representing a different server (API-1, API-2, and API-3). Each section contains a table of application status and a log of recent requests.

API-1 Status Table:

App name	id	version	mode	pid	status	restart	uptime	cpu	mem	user	watching
class-service	0	1.0.0	prod	26140	online	0	2s	61.8%	104.5 MB	askhs	disabled
material-service	1	1.0.0	prod	25067	online	0	28s	8%	32.3 MB	askhs	disabled
test-service	2	1.0.0	prod	25700	online	0	27s	8%	62.7 MB	askhs	disabled

API-2 Status Table:

App name	id	version	mode	pid	status	restart	uptime	cpu	mem	user	watching
class-service	0	1.0.0	prod	26220	online	0	2s	62.8%	104.4 MB	askhs	disabled
material-service	1	1.0.0	prod	7441	online	0	27s	0.4%	92.4 MB	askhs	disabled
test-service	2	1.0.0	prod	7463	online	0	27s	0.4%	63.5 MB	askhs	disabled

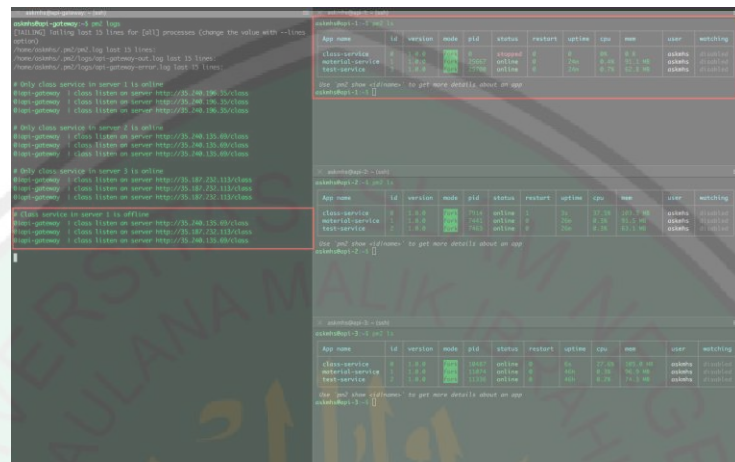
API-3 Status Table:

App name	id	version	mode	pid	status	restart	uptime	cpu	mem	user	watching
class-service	0	1.0.0	prod	10507	online	0	4s	40.4%	106.4 MB	askhs	disabled
material-service	1	1.0.0	prod	11074	online	0	40s	0.5%	97.5 MB	askhs	disabled
test-service	2	1.0.0	prod	11336	online	0	40s	0.2%	74.7 MB	askhs	disabled

The logs on the left of each section show the sequence of requests being processed by each server, indicating a round-robin distribution of traffic.

Gambar 4.35 Ketiga Server dengan Kondisi Online

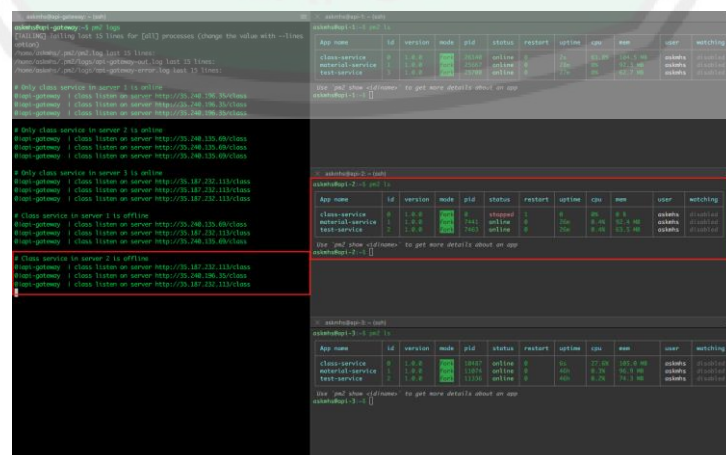
Kemudian salah satu *service* pada salah satu *server* dimatikan secara acak, disini sebagai contoh *class-service* pada *server* pertama dengan IP 35.240.196.35 dinonaktifkan seperti gambar 4.36 dan sistem dijalankan kembali pada *browser*.



Gambar 4.36 class-service pada Server Pertama Kondisi Offline

Karena *class-service* merupakan *service* yang dinonaktifkan maka perlu dicoba mengakses fitur *class*. Pada percobaan ini sistem dapat berjalan dan dapat mengakses fitur *class* karena masih ada *server* yang dalam kondisi *online*, yaitu *server* 2 dan 3. Pada gambar 4.36 terlihat *class-service* berjalan pada API 2 dan 3.

