PENERAPAN DJANGO REST FRAMEWORK DAN TEKNOLOGI OTENTIKASI OAUTH 2.0 UNTUK SISTEM INFORMASI AKADEMIK UNIVERSITAS LAMPUNG VERSI ANDROID

(Skripsi)

Oleh

SAIFUL ANWAR

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS LAMPUNG
BANDAR LAMPUNG

2018

ABSTRACT

DJANGO REST FRAMEWORK APPLICATION AND OAUTH 2.0 OTENTICATION TECHNOLOGY FOR ACADEMIC INFORMATION SYSTEMS UNIVERSITY OF LAMPUNG ON ANDROID VERSION

By

SAIFUL ANWAR

The University of Lampung is the oldest state university in Lampung province having different system units of different platforms and programming languages. These days there are more and more smartphone users and Android is the most dominant operating system. Android based smartphone is very easy to use, so many people choose it. The combination of Django REST Framework with OAuth 2.0 technology has the opportunity to be implemented so that it promises ease and improvement in support of integration of various system and application platforms..

This research has successfully implemented Django REST Framework architecture and OAuth 2.0 technology implemented in Android application client. The conclusion of this research is the application of academic information system of University of Lampung Android version successfully built and useful for every user who have used this application, evidenced by questionnaire of application test which got very good value.

Keywords : Web Service, Django REST Framework, OAuth 2.0, System
Academic Information, Android, Application Programming Interface

ABSTRAK

PENERAPAN DJANGO REST FRAMEWORK DAN TEKNOLOGI OTENTIKASI OAUTH 2.0 UNTUK SISTEM INFORMASI AKADEMIK UNIVERSITAS LAMPUNG VERSI ANDROID

Oleh

SAIFUL ANWAR

Universitas Lampung merupakan perguruan tinggi negeri tertua di propinsi Lampung memiliki unit-unit sistem yang berbeda platform dan bahasa pemrograman yang berbeda. Kondisi ini memberikan pengaruh besar pada kinerja masing-masing unit. Dari banyaknya pengguna smartphone, sistem operasi yang mempunyai banyak pengguna adalah Android. Smartphone berbasis Android sangat mudah digunakan, sehingga banyak orang memilih untuk menggunakan smartphone berbasis Android. Dengan adanya kombinasi antara Django REST Framework dengan teknologi OAuth 2.0 sangat mungkin diimplementasikan sehingga menjanjikan kemudahan dan perbaikan dalam mendukung integrasi berbagai platform sistem dan aplikasi.

Penelitian ini menghasilkan penerapan arsitektur Django REST Framework dan teknologi OAuth 2.0 yang di implementasikan dalam *client* aplikasi Android. Kesimpulan yang didapat dari penelitian ini adalah aplikasi sistem informasi

akademik Universitas Lampung versi Android berhasil dibangun dan berguna bagi setiap pengguna yang telah menggunakan aplikasi ini, dibuktikan dengan kuisioner pengujian aplikasi yang mendapat nilai sangat baik.

Kata Kunci : *Web Service*, Django REST Framework, OAuth 2.0, Sistem Infromasi Akademik, Android, *Application Programming Interface*

PENERAPAN DJANGO REST FRAMEWORK DAN TEKNOLOGI OTENTIKASI OAUTH 2.0 UNTUK SISTEM INFORMASI AKADEMIK UNIVERSITAS LAMPUNG VERSI ANDROID

Oleh

SAIFUL ANWAR

Skripsi

Sebagai Salah Satu Syarat Untuk Mencapai Gelar
SARJANA KOMPUTER

Pada

Jurusan Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS LAMPUNG
BANDAR LAMPUNG
2018

Judul Skripsi	: **PENERAPAN DJANGO REST FRAMEWORK DAN TEKNOLOGI OTENTIKASI OAUTH 2.0 UNTUK SISTEM INFORMASI AKADEMIK UNIVERSITAS LAMPUNG VERSI ANDROID**
Nama Mahasiswa	: Saiful Anwar
Nomor Pokok Mahasiswa	: 1417051129
Jurusan	: Ilmu Komputer
Fakultas	: Matematika dan Ilmu Pengetahuan Alam

MENYETUJUI

1. Komisi Pembimbing

Dr. Ir. Kurnia Muludi, M.S.Sc.
NIP 19640616 198902 1 001

2. Ketua Jurusan Ilmu Komputer

Dr. Ir. Kurnia Muludi, M.S.Sc.
NIP 19640616 198902 1 001

MENGESAHKAN

1. Tim Penguji

Ketua : **Dr. Ir. Kurnia Muludi, M.S.Sc.** .....

Penguji I
Bukan Pembimbing : **Dwi Sakethi, S.Si., M.Kom.** .....

Penguji II
Bukan Pembimbing : **Dr. rer.nat Akmal Junaidi, M.Sc.** .....

2. Dekan Fakultas Matematika dan Ilmu Pengetahuan Alam

Prof. Warsito, S.Si., D.E.A., Ph.D.
NIP 19710212 199512 1 001

Tanggal Lulus Ujian Skripsi : **15 Februari 2018**

PERNYATAAN

Saya yang bertanda tangan di bawah ini, menyatakan bahwa skripsi saya yang berjudul "**Penerapan Django REST Framework dan Teknologi Otentikasi OAuth 2.0 Untuk Sistem Informasi Akademik Universitas Lampung Versi Android**" merupakan karya saya sendiri dan bukan karya orang lain. Semua tulisan yang tertuang di skripsi ini telah mengikuti kaidah penulisan karya ilmiah Universitas Lampung. Apabila dikemudian hari terbukti skripsi saya merupakan hasil penjiplakan atau dibuat orang lain, maka bersedia menerima sanksi berupa pencabutan gelar yang telah saya terima.

Bandar Lampung, Februari 2018

Saiful Anwar
NPM 1417051129

RIWAYAT HIDUP

Penulis dilahirkan pada tanggal 27 Januari 1996 di Tulang Bawang Barat, sebagai anak ketiga dari empat bersaudara dengan Ayah bernama Saifudin dan Ibu Istiqomah.

Penulis menyelesaikan pendidikan formal pertama kali di Taman Kanak-Kanak ABA Tulang Bawang Barat dan selesai pada tahun 2002. Pendidikan dasar di SDN 1 Tri Tunggal Jaya Tulang Bawang Barat dan selesai pada tahun 2008. Pendidikan menengah pertama di SMPN 1 Gunung Agung Tulang Bawang Barat diselesaikan pada tahun 2011, kemudian melanjutkan ke pendidikan menengah atas di SMAN 1 Gunung Agung Tulang Bawang Barat yang diselesaikan pada tahun 2014.

Pada tahun 2014 penulis terdaftar sebagai mahasiswa Jurusan Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Lampung dengan jalur Bidikmisi. Selama menjadi mahasiswa beberapa kegiatan yang dilakukan penulis antara lain.

1. Pada bulan Februari 2015 sampai dengan Oktober 2016 penulis bekerja sebagai Android Developer di PT. Moonray Artha Gemilang.

2. Pada bulan Januari 2015 sampai Februari 2015 penulis *internship* atau magang di UPT. TIK di Universitas Lampung.
3. Pada bulan Februari 2015 sampai Desember 2016 penulis bekerja sebagai Android Programmer di PT. Moonray Artha Gemilang.
4. Pada bulan Agustus 2016 sampai dengan Desember 2016 penulis bekerja sebagai Programmer di CV. Denawa.
5. Pada bulan November 2016 sampai dengan Mei 2017 penulis bekerja sebagai Manager IT dan Business di PT. Anco Jaya Telekekomunikasi.
6. Pada bulan Januari 2017 penulis melaksanakan kerja praktik di PT. Anco Jaya Telekomunikasi.
7. Pada bulan Novermber 2017 sampai saat ini penulis sebagai Founder di Capung Technology.
8. Pada bulan Juli penulis melaksanakan Kuliah Kerja Nyata di Desa Batu Agung, Kecamatan Merbau Mataram, Kabupaten Lampung Selatan, Lampung.

PERSEMBAHAN

Puji dan syukur saya panjatkan kepada Allah SWT atas segala berkah-Nya sehingga skripsi ini dapat terselesaikan.

Kupersembahkan karya ini kepada :

Teristimewa kedua orang tuaku, Bapak Saifudin dan Ibu Istiqomah yang telah membesarkan, mendidik, memberikan doa, dukungan dan semangat untuk kesuksesanku. Terima kasih atas semua perjuangan, pengorbanan, kesabaran dan kasih sayang telah kalian berikan untukku. Serta kakak dan adikku yang aku sayangi dan keluarga besar tercinta

Keluarga Ilmu Komputer 2014,

Serta Almamater Tercinta, Universitas Lampung.

MOTTO

“(yaitu) orang-orang yang beriman dan hati mereka manjadi tenteram dengan mengingat Allah. Ingatlah, hanya dengan mengingati Allah-lah hati menjadi tenteram” (Q.S Ar-Ra’d: 28)

“Jangan Melihat Dimana Anda terjatuh tapi lihat dimana Anda tergelincir” (Kutipan Kata Bijak)

“*Fight like Tiher Win like Champion*” (Darmadi Darmawangsa dan Imam Munadhi, 2009)

SANWACANA

Puji syukur penulis panjatkan ke hadirat Alloh SWT atas berkah, rahmat, dan hidayah-Nya penulis dapat menyelesaikan skripsi yang berjudul "Penerapan Django REST Framework dan Teknologi Otentikasi OAuth 2.0 Untuk Sistem Informasi Akademik Universitas Lampung Versi Android" dengan baik dan lancar.

Terima kasih penulis ucapkan kepada semua pihak yang telah membantu dan berperan besar dalam menyusun skripsi ini, antara lain.

1. Kedua orang tua tercinta, Bapak Saifudin dan Ibu Istiqomah, Kakakku tercinta Habibbulloh dan Fina Fitriana, Adikku tercinta Intan Nur Laila, dan Keluarga Besar yang selalu memberi doa, motivasi dan kasih sayang yang tak terhingga. Satu-satunya alasan penulis untuk berdiri kembali ketika berungkali jatuh. Cinta dan kasihmu tidak akan pernah mampu untukku balas. Semoga Alloh SWT memberikan keberkahan dalam hidup dan diberikan kebahagiaan dunia dan akhirat.
2. Bapak Dr. Ir. Kurnia Muludi, M.S.Sc. sebagai pembimbing utama dan juga selaku Ketua Jurusan Ilmu Komputer FMIPA Universitas Lampung, yang telah membimbing penulis dan memberikan ide, kritik serta saran sehingga penulisan skripsi ini dapat diselesaikan.

3. Bapak Dwi Sakethi, S.Si., M.Kom. sebagai pembahas pertama, yang telah memberikan masukan yang bermanfaat dalam perbaikan skripsi ini.
4. Bapak Dr. rer.nat Akmal Junaidi, M.Sc. sebagai pembahas kedua, yang telah memberikan masukan yang bermanfaat dalam perbaikan skripsi ini.
5. Bapak Ir. Machudor Yusman, M. Kom selaku pembimbing akademik penulis.
6. Bapak Prof. Warsito, S.Si., D.E.A., Ph.D. selaku Dekan FMIPA Universitas Lampung.
7. Bapak dan Ibu Dosen Jurusan Ilmu Komputer FMIPA Universitas Lampung yang telah memberikan ilmu dan pengalaman dalam hidup untuk menjadi lebih baik.
8. Ibu Ade Nora Maela, Mas Zainuddin dan Mas Ardi Novalia yang telah membantu segala urusan administrasi penulis di Jurusan Ilmu Komputer.
9. Noni Kurniasih seorang adik, sahabat, dan teman terdekat penulis. Yang selalu berusaha membuat penulis tersenyum, sangat sabar membantu, memberikan semangat, motivasi, dan berbagi cerita dan suka duka bersama penulis. Semoga apa yang menjadi cita, visi dan misi di ridhoi oleh Alloh SWT.
10. Kakak seperjuangan Doris Hermawan A.md., Rahmat Widodo, S. Kom., dan Wibi Cahyo Hastono, S. Kom. yang telah mewarnai, sebagai guru, rekan bercanda, dan rekan kerja penulis. Yang telah bersama-sama jatuh bangun membangun mimpi. Semoga selalu diberikan keberkahan dalam hidup dan meraih cinta serta cita.
11. Rekan seperjuangan Faiz Azmi Rekatama, Amrullah Subekti, Muammar Rizki Fadhillah Ibrahim, Niki Rahmadi Wiharto, dan Feri Krisnanto yang telah menemani, teman diskusi, rekan kerja, rekan bercanda, rekan yang perduli dan

rekan yang membangun mimpi penulis. Bersama-sama untuk berjuang mencapai gelar S. Kom. Semoga kita bisa menggapai cita-cita kita dan meraih kesuksesan kita serta selalu menjadi sahabat penulis.