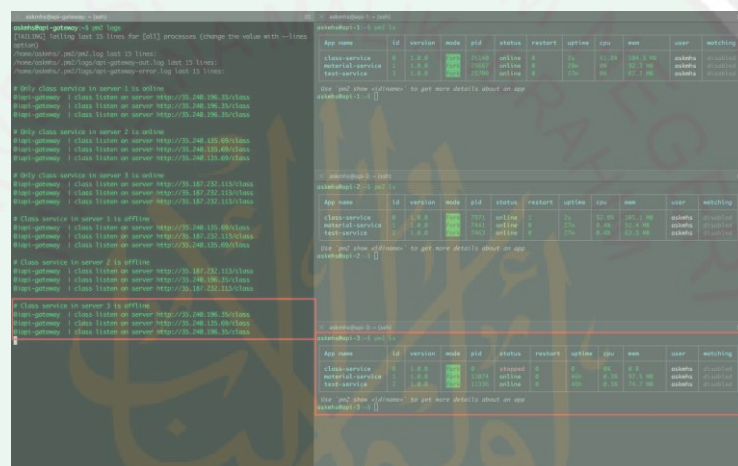
Kemudian *server* kedua *class-service* dinonaktifkan, seperti gambar 4.37 berikut.



Gambar 4.37 class-service pada Server Kedua Kondisi Offline

Gambar 4.37 menunjukkan bahwa *class-service* dapat diakses pada server 3 dan 1. Pada percobaan ini *class-service* berjalan di server 3 terlebih dahulu karena pada percobaan sebelumnya *class-service* berjalan pada server 2, dan karena server 2 dalam posisi *offline* sehingga pada percobaan ini *class-service* berjalan pada server selanjutnya, yaitu server 3 baru ke server 1.

Selanjutnya adalah server 3 dinonaktifkan, dan hasilnya adalah seperti gambar berikut.



Gambar 4.38 *class-service* pada Server Ketiga Kondisi Offline

Sama seperti percobaan sebelumnya, karena server 3 posisi *offline* sehingga pada gambar 3.38 *class-service* dapat berjalan pada server 1 dan 2. Pada percobaan ini *class-service* berjalan di server 1 terlebih dahulu karena pada percobaan sebelumnya *class-service* berjalan pada server 3, sehingga untuk selanjutnya *class-service* berjalan pada server selanjutnya, yaitu server 1.

Berikutnya adalah menonaktifkan *service* yang sama yaitu *class-service* pada kedua server yaitu server 1 dan 2, seperti gambar 4.39 berikut.

```

ashraf@api-gateway:~$ ps -ef
[...]
```

```

ashraf@api-1:~$ ps -ef
[...]
```

```

ashraf@api-2:~$ ps -ef
[...]
```

Gambar 4.39 *class-service* pada *Server 1* dan *Server 2* Kondisi *Offline*

Selanjutnya sistem dijalankan kembali pada *browser* dan tetap mengakses fitur *class*. Pada percobaan ini sistem tetap dapat berjalan dan dapat mengakses fitur *class* karena masih ada *server 3* yang dalam kondisi *online*.

Percobaan selanjutnya adalah menonaktifkan *server 1* dan *3*, sehingga *class-service* hanya dapat diakses pada *server ke 2* saja.

```

ashraf@api-gateway:~$ ps -ef
[...]
```

```

ashraf@api-1:~$ ps -ef
[...]
```

```

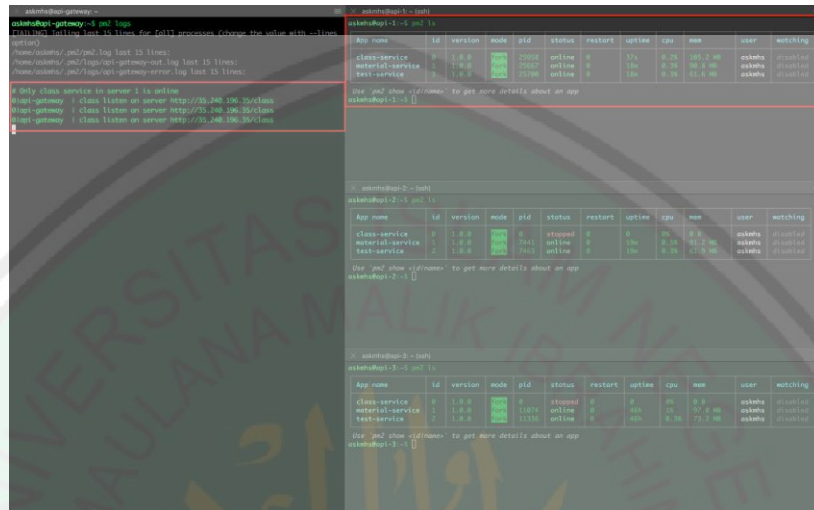
ashraf@api-2:~$ ps -ef
[...]
```

```

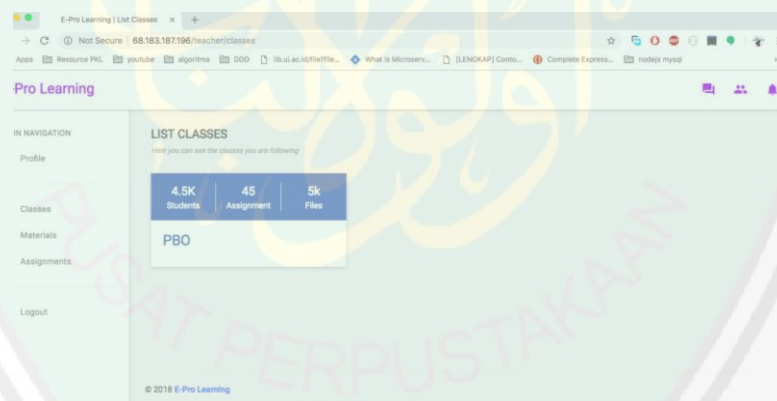
ashraf@api-3:~$ ps -ef
[...]
```