12. Rekan seperjuangan Devi Ranita, Syifa Trianingsih, Tri Lestari dan Nuha Hanifah Azmi. Bersama-sama untuk berjuang mencapai gelar S. Kom. Semoga kita bisa menggapai cita-cita kita dan meraih kesuksesan kita serta selalu menjadi sahabat penulis.
13. Rekan seperjuangan kakak Ririn, adik atin, adik ivan, aris, arif. Yang telah memberi warna semasa perkuliahan.
14. Keluarga Ilmu Komputer 2014 yang tidak bisa penulis sebut satu persatu. Keluarga kedua penulis, rekan kelompok, rekan diskusi, rekan bercanda dan telah memberi arti dan warna serta pengalaman tak ternilai semasa duduk dibangku kuliah.
15. Keluarga KKN Desa Batu Agung, yang telah mengajari ilmu bermasyarakat penulis dan telah memberikan kenyaman dan ruang berekspresi penulis.
16. Seluruh kakak-kakak tingkat Ilmu Komputer yang tidak bisa disebutkan satu persatu, terimakasih atas ilmu-ilmu yang diberikan, nasihat, arahan, semangat dan dukungan kakak-kakak dalam menghadapi perkuliahan.
17. Seluruh adik-adik tingkat Ilmu Komputer yang tidak bisa disebutkan satu persatu yang telah menjadi warna selama masa perkuliahan penulis.
18. Keluarga bidimisi 2014 yang telah memberikan semangat dalam menempuh perkuliahan penulis. Menjadi tempat penulis mengasah soft skill dan menjadi renungan dan semangat tanggung jawab dalam menyelesaikan pendididkan, penelitian dan pengabdian penulis.

19. Teman-teman Himakom yang sudah mengajari banyak hal dalam berorganisasi, memberikan banyak pengalaman, berjuang bersama memajukan Himakom, berjuang bersama membawa nama baik Jurusan Ilmu Komputer. Semoga Himakom semakin sukses untuk kedepannya. Himakom, MAKIN JAYA.
20. Teman-teman Asisten Dosen yang juga menjadi keluarga kedua, mengajari banyak hal dalam berorganisasi, memberi banyak pengalaman, berjuang bersama memajukan Lab Ilmu Komputer.

Penulis menyadari bahwa skripsi ini masih jauh dari kata semupurna, semoga skripsi ini membawa manfaat dan keberkahan bagi semua civitas Ilmu Komputer Unila.

Bandar Lampung, 15 Februari 2018

Saiful Anwar
NPM 1417051129

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Gambar Halaman

DAFTAR TABEL

Tabel Halaman

I. PENDAHULUAN

A. Latar Belakang dan Masalah

Dalam era globalisasi penggunaan teknologi informasi dan komunikasi di perguruan tinggi merupakan hal penting yang seharusnya ada dalam meningkatkan mutu kualitas pendidikan. Tersedianya sistem informasi yang baik akan sangat menunjang kegiatan pendidikan pada suatu lembaga atau institusi pendidikan. Sarana atau media informasi penting yang berada di lembaga pendidikan salah satunya yaitu sistem informasi akademik yang meliputi pengolahan data entitas yang terkait (mahasiswa, dosen, karyawan, dekan, dan rektor), mata kuliah, jadwal kuliah, nilai mahasiswa, absensi mahasiswa dan keuangan (dalam hal ini adalah SPP mahasiswa). Itulah sebabnya, manajemen pengelolaan perguruan tinggi dewasa ini memanfaatkan teknologi informasi. Karena penggunaan teknologi informasi dan komunikasi ternyata berdampak lebih efektif, efisien dan optimal dibanding dengan cara-cara manual.

Dengan berkembangnya teknologi informasi yang sangat pesat, kemungkinan terjadinya gangguan keamanan informasi juga semakin meningkat. Masalah keamanan merupakan salah satu aspek penting dalam penggunaan sistem informasi akademik, seringkali keamanan ditempatkan

pada urutan terakhir dalam daftar yang dianggap penting. Saat ini sangat dibutuhkan sebuah sistem keamanan sistem informasi akademik yang baik untuk menjaga informasi yang dimiliki oleh perguruan tinggi, meskipun pada kenyataannya belum ada sebuah keamanan sistem informasi yang sempurna karena selalu saja ada celah atau cara untuk menembus keamanan sebuah sistem informasi.

Universitas Lampung (Unila) sebagai salah satu institusi pendidikan tinggi negeri tertua di Lampung memiliki jumlah mahasiswa 32.903 orang (2014), pengajar 1.164 orang (2012), tenaga administrasi 673 orang (2012) dan akan meningkat dalam setiap tahunnya. Berdasarkan *https://www.unila.ac.id/*, pada tahun 2014, Sistem Informasi Akademik (SIAKAD-Online) Universitas Lampung yang beralih ke versi terbaru yakni SIAKAD-Online v4.0 sebelumnya adalah v3.0 dikarenakan adanya *bugs* dan *password user* yang bisa dilihat pengguna lain. Sistem yang dikembangkan selama ini *nonscalable* atau sangat sulit untuk dikembangkan. Terbatas pada *resources* dan jika akan di-upgrade butuh biaya miliaran (Universitas Lampung, 2014).

Dalam rangka menjamin aksesibilitas dan ekuitas pendidikan tinggi sebagaimana tercantum dalam misi Rencana Pembangunan Jangka Panjang (RPJP) 2005-2025 maka diperlukannya sebuah sistem informasi akademik yang memiliki kemampuan dalam menangani transaksi data, otentikasi dan otorisasi pengguna serta memotong alur yang rumit. Masalah akan timbul ketika akan mengintegrasikan data dan fungsi yang berada pada *platform*

yang berbeda-beda tersebut. Di samping itu juga sekarang terdapat kecenderungan dalam hal pemanfaatan sistem informasi menggunakan berbagai macam *device* seperti *handphone* dan PDA, sehingga rancangan sistem informasi universitas akan semakin kompleks.

Python merupakan salah satu bahasa pemrograman populer yang digunakan oleh banyak *developer*. Menurut survei bahasa pemrograman versi *www.tiobe.com* (TIOBE, 2017). Python berada diperingkat ke-5 pada tahun 2017 dapat dilihat pada Gambar 1. Selain itu, Python juga bisa digunakan untuk enterprise. Dalam tingkatan bahasa pemrograman, Python termasuk *high level language*. Python menjadi salah satu bahasa pemrograman yang dapat digunakan untuk membangun aplikasi, baik itu berbasis *desktop*, *web* ataupun berbasis *mobile*.

Apr 2017	Apr 2016	Change	Programming Language	Ratings	Change
1	1		Java	15.568%	-5.28%
2	2		C	6.966%	-6.94%
3	3		C++	4.554%	-1.36%
4	4		C#	3.579%	-0.22%
5	5		Python	3.457%	+0.13%
6	6		PHP	3.376%	+0.38%

Gambar 1 Peringkat Bahasa Pemrograman (TIOBE, 2017)

Python memiliki beberapa *web framework* salah satunya adalah Django. Django merupakan sebuah *web framework* berbasis Python yang mendukung pembuatan sebuah website secara *rapid development* dengan desain yang elegan. Django merupakan *web framework* yang dirancang dan

dibangun oleh Adrian Holovaty dan Jacob Kaplan Moss. Menurut survei *framework* Python versi *hotframeworks.com* (HotFramwork, 2017), *framework* Django berada di peringkat pertama.

Aplikasi berbasis *mobile* tentunya sekarang sangat banyak digunakan oleh masyarakat karena penggunanya yang mudah dan gampang dibawa kemana-kemana dan aplikasi *mobile* berbasis *Android* yang sedang ramai digunakan oleh masyarakat. Berdasarkan website *https://www.kominfo.go.id* pada tahun 2017 dari jumlah penduduk Indonesia yang mencapai 250 juta jiwa adalah pasar yang besar. Pengguna *smartphone* Indonesia juga bertumbuh dengan pesat. Lembaga riset digital *marketing emarketer* memperkirakan pada 2018 jumlah pengguna aktif *smartphone* di Indonesia lebih dari 100 juta orang. Dengan jumlah sebesar itu, Indonesia akan menjadi negara dengan pengguna aktif *smartphone* terbesar keempat di dunia setelah Cina, India, dan Amerika (Kominfo, 2017). Selain itu menurut website *https://id.techinasia.com/*, pertumbuhan jumlah pengguna internet ini turut diiringi oleh meningkatnya jumlah pengguna layanan media sosial. Hanya berjumlah 79 juta pada tahun lalu, angka tersebut kini telah naik menjadi 106 juta pengguna. Para pengguna yang secara aktif menggunakan media sosial di perangkat *mobile* pun naik dari angka 66 juta menjadi 92 juta. Perkembangan dunia digital Indonesia disajikan pada Gambar 2 (Tech in Asia, 2016).

Gambar 2 Perkembangan Pengguna Internet di Indonesia Tahun 2017 (Tech in Asia, 2016)

Permasalahan otentikasi dan otorisasi pengguna dalam sistem informasi akademik menjadi hal yang serius untuk diperhatikan. Protokol OAuth (*Open Authorization*) adalah protokol otentikasi yang bersumber dari layanan penyedia API, yang memberikan kuasa kepada seseorang untuk mendapatkan hak akses mereka dengan menukarkan *credential* (*username* dan *password*) menjadi akses *token*.

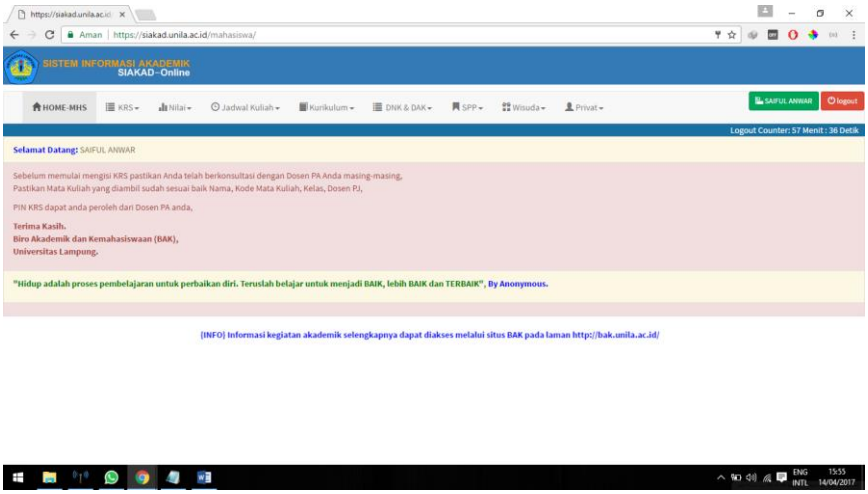
Otentikasi dengan menggunakan OAuth mengatur Sistem Informasi Akademik Unila sedemikian rupa agar pengguna tetap aman menggunakan layanan aplikasi dengan mempercayakan kehadiran pihak ketiga (*third party*) yang telah dipercayai sebagai sumber *server* (*resource server*) yang mempunyai kuasa untuk menertibkan dan mengatur segala credential yang ada. *Resource server* juga merupakan *authorization server* yang melakukan proses otorisasi *client* permintaan API dengan menukarkan *authorization code* dalam bentuk akses *token*. Biasanya *access token* diterbitkan bersamaan dengan *refresh token*.

RESTful *web service* memungkinkan suatu aplikasi "berbicara" dengan aplikasi lainnya. RESTful *web service* suatu gaya arsitektur yang bertujuan untuk meminimalkan latensi dan komunikasi jaringan, sementara pada saat yang bersamaan berusaha untuk memaksimalkan independensi dan skalabilitas dari implementasi komponen (Fielding, 2000). Dengan sistem *web service* tersebut diharapkan akan meningkatkan kolaborasi antar pemrogram dan antar organisasi bisnis, yang memungkinkan suatu fungsi dalam *web service* dapat digunakan oleh aplikasi lain tanpa perlu mengetahui detail pemrograman yang terdapat di dalamnya. RESTful *web service* cocok untuk menyelesaikan masalah pada sistem bisnis konsep lama ke sistem bisnis terintegrasi, sehingga dengan satu model konsep bisnis dapat diakses dan dipergunakan oleh bermacam-macam aplikasi dan *device*.

Perantara pertukaran data dalam *web service* biasanya dibangun dengan *metalanguage* XML, meskipun banyak perantara lain yang dapat digunakan seperti JSON, RESTful dan lain sebagainya. Secara umum, *web service* diidentifikasikan dengan menggunakan URL seperti hanya *web* pada umumnya, misal : *www.namaweb.com/service/service.php?wsdl*, untuk memperoleh berbagai service dan *library* yang disediakan oleh *web service* (Lim, 2008). Berbeda dengan URL *web* pada umumnya, URL *web service* hanya menggandung kumpulan informasi, perintah, konfigurasi atau sintak yang berguna membangun sebuah fungsi-fungsi tertentu dari aplikasi.

Dengan dibangunnya arsitektur *web service*, diharapkan pengembangan aplikasi Sistem Informasi Akademik Unila dapat berjalan ke semua

platform, dapat memberikan *interface* yang stabil, mengurangi biaya integrasi aplikasi enterprise dan mudah dalam mengembangkan dengan *semantic transport* tambahan. Tampilan halaman *home* mahasiswa berbasis website dapat dilihat pada Gambar 3.



Gambar 3 Tampilan Halaman *Home* Mahasiswa (Siakad Unila, 2017)

Sudah banyak penelitian dalam menerapkan teknologi *web service* antara lain dalam sistem informasi wisata alam dan wisata kuliner DIY, sistem informasi daftar pemilih tetap dan perancangan dan implementasi *resource server*, pengembangan *web service* pengurai morfologi bahasa Indonesia pada *language grid*, dan perancangan dan implementasi *resource server* dan *authorization server* menggunakan teknologi otentikasi OAuth *2*.

Namun untuk sistem informasi akademik Universitas Lampung belum mengimplementasikan *web service*. Oleh karena itu, pada penelitian akan dibuat aplikasi "SIAKAD Online" berbasis *Android* dengan menerapkan *web service* dengan teknologi otentikasi OAuth 2.0 untuk otorisasi

penggunaannya, dimana aplikasi ini akan memberikan informasi mahasiswa (jadwal, nilai, ruang kelas), dosen dan pengembangan media sosial sederhana untuk komunikasi antar dosen dan mahasiswa.

B. Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, didapatkan masalah pokok yang perlu diselesaikan dan menjadi dasar dalam pengerjaan skripsi ini sebagai berikut.

1. Bagaimana membangun sebuah aplikasi SIAKAD Unila berbasis *Android* dengan menerapkan Django REST *Framework*.
2. Bagaimana mempelajari, merancang, dan mengimplementasikan *resource server* dan *authorization server* pada proses otentikasi OAuth 2.0 pada aplikasi SIAKAD Unila.

C. Batasan Masalah

Dalam perancangan dan penerapan *web service* (RESTful) pada pengembangan aplikasi SIAKAD Unila berbasis *Android* ini diberikan batasan masalah sebagai berikut.

1. Aplikasi ini hanya dapat dioperasikan dalam *smartphone* yang mendukung OS (*Operating System*) *Android*.
2. Aplikasi ini membutuhkan koneksi internet untuk menjalankannya.
3. Aplikasi ini didukung dengan *web service* dan otentikasi *user* dengan teknologi OAuth 2.0.

4. Aplikasi akan mendaftarkan *id* perangkat ke *server* sebagai identifikasi *user login* menggunakan perangkat tersebut.
5. Aplikasi akan *refresh* token login apabila token *expired*.
6. Aplikasi akan mengkonfirmasi *user* apabila terdapat pengguna *login* menggunakan perangkat lain.
7. Aplikasi menerapkan pengamanan berganda (bersifat opsional) terhadap *account* yang *login* pada aplikasi dengan menggunakan nomor telepon.
8. Aplikasi ini dapat menampilkan pengumuman dan *push notification* pengumuman masuk.
9. Aplikasi ini menyediakan layanan jadwal kuliah dilengkapi dengan *push notification* jadwal yang akan berlangsung.
10. Aplikasi ini menampilkan dan mencetak Kartu Hasil Studi (KHS) berdasarkan semester yang diambil oleh mahasiswa.
11. Aplikasi ini menampilkan transkrip nilai berdasarkan semester yang diambil oleh mahasiswa.
12. Aplikasi ini menyediakan layanan Kartu Rencana Studi (KRS) meliputi mengisi KRS, mengedit KRS, dan menampilkan KRS mahasiswa.
13. Aplikasi ini menampilkan *list* kurikulum berdasarkan program studi yang ditempuh oleh mahasiswa.
14. Aplikasi ini menyediakan *list* dan notifikasi pemberitahuan SPP yang telah dibayarkan oleh mahasiswa.
15. Aplikasi ini merekomendasikan, menampilkan dan notifikasi pemberitahuan sosial komunikasi antara mahasiswa dan dosen

berdasarkan mata kuliah yang diambil oleh mahasiswa dan diampu oleh dosen.

16. Aplikasi ini menyediakan *button* wisuda untuk membuka website Sistem Informasi Wisuda Online.
17. Aplikasi ini menyediakan fungsi mengubah bahasa aplikasi.
18. Aplikasi ini dilengkapi dengan Halaman Info untuk mengetahui versi aplikasi dan tentang pengembang, Halaman *Manual Book* untuk mengunduh manual penggunaan aplikasi "SIAKAD *Mobile*", Halaman FAQ yang berisi untuk mengetahui pertanyaan yang sering ditanyakan mengenai aplikasi, dan Halaman *Contact List* yang berisikan daftar kontak informasi mengenai keadaan di program studinya.

D. Tujuan Penelitian

Tujuan dari penelitian ini adalah merancang dan membangun sebuah aplikasi SIAKAD Unila berbasis *Android* dengan menerapkan Django REST *Framework* dan otentikasi pengguna dengan teknologi OAuth 2.0.

E. Manfaat Penelitian

Manfaat yang diperoleh dari penelitian ini adalah sebagai berikut.

1. Manfaat Praktis

a) Memberikan kemudahan bagi *user* (mahasiswa dan dosen) dalam menggunakan aplikasi SIAKAD tanpa selalu login ke *website*.
b) Meningkatkan rasa aman bagi *user* dalam mengakses sistem informasi akademik melalui aplikasi.

c) Melibatkan *user* dalam mengontrol aplikasi dengan menggunakan *push notification* pada layanan pengumuman, SPP dan sosial komunikasi.

2. Manfaat Akademis

a) Hasil penelitian dapat menambah pengetahuan baru untuk penulis.
b) Mengembangkan dan mengaplikasikan ilmu yang telah didapatkan selama kuliah.
c) Untuk dijadikan acuan terhadap pengembangan ataupun pembuatan dalam penelitian yang sama.

II. TINJUAN PUSTAKA

A. Pengertian Sistem dan Akademik

1. Pengertian Sistem

Menurut McLeod (2004) analisis sistem adalah penelitian atas sistem yang telah ada dengan tujuan untuk merancang sistem baru atau diperbarui.

Sistem dapat didefinisikan ke dalam dua kelompok pendekatan, yaitu pendekatan yang menekankan pada prosedurnya dan pendekatan yang menekankan pada komponen atau elemennya (Hartono, 2007).

Pendekatan yang menekankan pada prosedurnya menyebutkan bahwa sistem adalah suatu jaringan kerja dari prosedur-prosedur yang saling berhubungan, berkumpul bersama-sama untuk melakukan suatu kegiatan atau untuk menyelesaikan suatu sasaran yang tertentu.

Sedangkan pendekatan sistem lebih menekankan pada komponen atau elemennya menyebutkan bahwa sistem adalah kumpulan dari elemen-elemen yang berinteraksi untuk mencapai tujuan tertentu (Hartono, 2007).

Dengan dua pengertian diatas penulis dapat menyimpulkan bahwa pengertian sistem adalah kumpulan dari beberapa elemen yang saling berinteraksi dan bekerja sama untuk mencapai suatu tujuan tertentu.

Suatu sistem mempunyai karakteristik atau sifat-sifat tertentu yaitu (Hartono, 2007).

a) Komponen Sistem, suatu sistem terdiri dari sejumlah komponen yang saling berinteraksi, yang artinya saling bekerja sama membentuk satu kesatuan.
b) Batas Sistem (*boundary*), merupakan daerah yang membatasi antara suatu sistem dengan sistem yang lainnya atau dengan lingkungan luarnya.
c) Lingkungan Luar Sistem (*environment*), apa pun diluar batas sistem yang mempengaruhi operasi sistem.
d) Penghubung Sistem (*interface*), merupakan media penghubung antara satu sub sistem dengan sub sistem yang lainnya.
e) Masukan Sistem, merupakan energi yang dimasukkan ke dalam sistem.
f) Pengolahan Sistem, sistem dapat mempunyai suatu bagian pengolah atau sistem itu sendiri sebagai pengolahnya. Pengolah akan merubah masukan menjadi keluaran.
g) Sasaran dan tujuan Sistem, merupakan apa yang harus dicapai sebuah sistem. Suatu sistem dikatakan berhasil bila mengenai sasaran dan tujuannya.

2. Pengertian Akademik

Kata akademik berasal dari bahasa Yunani yakni *academos* yang berarti sebuah taman umum (plasa) di sebelah barat laut Athena. Nama *academos* adalah nama seorang pahlawan yang terbunuh pada saat perang legendaris Troya. Pada plasa inilah filosof Scorates berpidato dan membuka arena perdebatan tentang berbagai hal. Tempat ini juga menjadi tempat Plato melakukan dialog dan mengajarkan pikiran-pikiran filosofinya kepada orang-orang yang datang. Sesudah itu, kata *academos* berubah menjadi akademik, yaitu semacam tempat perguruan. Para pengikut perguruan tersebut disebut *academist*, sedangkan perguruan semacam itu disebut *academia*. Berdasarkan hal ini, inti dari pengertian akademik adalah keadaan orang-orang bisa menyampaikan dan menerika gagasan, pemikiran, ilmu pengetahuan, dan sekaligus dapat mengujinya secara jujur, terbuka, dan leluasa (Fadjar dan Effendy, 1989).

3. Konsep Dasar Sistem Informasi Akademik

Albahara (2005) mendefinisikan sistem informasi sebagai berikut.

a) Suatu sistem yang dibuat oleh manusia yang terdiri dari komponen-komponen dalam organisasi untuk mencapai suatu tujuan yaitu menyajikan informasi.
b) Sekumpulan prosedur organisasi yang pada saat dilaksanakan akan memberikan informasi bagi pengambil keputusan dan atau untuk mengendalikan organisasi.
c) Suatu sistem di dalam suatu organisasi yang mempertemukan kebutuhan pengelolaan transaksi, mendukung operasi, bersifat manajerial dan kegiatan strategis dari suatu organisasi dan menyediakan pihak luar tertentu dengan paloran-loran yang dipertemukan.

Sistem informasi yang didefinisikan sebagai suatu sistem di dalam suatu organisasi yang merupakan kombinasi dari orang-orang, fasilitas, teknologi, media, prosedur-prosedur dan pengendalian yang ditujukan untuk mendapatkan jalur komunikasi penting, memproses tipe transaksi rutin tertentu, memberi sinyal kepada manajemen dan yang lainnya terhadap kejadian-kejadian internal dan eksternal yang penting dan menyediakan suatu dasar informasi untuk pengambilan keputusan yang baik.

Dari beberapa penjelasan diatas, penulis mendefinisikan sistem informasi akademik sebagai suatu sistem di dalam suatu lembaga pendidikan yang merupakan kombinasi dari orang-orang, fasilitas,

teknologi, media, dan prosedur-prosedur dalam mengolah, menyimpan, dan mendistribusikan data dan informasi yang berkaitan dengan pendidikan atau akademik (Albahara, 2005).

B. *Resource-oriented Architecture* (ROA)

Resource adalah segala sesuatu yang dapat disimpan pada komputer dan direpresentasikan dalam bentuk aliran bit: dokumen, *record* pada basis data, atau hasil akhir dari eksekusi suatu algoritma. *Resource* akan bermanfaat jika memiliki minimal sebuah *Uniform Resource Identifier* (URI) supaya dapat diakses. ROA merupakan bentuk kongkret dari RESTful *web service* sebagai salah satu cara untuk memecahkan permasalahan menjadi solusi: komposisi dari URI, HTTP, dan XML yang saling bekerja sama (Richardson dan Ruby, 2007).

Resource merupakan abstraksi utama pada RESTful. Resource dapat statis, yang berarti tidak berubah dari waktu ke waktu, atau dapat pula bersifat dinamis yang terus berubah seiring dengan waktu (Roth, 2009).

ROA memiliki dua buah fitur utama, yaitu *addressability* dan *statelessness*. *Addressabililty* berarti bahwa sebuah aplikasi dikatakan *addressable* jika aplikasi tersebut menampakkan data yang dimilikinya sebagai suatu *resource* dan memiliki URI sendiri, misalnya sebuah URI tentang *resource jelly fish*: *http://www.google.com/search?q=jellyfish*s. URI yang *addressable* memungkinkan pencatatan terhadap URI tersebut sehingga jika akan digunakan lagi, yang perlu dilakukan hanya mengetik URI tersebut

pada peramban. Anggap saja website Google tidak *addressable*, maka tidak mungkin menyimpan URI tersebut pada sebuah catatan, sebaliknya harus dilakukan secara manual: membuka peramban, ketik *www.google.com* di peramban, ketik '*jellyfish*' pada kotak pencarian, lalu klik tombol '*Penelusuran Google*'.