Gambar 4.40 *class-service* pada *Server 1* dan *Server 3* Kondisi *Offline*

Untuk percobaan terakhir yaitu menonaktifkan server 2 dan 3, sehingga *class-service* hanya dapat diakses pada server ke 1 saja. Dapat dilihat pada gambar 4.41 bahwa *class-service* hanya berjalan pada server 1.



Gambar 4.41 *class-service* pada Server 2 dan 3 Kondisi Offline



Gambar 4.42 Layanan *class-service* dapat Diakses

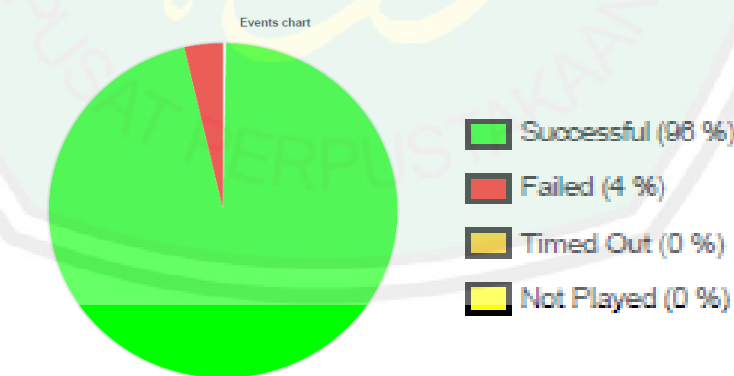
Gambar 4.42 diatas adalah tampilan ketika *class-service* dapat diakses. Dari percobaan diatas maka dapat disimpulkan bahwa sistem dapat diakses apabila terdapat minimal 1 server yang berada dalam kondisi *online*. Dengan adanya percobaan ini maka dapat dikatakan bahwa sistem yang dibangun dengan *microservices* dan ditempatkan di banyak server menunjukkan aspek *scalable* dan

kualitas *resilient*. Aspek *scalable* merupakan kemampuan sistem untuk menangani penambahan beban yang diberikan. Sehingga apabila ingin memberikan beban atau *service* baru sistem masih dapat berjalan pada *server* yang lain. Sedangkan *resilient* merupakan kemampuan beradaptasi terhadap perubahan. Aspek *scalable* dan *resilient* dapat dikatakan saling berkaitan. Apabila sistem mampu menambah beban atau *server* tanpa masalah dapat dikatakan mempunyai kemampuan beradaptasi terhadap dengan baik.

4.4.2 Pengujian Fungsionalitas

Uji keberfungsian (*blackbox testing*) dilakukan dengan perangkat lunak *AppPerfect Web Test* versi 15.0.0 yang masing-masing dilakukan sebanyak 10 kali perulangan dengan tiga kondisi, yaitu : (1) *Successful* atau sukses; (2) *Failed* atau gagal; (3) *Timed Out* atau waktu koneksi habis; dan (4) *Not Played* atau tidak dijalankan. Berikut hasil masing-masing percobaan :

1. Administrator

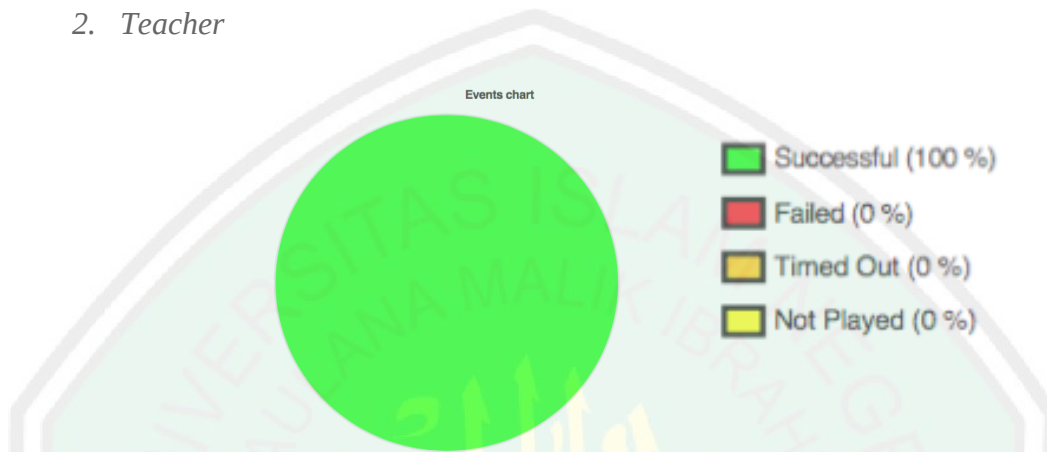


Gambar 4.43 Grafik Uji Fungsionalitas Role Administrator

Berdasarkan data gambar 4.43 diatas, fungsionalitas dengan nilai berhasil (*successfull*) didapatkan nilai 96%, kegagalan sebesar 4%. Hal ini terjadi karena

koneksi maupun mengenali variable email. Dengan nilai 96% dapat dikatakan menu, tombol, *form* (isian), *link* (tautan) pada *role administrator* berfungsi semua dengan baik. Sehingga secara teknis, dapat dikatakan laman ini layak digunakan.

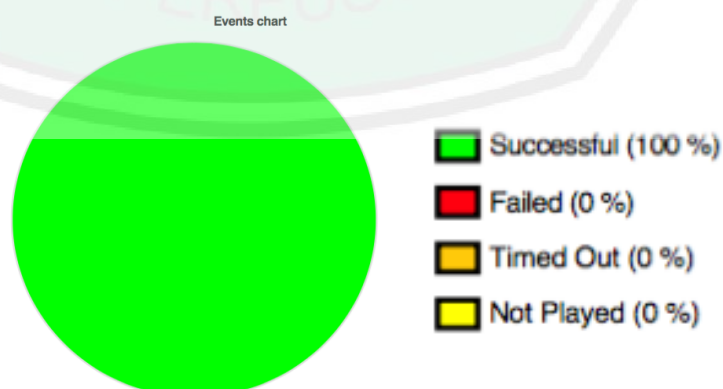
2. Teacher



Gambar 4.44 Grafik Uji Fungsionalitas *Role Teacher*

Berdasarkan data gambar 4.44 diatas, fungsionalitas dengan nilai berhasil (*successfull*) didapatkan nilai 100% dengan kata lain menu, tombol, *form* (isian), *link* (tautan) pada *role teacher* berfungsi semua dengan sangat baik. Sehingga secara teknis, dapat dikatakan laman ini layak digunakan.

3. Student



Gambar 4.45 Grafik Uji Fungsionalitas *Role Student*

Berdasarkan data gambar 4.45 diatas, fungsionalitas dengan nilai berhasil (*successful*) didapatkan nilai 100% dengan kata lain menu, tombol, *form* (isian), *link* (tautan) pada *role student* berfungsi semua dengan sangat baik. Sehingga secara teknis, dapat dikatakan laman ini layak digunakan.

4.4.3 Unit Test dan Code Coverage

Unit test menguji potongan kecil dan terisolasi dari fungsi. *Unit test* berfokus pada unit kode tunggal yang biasanya berupa *function* dalam objek atau modul. *Unit tests berfungsi* untuk menguji bagian terkecil perangkat lunak apakah sistem berjalan sesuai kebutuhan apa belum. Dengan *unit tests developer* dapat mengetahui letak kesalahan dalam *code* lebih detail sehingga akan mempermudah *debugging*. Pengujian ini dilakukan dengan menulis kode untuk melakukan pengujian dengan menambahkan *module* dari *Node.js* yaitu, *mocha*. Setelah kode pengujian selesai dibuat dilanjutkan dengan menguji aplikasi dengan kode program yang telah dibuat tadi. Apabila hasil dari pengujian terdapat error maka kode aplikasi harus disesuaikan dengan kondisi yang diinginkan. Apabila hasil pengujian tidak terdapat error maka dilanjutkan dengan pengujian yang lain.