Fitur kedua yaitu *statelessness* yang berarti bahwa semua *request* HTTP yang terjadi harus dilakukan dalam keadaan terisolasi. Saat *client* melakukan *request* kepada *server*, maka *request* tersebut harus berisi semua informasi yang dibutuhkan oleh *server* untuk memprosesnya lebih lanjut. *Server* tidak boleh bergantung pada informasi *request* sebelumnya yang dilakukan oleh *client* tersebut. Secara praktik hal ini jika dihubungkan dengan sifat *addressability*, maka berarti bahwa *state* dari *server* dapat dijadikan *resource* dan memiliki URI tersendiri.

C. *Representational State Transfer* (REST)

Menurut Higashino *et al*. (2009) *web service* yang berbasis teknologi SOAP telah menghasilkan aplikasi pada banyak bidang. SOAP didesain untuk transparansi komunikasi dan keterikatan yang rendah antar aplikasi. SOAP tidak lepas dari kelemahan karena dikritik mengenai kompleksitas dan spesifikasi yang harus dipenuhi terlalu banyak. Selain itu permasalahan teknis yang terlalu terikat pada satu vendor tertentu menjadikan keterikatan rendah yang tidak dapat terwujud (Pautaso *et al.*, 2008).

Representational State Transfer (REST) adalah suatu gaya arsitektur yang bertujuan untuk meminimalkan latensi dan komunikasi jaringan, sementara pada saat yang bersamaan berusaha untuk memaksimalkan independensi dan skalabilitas dari implementasi komponen (Fielding, 2000). REST berkembang bersamaan dengan berkembangnya teknologi *web*, sehingga sering digunakan bersamaan dengan teknologi *Hypertext Transfer Protocol* (HTTP), dengan alasan inilah implementasi RESTful menggunakan teknologi HTTP disebut RESTful *HTTP*.

Sebagai sebuah gaya arsitektur pengembangan sistem, REST memiliki aturan yang menjadi ciri dasarnya (Fielding, 2000).

a) *Client Server*

   Terdapat *interface* seragam antara *client* dan *server*. Hal ini berarti *client* tidak berurusan dengan hal teknis yang menjadi tanggung jawab *server*, misalnya *database*, bahasa pemrograman, dan lain-lain. Hal ini bertujuan untuk *portability* atau kemudahan migrasi pada *client*. Pada sisi lain, *server* tidak berurusan dengan hal teknis pada *client*, misalnya tampilan sistem atau *state* dari aplikasi *client* sehingga pengembangan sistem pada *server* menjadi lebih sederhana dan lebih mudah dikembangkan (*scalable*). *Server* dan *client* dapat diganti dan dikembangkan secara terpisah selama *interface* yang digunakan di antara keduanya tidak diubah.

b) *Stateless*

   Komunikasi *client-server* selanjutnya dibatasi dengan aturan tidak diperbolehkannya *state* dari suatu *client* disimpan pada *server*. Setiap

request dari *client* harus disertai dengan seluruh informasi yang dibutuhkan oleh *server* untuk memproses *request* tersebut. Hal ini bertujuan supaya *server* menjadi lebih tidak tergantung pada *client* sehingga mudah diganti terutama saat terjadi kegagalan jaringan (*reliability*).

c) *Chaceable*

Respon yang diberikan kepada *client* dapat disimpan sementara dalam *cache client*. Hal ini berakibat bahwa respon yang diberikan oleh *server* harus menyertakan informasi/metadata tentang kemampuan *resource* tersebut untuk disimpan sementara atau tidak. Hal ini bertujuan untuk mengurangi jumlah komunikasi yang ada sehingga meningkatkan performa *server*.

d) *Layered System*

Client tidak perlu tahu apakah *server* yang melayani *request*-nya merupakan *server* utama ataukah bukan. Hal ini bertujuan untuk meningkatkan skalabilitas *server* misalnya menambahkan *server* untuk melakukan *load balancing* dan *cache* yang dapat diakses siapapun.

e) *Code on Demand*

Client server dapat mengkustomisasi fungsionalitas dari *client* dengan mengirimkan kode yang dapat dieksekusi pada *client*, misalkan *JavaScript*.

f) *Uniform Interface*

Simplifikasi arsitektur sistem yang dikembangkan berdasarkan RESTful dilakukan salah satunya dengan menggunakan *Uniform Resource Interface* seperti pada Tabel 1. *Interface* ini terdiri atas sekumpulan

operasi yang telah terdefinisi untuk mengakses dan memanipulasi *resource*. *Interface* yang sama dapat digunakan pada *resource* yang berbeda, dan tidak tergantung pada *resource* yang digunakan.

Tabel 1 *Uniform Resource Interface* yang Digunakan pada RESTful HTTP (Roth, 2009)

Nama	Pemanfaatan	Sifat
GET	Menerima respon (*retrieve*)	*Safe,* Idempoten
DELETE	Menghapus *resource*	Idempoten
PUT	Memperbarui *resource* dengan mengganti yang lama	Idempoten
POST	Membuat *resource* baru atau sub-*resource* baru dengan *id* dikelola oleh *server*	

Fielding (2000) membagi sifat operasi tersebut menjadi *safe* dan *idempotent*. Operasi bersifat *safe* berarti saat suatu *request* dilakukan terhadap *server*, maka *state* dari *resource* yang diminta tidak berubah. Operasi *GET* termasuk *safe* karena saat suatu *request* dilakukan terhadap suatu *resource*, *resource* tersebut akan tetap sama. Di sisi lain operasi *PUT* termasuk tidak *safe* karena saat *request* dilakukan menggunakan operasi tersebut maka *resource* tersebut menjadi berubah. Operasi bersifat *idempotent* berarti hasil dari operasi yang sukses tidak tergantung dari jumlah operasi tersebut dieksekusi, misalkan operasi *PUT* untuk memperbarui suatu *resource* dapat dilakukan berulang kali dengan hasil akhir yang sama. Berbeda dengan *PUT*, operasi *POST* tidak bersifat *idempotent* yang berarti jika suatu operasi *POST* dikirimkan lebih dari sekali, hasil akhirnya bisa saja terdapat duplikasi *resource* pada *server*. Hal

tersebut disebabkan karena operasi *POST* digunakan untuk menciptakan *resource* baru tanpa melihat id spesifik pada *client*.

Fielding (2000) menjelaskan beberapa batasan yang menjadi ciri khas dari arsitektur REST:

a) Identifikasi dari *resource*, yang berarti setiap *resource* yang ada harus memiliki identitas yang standar, misalnya menggunakan URI.
b) Manipulasi *resource* dilakukan melalui representasi. *Resource* asli tidak diberikan kepada *client* melainkan representasinya dalam bentuk XML, JSON, ataupun HTML. Representasi tersebut terdiri atas data dan metadata yang menjelaskan data, misalnya sebuah instansi *content-type* pada *header* HTTP merupakan atribut dari metadata.
c) Pesan yang mendefinisikan dirinya sendiri, *server* akan mengolah *request* yang dikirim kepadanya melalui metadata yang ada pada *request* tersebut, misalnya melalui dari *request* tersebut.
d) *Hypermedia* sebagai pengatur transisi dari *state*. Transisi dari suatu *state* dilakukan menggunakan *hypermedia*.

Implementasi REST (RESTful) menjadikan beberapa aplikasi dapat berkomunikasi dan bekerja sama. Melalui *web service* tersebut, setiap aplikasi pada *web* memiliki potensi untuk mencapai aplikasi lainnya. Saat aplikasi bertukar informasi melalui standard *web service*, mereka dapat berkomunikasi secara terpisah dan lepas dari sistem informasi, pemrograman, *processor*, dan protokol internal (Breitman *et al*., 2007).

Menurut Filho dan Ferreira (2009), pendekatan *RESTful* pada dasarnya terdiri dari lima konsep (*resource*, representasi, *uniform identifier*, *unified interface*, dan lingkup eksekusi), dan tiga prinsip (*addressability*, *statelessness*, dan keterhubungan). Hal yang membedakan RESTful dengan *web service* lainnya adalah lingkup dari *request* dan *response* harus didefinisikan saat operasi berlangsung.

D. *Application State and Resource State*

Richardson dan Ruby (2007) menjelaskan *state* berarti suatu keadaan dari suatu informasi yang tersimpan pada suatu sistem di waktu tertentu. Sebagai contoh, jika terdapat suatu *resource* bernama *order*, akan terdapat dua keadaan/*state* dari *resource* tersebut: *order* telah terkirim atau *order* belum terkirim. Suatu *resource* jarang sekali hanya memiliki satu *state* dari waktu ke waktu. Perubahan suatu *state* tersebut disebut *state transition* dan bagian dari sistem yang mengatur transisi *state* tersebut disebut sebagai *state engine*.

State pada REST terdiri atas dua tipe: *application state* dan *resource state*. *Application state* berada pada mesin *client*, sedangkan *resource state* berada pada mesin *server*. *Application state* dapat digambarkan sebagai suatu kondisi dari aplikasi yang dijalankan pada *client*, pada kasus pencarian menggunakan mesin pencari (*search engine*), halaman hasil pencarian yang sedang dibuka oleh *client* dapat dikatakan sebagai *application state*. *Resource state* dapat dikatakan sebagai *persistence* dari *resource* yang ada

dan terletak di *server*. *Application state* dari *client* yang satu dengan yang lain bisa berbeda, sebaliknya *resource state* dari seluruh *client* adalah sama.

E. *Hypermedia As The Engine Of Application State* (HATEOAS)

W3C Glosary Dictionary (W3C Glosary Dictionary, 2003) mendefinisikan *hypertext* sebagai teks yang mengandung tautan ke teks lainnya. *Hypermedia* adalah istilah yang digunakan untuk *hypertext* yang mengacu tidak hanya kepada teks, tapi suara, video, grafik, dan lain-lain. HATEOAS (*Hypermedia As The Engine Of Application State*) adalah aturan yang membedakan arsitektur aplikasi berbasis REST dengan aplikasi lain yang berbasis *client-server*. Prinsip dasarnya adalah *client* berinteraksi dengan aplikasi jaringan melalui *hypermedia* yang diberikan secara dinamis oleh *server* (Fielding, 2000).

Filosofi pemanfaatan *hypermedia* adalah membuat aplikasi *client* dapat menelusuri seluruh *resource* yang ada di *server* maupun melakukan modifikasi terhadap *state* dari *resource* yang ada walaupun tanpa adanya antarmuka sehingga dapat dilakukan oleh aplikasi yang berjalan tanpa manusia. HATEOAS terdiri atas dua jenis: *link* dan *form.*

Link adalah atribut dari suatu *resource* yang berisi suatu *hypermedia*. *Form* adalah suatu representasi yang dikirim oleh *server* dalam bentuk yang deskriptif. Bentuk tersebut berisi semua informasi yang dibutuhkan oleh *client* untuk melakukan manipulasi terhadap *resource state* di *server*.

Komunikasi diawali oleh *client* dengan cara melakukan suatu *request* menggunakan URI statis yang sederhana. Interaksi lanjutan dari aksi sebelumnya akan dilakukan berdasarkan representasi *resource* yang dikembalikan oleh *server* dan relasi *link* (*hypermedia*) yang ada di dalamnya. Perubahan *application state* yang ada di *client* dilakukan melalui pemilihan *link* yang tersedia di dalam representasi yang dikirim oleh *server* seperti pada Gambar 4 berikut.

```
<photo xmlns:atom="http://www.w3.org/2005/Atom">
  ...
  <atom:link link href="http://east-nj1.photos.example.org/987/nj1-1234"
        type="image/jpeg"
        rel="alternate"
        title="Sunset view from our backyard"/>
</photo>
```

Gambar 4 Representasi *Photo* yang Berisi *Hypermedia* Jenis *Link* (Allamaraju, 2010)

F. *Template* URI

Menurut Gregorio *et al*. (2012), *template* URI adalah sekumpulan karakter kompak yang mendeskripsikan jangkauan dari suatu URI melalui ekspansi variabel. Sebagai contoh, URI *http://www.example.com/marketing/customer/{id}* adalah URI yang salah, karena mengandung karakter yang tidak diizinkan pada URI, yaitu kurung kurawal ({}), namun demikian URI tersebut adalah *template* URI yang sah. Karakter {id} adalah variabel yang dapat diganti dengan nilai lain yang mendefinisikan *id* dari *customer*.