Salah satu indikasi baiknya kualitas *testing* adalah *code coverage*. *Code coverage* menampilkan seberapa banyak baris *code* yang sudah diuji. Sehingga *code coverage* merupakan ukuran berapa persen *code* sudah teruji kebenarannya. Pada penelitian ini *code coverage* dibuat dengan *package Istanbul*. Berikut hasil *unit test* dan *code coverage*. Berikut contoh *unit test* dan *code coverage auth-service*. Pada gambar 4.46 terlihat bahwa semua *unit* tercentang semua yang berarti alur sistem *auth-service* sudah benar. Sedangkan gambar 4.47 terlihat

bahwa semua *statement*, *branch*, *function* dan *line* bernilai 100% yang berarti semua kode diuji kebenarannya.

```

USER LOGIN SERVICE TEST
# ADMIN LOGIN
✓ Should return result as an object
✓ Should return a valid response body
Negative tests
✓ Should throw UserNotFoundException if the username or password is not valid
✓ Should have 'Invalid username or password!' error message
# STUDENT LOGIN
✓ Should return result as an object
✓ Should return a valid response body
Negative tests
✓ Should throw UserNotFoundException if the username or password is not valid
✓ Should have 'Invalid username or password!' error message
# TEACHER LOGIN
✓ Should return result as an object
✓ Should return a valid response body
Negative tests
✓ Should throw UserNotFoundException if the username or password is not valid
✓ Should have 'Invalid username or password!' error message

USER REGISTRATION TEST
# REGISTER ADMIN
✓ Should return result as an object
✓ Should return a valid response body
Negative tests
✓ Should throw error if the data is not valid
# REGISTER STUDENT
✓ Should return result as an object
✓ Should return a valid response body
Negative tests
✓ Should throw error if the data is not valid
# REGISTER TEACHER
✓ Should return result as an object
✓ Should return a valid response body
Negative tests
✓ Should throw error if the data is not valid

21 passing (200ms)

```

Gambar 4.46 unit test

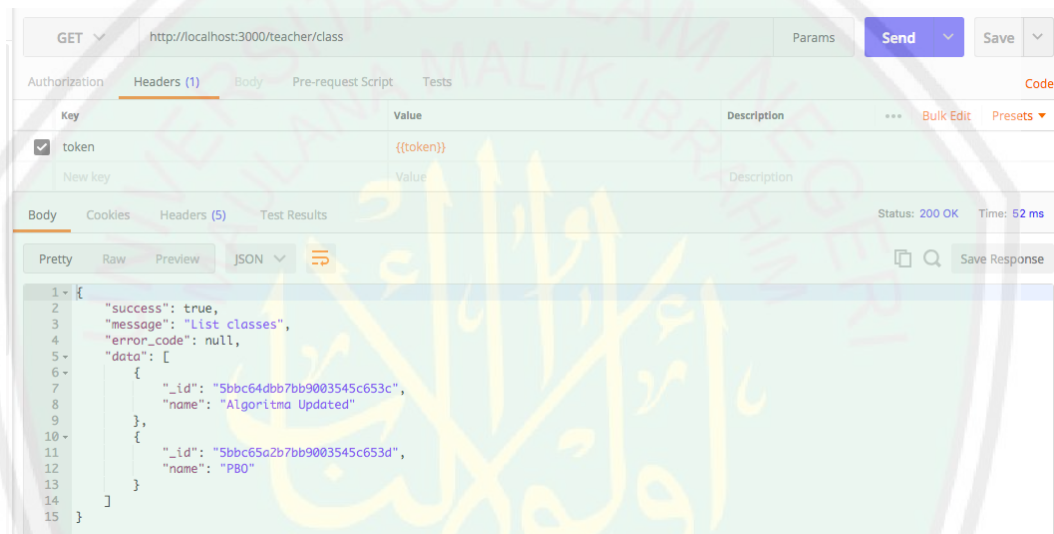
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
Application/Exceptions	100	100	100	100	
UserNotFoundException.js	100	100	100	100	
Domain/Student/Entities	100	100	100	100	
StudentEntity.js	100	100	100	100	
Domain/Student/Services	100	100	100	100	
CreateStudentService.js	100	100	100	100	
Domain/Teacher/Entities	100	100	100	100	
TeacherEntity.js	100	100	100	100	
Domain/Teacher/Services	100	100	100	100	
CreateTeacherService.js	100	100	100	100	
Domain/User/Entities	100	100	100	100	
UserEntity.js	100	100	100	100	
UserLoginEntity.js	100	100	100	100	
Domain/User/Services	100	100	100	100	
UserLoginService.js	100	100	100	100	
UserRegistrationService.js	100	100	100	100	
Infrastructure	100	100	100	100	
TokenGenerator.js	100	100	100	100	
Infrastructure/Repositories	100	100	100	100	
UserRepository.js	100	100	100	100	

Gambar 4.47 code coverage

4.4.4 Integration Test

Kemudian penulis melakukan pengujian *integration testing*, yaitu pengujian antar *service*. Pengujian ini penting untuk mengetahui *logging* melalui layanan *API* pada saat pengujian serta menunjukkan *response* yang didapat saat melakukan *request*. Pada *integration test* ini penulis menggunakan *postman*.

- Integrasi *auth-service* ke *class-service*



Gambar 4.48 Integrasi *auth-service* ke *class-service*

Gambar 4.48 menunjukkan bahwa *auth-service* (*teacher*) melakukan *request* ke *class-service* untuk mendapatkan data *list class*. Berikut rincian REST API :

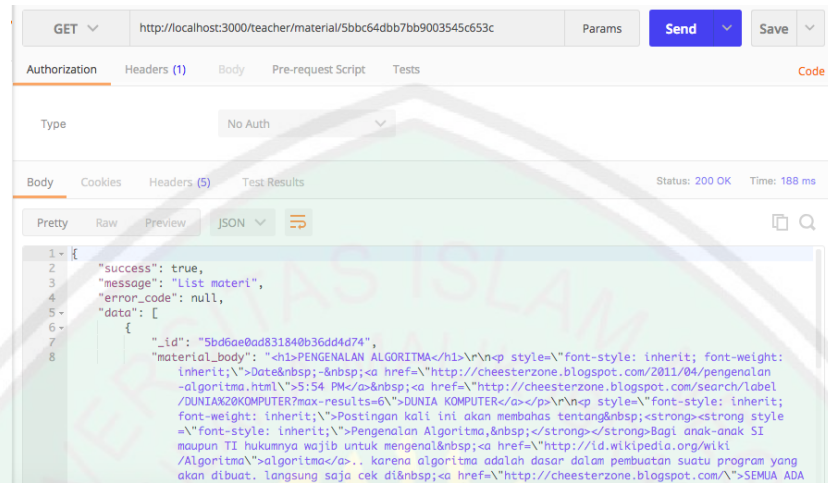
HTTP request :

- HTTP method : GET
- URI : `http://localhost:3000/teacher/class`
- Header : token teacher

HTTP response :

- Response Code : 200

- *Response Body* : data *list class* berupa id dan nama
- Integrasi *auth-service* ke *material-service*



Gambar 4.49 Integrasi *auth-service* ke *material-service*

Gambar 4.49 menunjukkan bahwa *auth-service (teacher)* melakukan *request* ke *material-service* untuk mendapatkan detail data materi. Berikut rincian REST API :

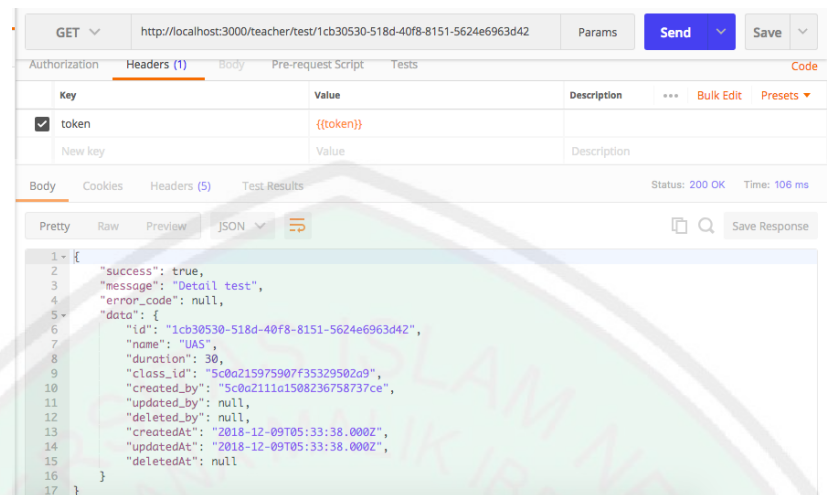
HTTP request :

- *HTTP method* : GET
- *URI* : *http://localhost:3000/teacher/material/id_material*
- *Header* : *token teacher*

HTTP response :

- *Response Code* : 200
- *Response Body* : data *list materi*

- Integrasi *auth-service* ke *test-service*



Gambar 4.50 Integrasi *auth-service* ke *test-service*

Gambar 4.50 menunjukkan bahwa *auth-service* (*teacher*) melakukan *request* ke *test-service* untuk mendapatkan detail data *test* dengan *id test* tertentu.