G. Kode Respon HTTP

Salah satu fitur utama arsitektur RESTful adalah pemanfaatan kode respon HTTP (HTTP *response code*). RFC 2616 (Fielding *et al*, 1999) membagi kelompok kode menjadi beberapa bagian sesuai dengan informasi yang diberikan: 1xx (meta), 2xx (*success*), 3xx (*redirection*), 4xx (*client-side error*), dan 5xx (*server-side error*). Kode 1xx merupakan kelompok respon yang digunakan pada negosiasi antara *client* dengan *server*.

Kode 2xx mengindikasikan operasi yang dilakukan antara *client* dengan *server* berhasil dilakukan. Kelompok ini terdiri atas.

a. 200 (*Ok*). Kode ini mengindikasikan *server* sukses melakukan operasi yang diminta oleh *client*.
b. 201 (*Created*). *Server* mengirimkan kode ini jika *resource* telah tersimpan/tercipta sesuai dengan *request* yang dikirim oleh *client*.
c. 204 (*No Content*). Kode ini biasanya merupakan respon dari operasi *PUT*, *POST*, atau *DELETE* saat *server* tidak ingin mengembalikan representasi apa pun kepada *client*. Dapat pula digunakan untuk operasi *GET* terhadap *resource* yang ada namun memiliki representasi yang kosong.

Kelompok 3xx (*redirection*) menginformasikan kepada *client* bahwa *client* harus melakukan operasi lainnya hingga *request* dapat diproses sampai selesai.

Kelompok 4xx merupakan kelompok kode yang menandakan terjadi kesalahan yang bersumber pada *client*. Kelompok ini terdiri atas.

a) 400 (*Bad Request*). Merupakan kode generik dari kesalahan yang disebabkan oleh *client*.

b) 401 (*Unauthorized*). Kode ini mengindikasikan *resource* yang diminta oleh *client* merupakan *resource* yang membutuhkan otoritas tertentu yang tidak disertakan oleh *client*. Kode ini juga berarti *client* harus mengulangi *request* yang dilakukannya dengan menyertakan akun otorisasi yang dibutuhkan.

c) 403 (*Forbidden*). Kode ini berarti *server* tidak mengizinkan *client* untuk mengakses *resource* yang dituju. Hal ini dapat berarti karena *resource* tersebut hanya tersedia pada saat-saat tertentu atau hanya dapat diakses oleh alamat IP tertentu.

d) 404 (*Not Found*). Kode ini mengindikasikan bahwa URI yang diberikan tidak dapat dipetakan kepada *resource* yang ada, dengan kata lain *resource* pada URI yang diberikan tidak ada.

e) 405 (*Method Not Allowed*). Kode ini mengindikasikan *resource* yang dituju tidak mendukung operasi menggunakan *interface* yang diberikan oleh *client*. Sebagai contoh, *resource* yang hanya dapat dibaca akan mengembalikan kode 405 (*Method Not Allowed*) saat *client* melakukan operasi selain *GET*.

f) 406 (*Not AccepTabel*). Kode ini diberikan saat *server* tidak dapat memenuhi permintaan format representasi yang diminta oleh *client*.

g) 409 (*Conflict*). Kode ini diberikan saat *client* melakukan *request* untuk mengubah *state* dari *resource* menjadi tidak konsisten. Sebagai contoh menghapus suatu *resource* yang memiliki sub-*resource* yang tidak

kosong.

h) 415 (*Unsupported Media Type*). *Server* mengirimkan kode ini untuk memberitahukan *client* yang mengirimkan representasi data yang tidak dimengerti. Sebagai contoh *client* mengirimkan representasi dalam format JSON, sedangkan *server* hanya menerima format XML.

i) 417 (*Expectation Failed*). Saat *client* meminta izin *server* untuk mengirimkan suatu representasi kepada *server*, namun *server* tidak mau menerima representasi yang diberikan oleh *client*, maka *server* akan mengirimkan kode ini.

Kelompok kode 5xx menandakan terjadi suatu permasalahan yang disebabkan oleh *server* atau pada *proxy* yang digunakan. Kelompok ini terdiri atas.

a) 500 (*Internal Server Error*). Kode ini merupakan kode generik dari respon kesalahan yang terjadi oleh *server*.

b) 503 (*Service Unavailable*). Kode ini berarti *web service* yang dituju oleh *client* sedang bermasalah. *Server* berjalan namun karena suatu hal *server* tidak dapat menangani *request* dari *client*.

H. HTTP *Header*

HTTP *header* merupakan komponen dari *header* suatu pesan yang dikirimkan (*request*) dan diterima (*response*) pada protokol HTTP (RFC 2616). *Header* ini merupakan parameter dari transaksi yang terjadi antara

client dengan *server* pada transaksi HTTP. Beberapa *header* yang umum digunakan diantaranya (Richardson dan Ruby, 2007).

a) *Accept*. *Client* menggunakan ini untuk menginformasikan kepada *server* format representasi yang diharapkan dikirim oleh *server*.
b) *Authorization*. *Header* ini mengandung informasi otorisasi seperti *username* dan *password* yang dikodekan menggunakan skema yang disepakati antara *client* dan *server*.
c) *Connection*. *Header* ini digunakan oleh *proxy* yang menghubungkan oleh *server* dengan *client*, *proxy* HTTP selanjutnya akan menghapus *header* ini saat representasi diteruskan kepada *client*.
d) *Content-Encoding*. *Header* ini menjelaskan pengkodean yang digunakan pada representasi *response* yang diberikan oleh *server*.
e) *Content-Length*. *Header* ini berisi ukuran dari representasi data yang dikirimkan.
f) *Content-Type*. *Header* ini berisi jenis representasi yang ada pada *request*.
g) *Date*. Pada *request* yang berasal dari *client*, *header* ini berisi waktu *request* dikirimkan. Pada *response* yang diberikan oleh *server*, *header* ini berisi waktu *response* tersebut dikirim oleh *server*.
h) *Expires*. *Header* ini memberi tahu *client* atau proxy di antara *client* dan *server*, bahwa respon yang diberikan oleh *server* dapat disimpan sementara (*cache*) hingga periode tertentu.
i) *Host*. *Header* ini berisi nama domain dari URI. Saat *request* dilakukan pada *http://www.example.com/page.html*, maka *header* ini akan berisi

www.example.com:80.

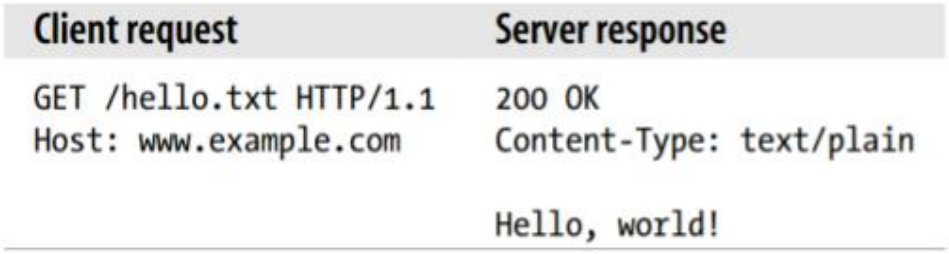
j) *Last-Modified*. *Header* ini berisi waktu terakhir *resource* tersebut mengalami perubahan.

k) *User-Agent*. *Header* ini menginformasikan jenis perangkat lunak yang digunakan saat melakukan *request*.

l) WWW-*Authenticate*. *Header* ini digunakan bersamaan dengan kode respon 401 ("*Unauthorized*"). Hal ini berarti *server* mengharapkan *client* mengulangi *request* tersebut dengan menyertakan akunnya beserta skema autentikasi yang digunakan.

I. *Uniform Resource Identifier* (URI)

URI atau yang lebih dikenal dengan URL (*Uniform Resource Locator*) merupakan frase yang terdiri atas sekumpulan karakter yang digunakan untuk mengidentifikasi suatu nama atau sumberdaya (*resource*). Identifikasi tersebut memungkinkan terjadinya interaksi berdasarkan representasi *resource* melalui jaringan menggunakan protokol tertentu (Berners *et al*. 2005). Misalkan seseorang memasukkan URL di peramban: *http://www.example.com/hello.txt*. Peramban dapat memanipulasi *resource* dengan menghubungi *server* lokasi tempat *resource* tersebut berada (*www.example.com*) dengan cara mengirimkan sebuah operator "*GET*" yang menuju ke arah *resource* /hello.txt pada *server* tersebut. Selanjutnya *server* akan memberikan respon sesuai dengan permintaan yang diberikan kepadanya. Catatan respon dari *client* yang mengakses *resource* dapat dilihat pada Gambar 5.



Gambar 5 Catatan Respon dari Pembacaan *Resource* (Richardson dan Ruby, 2007)

URI pada ROA harus didesain sedemikian rupa sehingga URI tersebut memiliki beberapa sifat berikut (Richardson dan Ruby, 2007).

a) URI harus deskriptif

   URI dan *resource* yang direpresentasikan harus memiliki hubungan secara intuitif. Berikut contoh URI yang baik:

   - *http://www.example.com/wiki/Jellyfish;*
   - *http://www.example.com/bugs/by-state/open;*
   - *http://www.example.com/resource/sales/2004/Q4;*
   - dan *http://www.example.com/resource/sales/Q42004*.

b) URI harus merepresentasikan paling banyak sebuah *resource*

   URI harus didesain untuk merepresentasikan maksimal sebuah *resource*, tapi sebuah *resource* dapat direpresentasikan oleh lebih dari satu URI. Pengaksesan */resource/sales/2004/Q4* mungkin akan memiliki hasil yang sama dengan */resource/sales/Q42004*, namun keduanya bukanlah *resource* yang berbeda.

J. Web Service

web service adalah layanan yang tersedia di Internet. *web service* menggunakan format standar XML untuk pengiriman pesannya. *Web services* juga tidak terikat kepada bahasa pemrograman atau sistem operasi tertentu (Cerami, 2002).

Menurut Gottschalk, *et al* (2002) *web services* adalah antar muka yang mendeskripsikan koleksi yang dapat diakses dalam jaringan menggunakan format standar XML untuk pertukaran pesan. *web services* mengerjakan tugas yang spesifik. *web services* dideskripsikan menggunakan format standar notasi XML yang disebut *services description*.

Menurut Booth *et al*. (2004) *web service* adalah sistem perangkat lunak yang dirancang untuk mendukung interaksi yang bisa beroperasi *machine-to-machine* di atas jaringan. Web service mempunyai alat penghubung yang diuraikan di dalam format *machine-processable* (secara spesifik WSDL). Sistem lain saling berhubungan dengan *web service* di dalam cara yang ditentukan oleh deskripsinya yang menggunakan pesan SOAP, secara khas disampaikan menggunakan HTTP dengan XML *serialization*, bersama dengan standar lain yang terkait dengan *web*.

4. Agen dan Service

Untuk menjalankan fungsinya, *web service* memerlukan agen. Agen adalah bagian perangkat lunak atau perangkat keras yang mengirimkan dan menerima pesan. Agen dapat ditulis dengan berbagai bahasa

pemrograman dan dapat berganti-ganti bahasa pemrograman dengan fungsi yang sama (Suyanto, 2007).

5. Requests dan Providers

Tujuan *web service* adalah untuk menyediakan beberapa fungsi atas nama pemilik nya (seseorang atau organisasi) baik itu untuk keperluan bisnis atau perorangan. *Provider entity* adalah orang atau organisasi dengan agen yang akan menerapkan *web service*. Dalam implementasinya *provider entity* adalah *server* yang menyediakan *web service*. *Request entity* adalah seseorang atau organisasi yang ingin menggunakan *web service* yang disediakan oleh *provider entity*. Dalam implementasinya *request entity* adalah *client* yang menggunakan *web service*. Dalam banyak kasus, *request agent* adalah agen yang akan memulai untuk melakukan komunikasi dengan *provider agent*. Agar pertukaran pesan dapat berjalan sesuai dengan apa yang diharapkan, *request entity* dan *provider entity* harus sepakat untuk menggunakan semantik dan mekanisme yang sama dalam pertukaran pesan (Suyanto, 2007).

6. Deskripsi *Service*

Mekanisme pertukaran pesan antara *request entity* dan *provider entity* di dokumentasikan di *Web service Description* (WSD). *Web service Description* adalah sebuah spesifikasi dari antarmuka *web service* yang dapat diproses oleh mesin/aplikasi, WSD biasanya ditulis dalam WSDL

(*Web service Description Language*). Spesifikasi yang dimaksud berupa format pesan, tipe data, protokol transpor, dan format serialisasi yang digunakan antara *requester agent* dan *provider agent*. Hal ini juga menentukan terdapat satu atau lebih jaringan dari *provider agent* yang dapat digunakan. Selain itu juga dapat memberikan beberapa informasi tentang pola pertukaran pesan yang diharapkan (Suyanto, 2007).