Berikut rincian REST API :

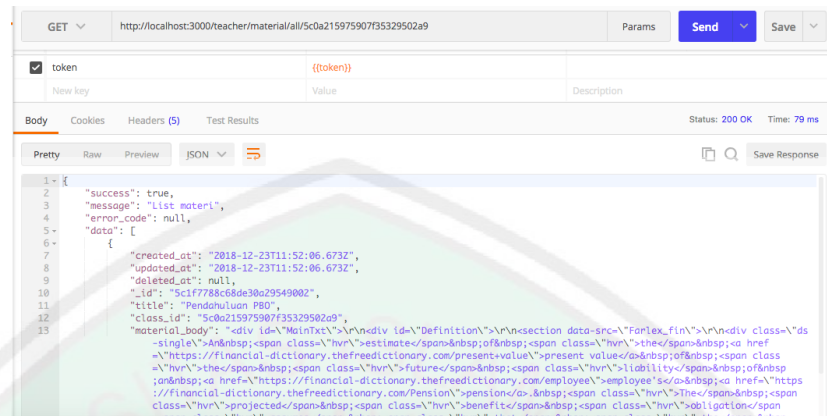
HTTP request :

- HTTP method : GET
- URI : http://localhost:3000/teacher/test/id_test
- Header : token teacher

HTTP response :

- Response Code : 200
- Response Body : detail data test

- Integrasi *material-service* ke *class-service*



Gambar 4.51 Integrasi *material-service* ke *class-service*

Gambar 4.51 menunjukkan bahwa *material-service* melakukan *request* ke *class-service* untuk mendapatkan data *list* materi dengan *id class* tertentu. Berikut rincian REST API :

HTTP *request* :

- HTTP method : GET
- URI : http://localhost:3000/teacher/material/all/id_class
- Header : token teacher

HTTP *response* :

- *Response Code* : 200
- *Response Body* : data list materi yang mempunyai id class tersebut

BAB V

PENUTUP

5.1. Kesimpulan

Setelah peneliti melakukan penelitian yang berjudul “ Rancang Bangun Sistem *e-Learning* Berbasis *Microservices* dan *Domain Driven Design* (Studi Kasus Prohistek UIN Maulana Malik Ibrahim Malang) ”, dapat ditarik kesimpulan bahwa :

1. Dalam membangun sistem *e-learning* yang pertama kali dilakukan adalah menentukan proses bisnis sehingga dapat menentukan *domain*, *subdomain* dan *bounded context*. Kemudian menentukan bahasa pemrograman dan *DBMS* sesuai dengan kebutuhan setiap *services*.
2. Pengaruh implementasi terhadap sistem setelah dilakukan penelitian adalah :
 - a. Implementasi *multiple servers* dari masing-masing *services* adalah *scalability*, yang terdiri dari :
 1. Sistem tetap berjalan normal ketika salah satu *server* mati / bermasalah
 2. Sistem tetap berjalan normal ketika ada salah satu *service* pada salah satu *server* yang mati
 3. Pembagian beban kerja merata pada masing-masing *servers* dan atau *services*
 - b. Implementasi *microservices* dan *domain driven design*
 1. *Scalability*
 - a. Pengembangan sistem dapat dilakukan dengan mudah karena tidak akan mengganggu kinerja *services* yang lain.

- b. Tidak bergantung terhadap bahasa pemrograman

2. *Effectivity Deployment*

Services dapat dideploy secara otonom tanpa harus menunggu keseluruhan *services* selesai dibangun.

5.2. Saran

Peneliti menyadari bahwa dalam penelitian ini banyak yang perlu dikembangkan guna tercapai kinerja yang lebih baik lagi. Maka saran untuk penelitian ini adalah :

1. Untuk mendapatkan skalabilitas yang lebih tinggi perlu diterapkan CQRS (*command query responsibility segregation*) pada penelitian berikutnya.
2. Perlu adanya kesempurnaan pada pembuatan soal yaitu berupa pertanyaan *essay*.

DAFTAR PUSTAKA

- As-Sa'di, Abdurrahman. (2002). Taisirul Kariimir Rahman fi Tafsiiri Kalamil Mannan. Daarus Salam.
- Budiarto, Eko (2009). E-Learning. Tangerang: STMIK Raharja
- Evans, Eric. (2004). Domain Driven Design: Tackling Complexity in the Heart of Software. Canada: Addison Wesley
- Fediri. (2016). Mengenal RESTful Web Service. Dipetik Oktober 8, 2017 dari <https://www.codepolitan.com/mengenal-restful-web-services>
- Fowler, Martin. (2011). Polyglot Persistence. Dipetik 25 September 25, 2017 dari <http://martinfowler.com/bliki/PolyglotPersistence.html>
- Francesco, Paolo, D dkk. (2019). Architecting with Microservices: A Systematic Mapping Study. ScienceDirect
- Gonzalez, David. (2016). Developing Microservices with Node.js. Mumbai: Packt Publishing, 2
- Hartanto, A. Aditya & Purwo, Onno W. (2002). Buku Pintar Internet: Teknologi e-learning Berbasis PHP dan MySQL. Jakarta: Elex Media Komputindo
- Haryandi, Sepry. (2016). Mengenal RESTful API. Dipetik Oktober 8, 2017 dari <https://developers.kudo.co.id/2016/09/15/mengenal-restful-api/>
- Livora, Bc. Tomas. (2016). Master's Thesis: *"Fault Tolerance in Microservices"*. Brno: Masaryk University, 3
- Marton, Peter. (2017). Building an API Gateway using Node.js. Dipetik 12 November 12, 2017 dari <https://blog.risingstack.com/building-an-api-gateway-using-nodejs/>
- McFarlan, F. Warren. (thn). Portfolio Approach to Information Systems. Dipetik Oktober 12, 2017 dari <https://hbr.org/1981/09/portfolio-approach-to-information-systems>
- Munawar, Ghifari. Hodijah, Ade. (2018). Analisis Model Arsitektur Microservcies pada Sistem Informasi DPLK. Jurnal dan Penelitian Teknik

Informatika

- Nadareishvili, Irakli, dkk. (2016). *Microservices Architecture: Aligning Principles, Practices, and Culture*. Sebastopol: O'Reilly Media
- Newman, Sam. (2015). *Building Microservices: Designing Fine-Grained Systems*. Sebastopol: O'Reilly Media
- Pahl, C. dan Jamshidi, P. (2016). *Microservices: A Systematic Mapping Study. International Conference on Cloud Computing and Services Science* (vol 1)137.146
- Polladino, Marco. (2017). *Microservices & API Gateways, Part 1: Why an API Gateway* ?.Dipetik November 4, 2017 dari <https://www.nginx.com/blog/microservices-api-gateways-part-1-why-an-api-gateway/>
- Powell-Morse, Andrew. (2017). *Domain-Driven Design - What is it and how do you use it* ?.Dipetik September 25, 2017 dari <https://airbrake.io/blog/software-design/domain-driven-design>
- Purnama, Heri Purnama &B, Indra Yatini B. (2016). *Aplikasi Pengelolaan Skripsi di STMIK Akakom Yogyakarta Menggunakan Arsitektur Microservices dengan Node.js*. Seminar Riset Teknologi Informasi (SRITI).
- Rosenberg, Marc J. (2001). *e-Learning : Strategies for Delivering Knowledge in the Digital*. New York: McGraw Hill
- Sokhan, Berke. (2015). *Domain Driven Design for Services Architecture*.Dipetik Oktober 14, 2017 dari <https://www.thoughtworks.com/insights/blog/domain-driven-design-services-architecture>
- Suryostrisongko, Hatma. (2017). *Arsitektur Microservices untuk Resiliensi Sistem Informasi*. Surabaya: Institut Teknologi Sepuluh Nopember
- Triwahyudi, Boyke Dian. (2016). *Microservices, Konsep dan Implementasi 1*. Dipetik September 25, 2017 dari <https://indosystem.com/blog/microservices->

konsep-dan-implementasi/

Vernon, Vaughn. (2013). Implementing Domain-Driven Design. US: Addison Wesley, 46, 48, 50

Weir, Luis. Wunderlich, Robert. (2016). Microservices and SOA. San Francisco: Oracle Open World, 7

Wikipedia. (2016) Dipetik 10 Oktober 10, 2017 dari
https://id.wikipedia.org/wiki/Pembelajaran_elektronik#cite_note-1

Wikipedia. (2017). Dipetik Oktober 12, 2017 dari
https://en.wikipedia.org/wiki/Software_development.

Williams, Craig. (2015). Is REST Best in Microservices Architecture?. Dipetik Oktober 13, 2017 dari <https://capgemini.github.io/architecture/is-rest-best-microservices/>