7. Semantik *Web Service*

Semantik pada *web service* adalah harapan bersama tentang perilaku layanan, khususnya dalam menanggapi pesan yang dikirim ke tujuan. Akibatnya, ini adalah "kontrak" antara *requester entity* dan *provider entity* tentang tujuan dan konsekuensi dari interaksi. Meskipun kontrak ini merupakan keseluruhan perjanjian antara *requester entity* dan *provider entity* tentang bagaimana dan mengapa masing-masing agen akan berinteraksi, itu belum tentu tertulis atau eksplisit dinegosiasikan. Ini mungkin eksplisit atau implisit, lisan atau tertulis, mesin *processable* atau *human-orientation*, dan mungkin suatu perjanjian hukum atau kesepakatan informal (*non*-hukum) (Suyanto, 2007).

8. Gambaran dari *Web Service*

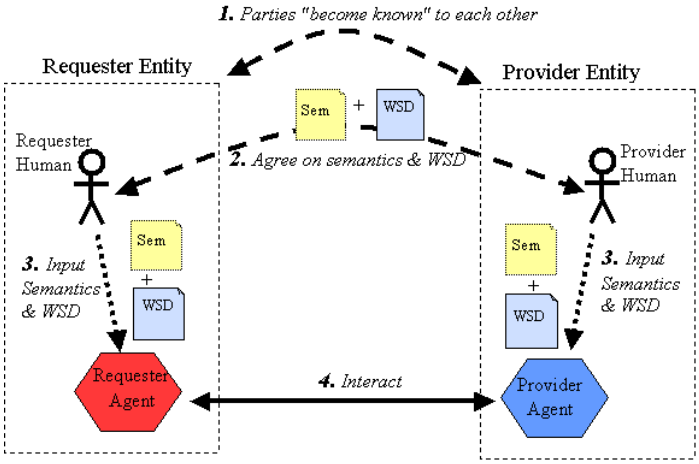
Ada banyak cara yang dapat membuktikan bahwa *requester entity* terlibat dan menggunakan *web service*. Secara umum, langkah-langkah berikut merupakan urutan yang diperlukan melakukan pertukaran pesan, seperti yang diilustrasikan pada Gambar 6.

a) *Requester* dan *provider entity* mengenal satu sama lain (atau setidaknya salah satu dari mereka mengetahui yang lain, biasanya *requester* mengetahui *provider*).

b) *Requester* dan *provider entity* setuju pada deskripsi layanan dan semantik yang akan mengatur interaksi antara *requester* dan *provider agent*.

c) Deskripsi layanan dan semantik yang tadi telah disetujui direalisasikan atau diterapkan oleh *requester* dan *provider agent*.

d) *Requester* dan *provider agent* melakukan pertukaran pesan, sehingga melakukan beberapa tugas atas nama *requester* dan *provider entity* (Contoh: pertukaran pesan dengan *provider agent* merupakan wujud nyata dari berinteraksi dengan layanan *web provider entity*).

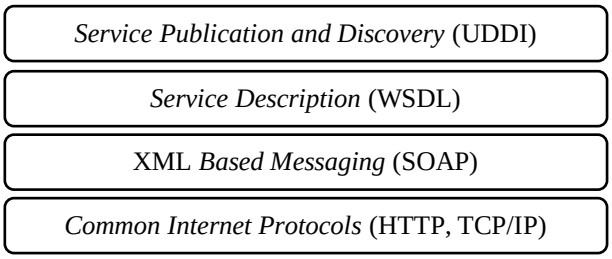


Gambar 6 Urutan Pertukaran Pesan pada *Web Service* (Suyanto, 2007)

9. Teknologi *Web Service*

Arsitektur *web service* melibatkan teknologi yang berlapis dan saling terkait. Ada banyak cara untuk memvisualisasikan teknologi ini, seperti halnya ada banyak cara untuk membangun dan menggunakan *web service*. Gambar 7 merupakan blok bangunan *web service* yang mana menyediakan fasilitas komunikasi jarak jauh antara dua aplikasi yang merupakan layer arsitektur *web service* (Tidwell, 2001).

a) Layer 1: protokol internet standar yang digunakan sebagai sarana transportasi adalah HTTP dan TCP/IP.
b) Layer 2: *Simple Object Access Protocol* (SOAP) berbasiskan XML dan digunakan untuk pertukaran informasi antar sekelompok layanan.
c) Layer 3: *Web service Definition Language* (WSDL) digunakan untuk mendeskripsikan atribut layanan.
d) Layer 4: *Universal Description, Discovery and Integration*, yang mana merupakan direktori pusat untuk deskripsi.



Gambar 7 Blok Bangunan *Web Service* (Tidwell, 2001)

10. XML

XML merupakan dasar terbentuknya *web service* yang digunakan untuk

mendeskripsikan data. Pada level paling detail *web service* secara keseluruhan dibentuk di atas XML. Fungsi utama dari XML adalah komunikasi antar aplikasi, integrasi data, dan komunikasi aplikasi eksternal dengan partner luaran. Dengan standarisasi XML, aplikasi-aplikasi yang berbeda dapat dengan mudah berkomunikasi antar satu dengan yang lain (Deviana, 2007).

11. SOAP (Simple Object Access Protocol)

SOAP merupakan protokol untuk pertukaran informasi dengan desentralisasi dan terdistribusi. SOAP merupakan gabungan antara HTTP dengan XML karena SOAP umumnya menggunakan *protocol* HTTP sebagai sarana transport datanya dan data akan dipertukarkan ditulis dalam format XML. Karena SOAP menggunakan HTTP dan XML maka SOAP memungkinkan pihak-pihak yang mempunyai *platform*, sistem operasi dan perangkat lunak yang berbeda dapat saling mempertukarkan datanya. SOAP mengatur bagaimana *request* dan respon dari suatu *web service* bekerja (Deviana, 2007).

12. WSDL (Web Services Description Language)

WSDL merupakan sebuah bahasa berbasis XML yang digunakan untuk mendefinisikan *web service* dan menggambarkan bagaimana cara untuk mengakses *web service* tersebut. Fungsi utama WSDL dalam *web service* adalah untuk mengotomasi mekanisme komunikasi *business-to-business* dalam *web service* melalui protokol internet.

WSDL merupakan representasi kontrak antara *requestor* dan *provider*-nya. Secara teknis merupakan representasi kontrak antara kode klien dan kode di *server*. Dengan menggunakan WSDL klien dapat memanfaatkan fungsi-fungsi publik yang disediakan oleh *server* (Deviana, 2007).

K. Python

Python adalah bahasa pemrograman yang interpretatif dan berorientasi objek yang cocok untuk berbagai tujuan. Python adalah bahasa pemrograman dengan sintaksis kode yang jelas, mudah dipahami, memiliki struktur data yang kuat, dan merupakan bahasa pemrograman yang dinamis. Python merupakan bahasa pemrograman yang interaktif, baik digunakan untuk membuat aplikasi yang dapat berdiri sendiri atau *stand-alone*, untuk aplikasi besar atau hanya sebagai *extensi* aplikasi yang sudah ada. Tidak perlu khawatir akan *operating system* yang digunakan, karena bahasa pemrograman ini dapat berjalan di berbagai *operating system* antara lain UNIX, Macintosh dan DOS Machine. Bahasa pemrograman Python memiliki dokumentasi yang lengkap (Swaroop, 2005).

13. Dibalik nama "Python"

Python dikembangkan oleh Guido van Rossum pada tahun 1990 di CWI, Amsterdam sebagai kelanjutan dari bahasa pemrograman ABC. Versi terakhir yang dikeluarkan CWI adalah 1.2. Nama Python diberikan oleh Guido setelah ia melihat acara di TV BBC *show* yang berjudul "*Monty Python's Flying Circus*". Guido tidak terlalu suka dengan ular yang

membunuh hewan lain untuk makan dengan melilit tubuh hewan tersebut dan menghancurkannya (Swaroop, 2005).

14. Fitur Python

Menurut Swaroop (2005), berikut adalah fitur-fitur dari bahasa pemrograman Python.

a) *Simple*

Python merupakan bahasa pemrograman yang *simple* dan minimalis. Membaca dan menulis dengan bahasa pemrograman Python hampir mirip dengan membaca dan menulis dengan menggunakan bahasa inggris. Ini adalah salah satu kekuatan Python yang membuat Python mudah dipahami dan diimplementasikan. Sehingga pengembang aplikasi dapat berkonsentrasi pada solusi dari masalah yang dihadapi daripada memahami bahasa Python itu sendiri.

b) Mudah dipahami

Seperti yang sudah disebutkan sebelumnya, Python merupakan bahasa pemrograman yang mudah dipelajari. Python memiliki sintaks yang sederhana.

c) Gratis dan *Open source*

Python merupakan salah satu contoh dari FLOSS (*Free/Libre and Open source Software*). Sederhananya, pengembang dapat dengan bebas mendistribusikan salinan dari *software* ini, membaca *source code*-nya, mengubah isi dari *source code* dan menggunakannya untuk *software* yang baru atau yang sudah ada. Latar belakang

FLOSS adalah komunitas yang saling berbagi pengetahuan. Oleh karena itu Python merupakan bahasa pemrograman yang baik, karena terus dikembangkan oleh komunitas, baik yang menggunakannya atau hanya ingin mencobanya.

Python termasuk kedalam kelompok bahasa pemrograman tingkat tinggi. Dengan menggunakan Python, pengembang tidak perlu repot-repot mengetahui cara mengelola memori untuk program yang dibuat seperti yang dilakukan oleh pemrograman tingkat rendah.

d) *PorTabel*

Karena Python bersifat *open source*, Python telah dibuat untuk dapat bekerja (*porting*) di berbagai *platform*. Semua program Python yang dibuat dapat bekerja pada setiap *platform* tanpa harus melakukan perubahan. Pengembang dapat menjalankan Python pada *platform* GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS / 2, Amiga, AROS, AS / 400, BeOS, OS / 390, z / OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE dan PocketPC. Bahkan pengembang dapat menggunakan *platform* seperti Kivy untuk membuat game untuk komputer, iPhone, iPad, dan Android.

e) *Interpreted*

Perlu sedikit penjelasan untuk memahami apa yang dimaksud dengan fitur ini. Apabila sebuah aplikasi yang ditulis dengan bahasa pemrograman dengan kategori *compiler* seperti C atau C++, *source code* akan dikompilasi menjadi *object code* (bahasa assembly atau

biner 0 dan 1) dengan menggunakan *compiler*. Ketika program dijalankan, *file link*er/*software* yang dapat di eksekusi dari *hardisk* ke *memory* lalu dijalankan.

Python tidak perlu di *compile* ke bahasa assembly atau biner. Pengembang langsung dapat menjalankan *software* langsung dari *source code*. Dibalik kemampuan Python dapat langsung menjalankan *source code*, Python mampu mengubah *source code* menjadi bentuk peralihan yang sering disebut *bytecode* dan kemudian menerjemahkan *bytecode* ke bahasa biner yang kemudian dijalankan. Hal ini membuat *software* yang dibuat menjadi lebih *portabel*, pengembang dapat menyalin *source code* dan dapat dijalankan langsung di komputer atau *server* yang lain.

f) *Object Oriented*

Python mendukung pemrograman berbasis prosedural (*procedure-oriented*) dan pemrograman berorientasi objek (*object-oriented*). Di dalam pemrograman *procedure-oriented* program dibangun dengan likup prosedur atau dengan kata lain fungsi yang dibuat hannyalah sebagai potongan program yang dapat digunakan kembali. Didalam pemrograman *object-oriented* program dibangun berdasarkan *object* yang menggabungkan data dan fungsi.

g) *Extensible*

Ekstensi dan modul-modul dapat secara mudah ditulis dalam bahasa C , C++ (atau java untuk Python atau .NET untuk IronPython).

i) *Embeddable*

Dapat dimasukkan ke dalam aplikasi/*software* lain sebagai antar muka skrip.

j) *Extensive Library*

Standar *library* Python sangatlah besar. Hal ini tentunya dapat membantu pengembang dalam melakukan berbagai hal antara yang melibatkan *regular expressions*, *documentation generation*, *unit testing*, *threading*, *database*s, *web browsers*, CGI, FTP, email, XML, XML-RPC, HTML, WAV *files*, *cryptography*, GUI (*graphical user interface*s), dan sistem atau aplikasi lainnya. Semua ini selalu tersedia setiap pengembang menginstal Python.

15. DJANGO

a. Pengertian Django

Berdasarkan *official web* Django (*https://www.djangoproject.com/*, 2017), Django adalah sebuah *web framework* yang *open source* dan berbasis Python. *Web framework* adalah sebuah kerangka kerja yang dapat mempermudah pengembang *software* khususnya *web*site. Django menggunakan konsep *Model Template and View* (MTV). *Model* adalah layer yang digunakan untuk berinteraksi dengan *database*. *Template* adalah layar yang digunakan untuk menangani masalah tampilan seperti XML, HTML dan lainnya. Sedangkan *view* adalah layer yang menghubungkan layer *model* dan *template* yang di dalamnya berisikan logika pengolah data dari model dan menampilkannya di *template*.

Sejak penyebaran pola MVC ke dalam pengembangan *web*, Python telah menyediakannya beberapa *web framework*, seperti Django, TurboGears, dan Zope. Django memiliki kelebihan di antara *web framework* lainnya yaitu (Hourieh, 2008).

a) *Tight Integration between Components*

   Pertama-tama, Django menyediakan seperangkat komponen yang terintegrasi. Semua komponen telah dikembangkan oleh tim Django sendiri. Django awalnya dikembangkan sebagai *framework in-house* untuk mengelola serangkaian berita yang berorientasi pada situs *web*. Kemudian kodenya dirilis di Internet dan tim Django melanjutkannya pengembangan menggunakan model *Open Source*. Oleh sebab itu, komponen Django dirancang untuk *integration*, *reusability* dan *speed* sejak awal.

b) *Object-Relational Mapper* (ORM, O/RM, and O/R *mapping tool*)

   Komponen *database* Django, *Object-Relational Mapper* (ORM), menyediakan jembatan antara model data dan *database engine*. Ini mendukung satu set besar dalam sistem *database*, dan beralih dari satu mesin ke mesin lainnya dalam permasalahan perubahan sebuah konfigurasi *file*. Hal ini memberi fleksibilitas besar bagi pengembang jika menginginkan keputusan dari satu mesin *database* ke *database* lainnya.

c) *Clean* URL *Design*

   Sistem URL di Django sangat fleksibel dan *powerful*. Ini memungkinkan dalam mendefinisikan pola untuk URL dalam

aplikasi dan menentukan fungsi Python untuk menangani setiap pola. Hal ini memungkinkan pengembang membuat URL yang ramah pengguna dan mesin pencari.

d) *Authomatic Admin Interface*

Django hadir dengan antarmuka administrasi yang siap digunakan. Antarmuka ini membuat pengelolaan data aplikasi dengan mudah. Hal ini juga sangat fleksibel dan dapat disesuaikan.

e) *Advanced Development Environment*

Selain itu, Django menyediakan lingkungan pengembangan yang sangat bagus. Django hadir dengan *web server* yang ringan untuk pengembangan dan pengujian. Saat *mode debugging* diaktifkan, Django memberikan pesan kesalahan yang sangat teliti dan terperinci dengan lebih banyak lagi informasi *debug*. Semua ini membuat proses isolasi dan memperbaiki *bug* dengan sangat mudah.

f) Mendukung Multibahasa

Django mendukung situs *web* multi bahasa melalui *built-in* internasionalisasi sistem. Ini bisa sangat berharga bagi mereka yang bekerja di situs *web* dengan lebih dari satu bahasa. Sistem membuat penerjemahan antarmuka menjadi tugas yang sangat sederhana.

Fitur standar yang diharapkan dari *framework* semuanya tersedia di Django, seperti (Hourieh, 2008):

- mesin *template* dan penyaringan teks dengan sintaks sederhana namun bisa diperluas;
- *form* generasi terbaru dan API validasi;
- sistem otentikasi yang dapat diperluas;
- sistem *caching* untuk mempercepat kinerja aplikasi;
- dan *feed framework* untuk menghasilkan RSS *feeds*.

Meskipun Django tidak menyediakan perpustakaan JavaScript untuk mempermudah kerja sama dengan Ajax tetapi, Django menyediakan seperangkat komponen yang terintegrasi dan matang, dengan Dokumentasi yang bagus, di *http://www.djangoproject.com/documentation/*, berkat komunitas pengembang dan penggunanya yang besar (Hourieh, 2008).

b. Sejarah Django

Django memulai sebagai proyek internal di surat kabar *Lawrence Journal-World* pada tahun 2003. Tim pengembangan *web* di sana sering harus mengimplementasikan fitur baru atau bahkan seluruh aplikasi dalam beberapa jam. Karena itu, Django diciptakan untuk memenuhi *deadline* dari situs jurnalisme, sementara pada saat bersamaan harus menjaga proses perkembangan agar bersih dan mudah dirawat (Hourieh, 2008).

Pada tahun 2005, Django menjadi cukup dewasa untuk menangani beberapa situs lalu lintas tinggi dan para pengembang memutuskan

untuk melepaskannya publik sebagai proyek *Open Source*. Proyek ini diambil dari nama Django Reinhardt. Sekarang Django adalah proyek *Open Source*, ia telah mengumpulkan pengembang dan pengguna dari seluruh dunia. Perbaikan bug dan fitur baru diperkenalkan setiap hari, sementara tim pengembang asli selalu mengawasi keseluruhan proses untuk memastikannya bahwa Django tetap menjadi *web framework* untuk membangun dengan *clean,* dapat dipertahankan dan dapat digunakan kembali (Hourieh, 2008).

c. Keamanan Django

Keamanan adalah sebuah topik dari sangat penting dalam pengembangan aplikasi jaringan dan Django menyediakan banyak alat perlindungan dan mekanisme, diantaranya (Dokumentasi Django, 2017).

a) Perlindungan *Cross Site Scripting* (XSS)

Serangan XSS memungkinkan pengguna menyuntikkan skrip sisi klien ke *browser* pengguna lain. Hal ini biasanya dicapai dengan menyimpan skrip berbahaya di *database* yang akan diambil dan ditampilkan ke pengguna lain, atau dengan membuat pengguna mengeklik tautan yang akan menyebabkan JavaScript penyerang dieksekusi oleh *browser* pengguna. Namun, serangan XSS dapat berasal dari sumber data yang tidak

terpercaya, seperti *cookies* atau layanan Web, kapan pun data tidak cukup disterilkan sebelum termasuk dalam halaman.

Menggunakan template Django melindungi proyek *web* terhadap mayoritas serangan XSS. Namun, penting untuk memahami perlindungan apa yang diberikannya dan keterbatasannya.

Django *template* melarikan diri karakter tertentu yang sangat berbahaya untuk HTML. Meskipun ini melindungi pengguna dari masukan paling jahat, namun hal itu tidak sepenuhnya mudah dilakukan.

b) Perlindungan *Cross Site Request Forgery (CSRF)*

Serangan CSRF memungkinkan pengguna jahat melakukan tindakan dengan menggunakan kredensial pengguna lain tanpa sepengetahuan atau persetujuan pengguna.

Django memiliki perlindungan built-in terhadap sebagian besar jenis serangan CSRF, yang memungkinkan dalam mengaktifkan dan menggunakannya sesuai kebutuhan. Namun, seperti teknik mitigasi lainnya, ada keterbatasan. Misalnya, adalah mungkin untuk menonaktifkan modul CSRF secara global atau untuk pandangan tertentu. Ada batasan lain jika situs *web* memiliki sub domain yang berada di luar kendali situs.

c) Perlindungan SQL *Injection*

Injeksi SQL adalah jenis serangan dimana pengguna jahat dapat mengeksekusi kode SQL sewenang-wenang pada *database*. Hal ini dapat mengakibatkan catatan dihapus atau kebocoran data.

Dengan menggunakan *querysets* Django, SQL yang dihasilkan akan benar-benar lolos dari *driver database* yang mendasarinya. Namun, Django juga memberi pengembang kekuatan untuk menulis *query* mentah atau mengeksekusi *custom SQL*. Kemampuan ini harus digunakan dengan hemat dan harus selalu berhati-hati untuk benar-benar lolos dari parameter yang dapat dikendalikan pengguna.

d) Perlindungan *Clickjacking*

Clickjacking adalah jenis serangan di mana situs berbahaya membungkus situs lain dalam bingkai. Serangan ini dapat mengakibatkan pengguna yang tidak curiga ditipu untuk melakukan tindakan yang tidak diinginkan di situs target.

Django berisi perlindungan *clickjacking* dalam bentuk *middleware X-Frame-Options* yang di *browser* pendukung dapat mencegah sebuah situs dirender dalam bingkai. Hal ini dimungkinkan untuk menonaktifkan perlindungan secara per tampilan atau untuk mengkonfigurasi nilai *header* yang tepat yang dikirim.

e) SSL/HTTPS

SSL/HTTPS selalu lebih baik untuk keamanan untuk menyebarkan situs *web* di belakang HTTPS. Tanpa ini, ada

kemungkinan pengguna jaringan berbahaya mengendus kredensial otentikasi atau informasi lain yang ditransfer antara klien dan *server*, dan dalam beberapa kasus - penyerang jaringan aktif - untuk mengubah data yang dikirim ke kedua arah.

f) *Host header validation*

Django menggunakan *header Host* yang disediakan oleh klien untuk membuat URL dalam kasus tertentu. Meskipun nilai-nilai ini disterilkan untuk mencegah serangan *Cross Site Scripting*, nilai *Host* palsu dapat digunakan untuk Pemalsuan Permintaan Lintas Situs, serangan keracunan *cache*, dan keracunan tautan dalam email.

Karena konfigurasi *server web* yang tampaknya tidak aman rentan terhadap *header Host* palsu, Django memvalidasi header Host melawan pengaturan *ALLOWED_HOSTS* di *metode django.http.HttpRequest.get_host()*.

d. Django *Project*

Terdapat empat buah file berisikan konfigurasi *database* serta konfigurasi-konfigurasi lainnya yang memiliki relasi dengan aplikasi Django, diantaranya (Hourieh, 2008).

a) *__init__.py*

Proyek Django adalah paket Python, dan file ini diperlukan untuk memberi tahu Python bahwa folder itu akan diperlakukan sebagai sebuah paket.

Paket dengan terminologi Python adalah kumpulan modul, dan digunakan untuk mengelompokkan file serupa bersama-sama dan mencegah konflik penamaan.

b) *manage.py*

Ini adalah skrip utilitas lain yang digunakan untuk mengelola proyek. Nama lain dari file ini adalah versi *django-admin.py* proyek. Sebenarnya, keduanya *django-admin.py* dan *manage.py* berbagi kode *back-end* yang sama.

c) *settings.py*

Ini adalah file konfigurasi utama untuk proyek Django. Dalam file ini dapat menentukan berbagai pilihan, termasuk pengaturan *database*, bahasa situs, fitur Django yang harus diaktifkan, dan seterusnya. Berbagai bagian dari file ini akan dijelaskan saat kita membangun aplikasi kita selama bab-bab berikutnya, namun di bab ini, kita hanya akan melihat bagaimana cara memasukkan *setting database*.

d) *url.py*

Ini adalah file konfigurasi lainnya. File ini adalah pemetaan antara fungsi URL dan Python yang menanganinya. File ini adalah salah satu fitur canggih Django.

e. Django dan REST

Menggabungkan Django dengan REST adalah praktik umum untuk membuat situs web kaya data. Ada banyak aplikasi yang dapat digunakan kembali di komunitas Django untuk membantu dalam

menerapkannya prinsip-prinsip ketat REST saat membangun API. Dalam beberapa tahun terakhir dua yang paling populer adalah *django-tastypie* dan *django-rest-framework*. Keduanya memiliki dukungan untuk menciptakan sumber dari ORM dan data *non*-ORM, otentikasi *pluggable* dan izin, dan dukungan untuk berbagai metode serialisasi, termasuk JSON, XML, YAML, dan HTML.

f. Django OAuth *Toolkit*

Sebelum aplikasi dapat menggunakan *Server* Otorisasi untuk *login* pengguna, harus terlebih dahulu mendaftarkan aplikasi (juga dikenal sebagai Klien.) Setelah terdaftar, aplikasi Anda akan diberi akses ke API, tergantung persetujuan dari penggunanya. Arahkan *browser* ke *http://localhost:8000/o/applications/* dan tambahkan *instance application*. *client id* dan *client secret* secara otomatis akan di *generate*; berikut adalah informasinya (Django OAuth Toolkit, 2013).

a) *User*; pemilik aplikasi (misalnya pengembang, atau pengguna yang saat ini masuk).
b) *Redirect uris*; aplikasi harus didaftarkan setidaknya URL pengalihan sebelum menggunakan titik akhir otorisasi. *Authorization server* akan mengirimkan token akses ke klien hanya jika klien menentukan salah satu uri pengalihan yang telah diverifikasi.
c) *Client type*; nilai ini mempengaruhi tingkat keamanan di mana

beberapa komunikasi antara aplikasi klien dan *server* otorisasi dilakukan.

d) *Authorization grant type*; biasanya adalah *Authorization code*

e) *Name*; nama aplikasi klien di *server*, dan akan ditampilkan di halaman permintaan otorisasi, di mana pengguna dapat mengizinkan / menolak akses ke datanya.

g. *Framework*

Ralph E. Johsson, Ketua *UIUC patterns/Software Architecture Group* dan koordinator program proyek senior di *Department of Computer Science* pada *University of Illinois*, menyatakan bahwa *framework* adalah desain yang *reuseable* dan biasanya dinyatakan sebagai satu set abstraksi *kelas* yang mengatur bagaimana *kelas* saling terhubung. Perancangan pada *framework* dibuat sedemikian rupa sehingga sebagian atau seluruh *software* dapat digunakan kembali. Contohnya: seorang *designer* membuat suatu UI (*user interface*) yang hanya dapat digunakan pada bagian tertentu suatu aplikasi. Desain tersebut tidak dapat digunakan kembali oleh bagian lainnya dari aplikasi tersebut. Hal ini tentunya kurang efektif. Bila menggunakan suatu *framework,* contohnya MacApp atau *Macintosh Application Framework* maka dapat menciptakan desain untuk keseluruhan aplikasi sehingga meningkatkan efisiensi kerja.

Menurut Pressman (2005), *framework* adalah kerangka kode yang dapat disempurnakan dengan *clasess* yang spesifik atau dengan

fungsi yang telah dirancang untuk mengatasi masalah yang dihadapi.

Dapat disimpulkan bahwa *framework* biasanya bersifat *object oriented* dan merupakan suatu desain sistem yang dapat digunakan kembali. Tujuannya untuk mengurangi pembuatan kembali kode yang sama sehingga Programmer dapat lebih fokus mengerjakan bagian lainnya.

Keuntungan menggunakan *framework*:

- menggunakan kembali kode yang telah dibuat dan diuji sehingga meningkatkan keandalan dari aplikasi baru dan mengurangi *programming* dan usaha pengujian;
- dan *framework* membantu menciptakan latihan pemrograman yang lebih baik dan penggunaan pola desain yang sesuai serta alat pemrograman yang baru.

Kekurangan *framework*:

- membuat sebuah *framework* tidaklah mudah dan memakan waktu serta biaya;
- dan seiring dengan berjalannya waktu, *framework* yang membuat dapat menjadi lebih kompleks.

16. Django REST *Framework* (DRF)

Django REST *framework* adalah *toolkit* untuk membangun *web*site API yang *powerful* dan fleksibel. Berikut alasan menggunakan REST *framework* (Christie, 2017).

a) *Web browsable* API atau API yang mudah dan dapat dibaca *browser* adalah kelebihan yang dapat memudahkan pengembangan *software*.
b) Dapat menggunakan *authentication package* seperti OAuth 1.0 dan OAuth 2.0.
c) Serialisasi yang mendukung ORM dan sumber data NON ORM.
d) Sangat mudah dimodifikasi, jika aplikasi yang dikembangkan tidak memerlukan fitur yang kompleks atau algoritma tertentu, pengembang dapat menggunakan fungsi *view* yang sudah tersedia dan sangan mudah diimplementasikan.
e) Dokumentasi yang sangat luas dan didukung oleh komunitas dengan anggota yang sangat banyak.
f) Digunakan dan dipercaya oleh perusahaan yang diakui secara internasional seperti Mozilla, Red Hat, Heroku, dan Eventbrite.

a. Membangun REST API

Sudah lebih dari satu dekade sejak Roy Fielding, seorang ilmuwan komputer Amerika dan salah satu penulis utama spesifikasi HTTP, memperkenalkan *REpresentational State Transfer* (REST) sebagai gaya arsitektur. Selama bertahun-tahun, REST telah mendapatkan momentum berkat popularitasnya untuk membangun layanan *web*

(Elman dan Lavin, 2015).

Dengan membuat pengenal unik untuk memungkinkan *caching*, *layering*, dan *scalability*, REST API menggunakan HTTP yang ada (GET, POST, PUT, dan DELETE) untuk membuat, memperbarui, dan menghapus sumber daya baru. Istilah REST sering digunakan untuk menggambarkan URL apa pun yang mengembalikan JSON alih-alih HTML. Klien juga harus dapat menavigasi API dan negara transisi melalui penggunaan tautan dan metadata dalam respons sumber daya. Klien tidak boleh menganggap adanya sumber daya atau tindakan selain beberapa titik masuk tetap, seperti *root* API (Elman dan Lavin, 2015).

b. *Requests* dan *Response*

Request kelas REST *framework* memperluas *HttpRequest* standar, menambahkan dukungan untuk *parsing request* yang fleksibel dan otentikasi *request*. REST *framework* menyediakan otentikasi *request* yang fleksibel, yang memberikan kemampuan dalam: (a) menggunakan kebijakan otentikasi yang berbeda untuk berbagai bagian API, (b) mendukung penggunaan beberapa kebijakan otentikasi, (c) dan memberikan informasi pengguna dan token yang terkait dengan permintaan masuk (Christie, 2017).

Menurut Dokumentasi Django (2017) tidak seperti objek *HttpResponse* dasar, objek *TemplateResponse* menyimpan rincian konteks yang disediakan oleh tampilan untuk menghitung *respons*.

Hasil akhir dari *response* tidak dihitung sampai dibutuhkan, kemudian dalam proses *response*.

REST *framework* mendukung negosiasi konten HTTP dengan menyediakan kelas *response* yang memungkinkan untuk mengembalikan konten yang dapat diberikan ke dalam beberapa jenis konten, bergantung pada permintaan klien. Kelas *response* adalah subkelas dari *SimpleTemplateResponse* Django. Objek *response* yang di inisialisasi dengan data, yang seharusnya konsisten dengan Python *native*. REST *framework* kemudian menggunakan konten HTTP standar untuk menentukan bagaimana seharusnya memberikan konten *final response* (Christie, 2017).

c. *Views*

REST *framework* menyediakan kelas *APIView*, yang merupakan subkelas dari kelas *View Django*. Kelas *APIView* berbeda dengan kelas *View* reguler dengan cara berikut (Christie, 2017).

a) *Request* diteruskan ke metode penanganan yang telah diinisialisasi oleh *request* REST *framework*, bukan *HttpRequest* Django.
b) Metode penanganan dapat mengembalikan *response* REST *framework*, bukan *HttpResponse* Django. Tampilan akan mengatur konten dan me*manage* sesuai *response*.
c) Setiap pengecualian *APIException* akan ditangkap dan dimediasi menjadi tanggapan yang tepat.

d) Permintaan masuk akan di otentikasi dan mengizinkan yang sesuai dan/atau mengecek *throttle* akan dijalankan sebelum mengirimkan permintaan ke *metode handler*.

Menggunakan kelas *APIView* hampir sama dengan menggunakan kelas *View* biasa, seperti biasa, permintaan masuk dikirim ke *metode handler* yang sesuai seperti *.get()* atau *.post()*. Selain itu, sejumlah atribut dapat ditetapkan di kelas yang mengontrol berbagai aspek dari kebijakan API (Christie, 2017).

d. *Serializers*

Serializers memungkinkan data kompleks seperti contoh *querysets* dan *model* yang akan dikonversi ke tipe data asli Python yang kemudian dapat dengan mudah diterjemahkan ke dalam JSON, XML atau jenis konten lainnya. *Serializers* juga memberikan *deserialization*, memungkinkan data parsing dikonversi kembali menjadi tipe kompleks, setelah pertama memvalidasi data yang masuk (Christie, 2017).

REST *framework serializer* bekerja sangat mirip dengan kelas *form Django* dan *ModelForm*. Ini menyediakan kelas *Serializer* yang memberi cara generik dan canggih untuk mengendalikan keluaran tanggapan, serta kelas *ModelSerializer* yang menyediakan pintasan yang berguna untuk membuat *serializer* yang berhubungan dengan contoh *model* dan *query* (Christie, 2017).

e. *Authentication*

Otentikasi adalah mekanisme untuk mengaitkan permintaan masuk dengan satu set identifikasi kredensial, seperti permintaan pengguna berasal, atau tanda yang ditandatangani dengan. Kebijakan izin dan pelarangan kemudian dapat menggunakan kredensial tersebut untuk menentukan apakah permintaan tersebut diizinkan. REST *framework* menyediakan sejumlah skema otentikasi di luar kotak, dan juga memungkinkan Anda untuk menerapkan skema *custom*. Otentikasi selalu dijalankan pada awal tampilan, sebelum izin dan pemeriksaan pencekalan terjadi, dan sebelum kode lainnya diizinkan untuk melanjutkan (Christie, 2017).

f. *Permissions*

Bersama dengan otentikasi dan *throttling*, izin untuk menentukan apakah sebuah permintaan harus diberikan atau ditolak aksesnya. Pemeriksaan izin selalu dijalankan pada awal tampilan, sebelum kode lainnya diizinkan untuk melanjutkan. Pemeriksaan izin biasanya akan menggunakan informasi otentikasi dalam *request.user* dan *request.auth* untuk menentukan apakah permintaan masuk harus diizinkan. Izin digunakan untuk memberi atau menolak akses kelas pengguna yang berbeda ke berbagai bagian API. Gaya izin yang paling sederhana adalah mengizinkan akses ke pengguna yang di autentikasi, dan menolak akses ke pengguna yang tidak berkepentingan. Ini sesuai dengan kelas *IsAuthenticated* dalam

REST *framework* (Christie, 2017).

L. *Open Authorization* (OAuth)

OAuth (*Open Authorization*) adalah protokol otorisasi standar terbuka yang memungkinkan pengguna mengakses aplikasi tanpa perlu berbagi password mereka (Chiragsh, 2012). Pemilik aplikasi mengintegrasikan *credential* milik pengguna dengan teknologi otentikasi yang berasal dari penerbit *API* (*Application Programming Interface*). OAuth juga mengizinkan otorisasi *API* yang sudah terproteksi yang berasal dari desktop ataupun aplikasi *web* melalui metode sederhana dan standard. Mengatur lalu lintas data antar aplikasi dan digunakan saat penerbit *API* ingin mengetahui siapa yang terlibat dan berkomunikasi di dalam sistem (Developers, 2017).

OAuth 1.0 (juga dikenal sebagai RFC 5849), diterbitkan pada tanggal 4 Desember 2007, direvisi pada tanggal 24 Juni 2009, dan diselesaikan pada bulan April 2010 yang memberikan pengaruh penting pada perkembangan keamanan *web API* tanpa harus pengguna berbagi *username* dan *password* mereka. Adapun pencipta dan penggagas utama dari otentikasi OAuth berbasis *API* ini adalah *E. Hammer-Lahav, Ed*.

Pada bulan April 2009, *Twitter* merilis sebuah solusi otentikasi user yang mereka sebut "*Sign-in with Twitter*" dengan menggunakan protokol OAuth. Otentikasi OAuth yang diciptakan dan dimodifikasi sendiri menciptakan layanan khusus untuk *Twitter*. Pembaharuan yang diciptakan oleh *twitter* mengundang para komunitas keamanan *web* merilis *First Security Advisory*

di tahun 2009, yang diikuti dengan versi revisi dari protokol OAuth, OAuth Revision A pada 24 Juni 2009 (Hammer, 2009).

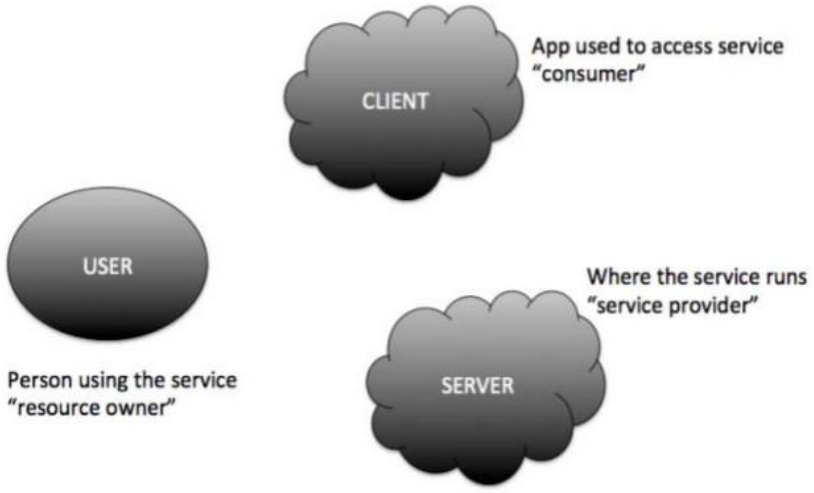
OAuth *2* adalah project lanjutan dari protokol OAuth yang asal mulanya diciptakan pada akhir tahun 2006. OAuth *2* lebih menekankan pada kemudahan *client* sebagai pemilik dan pengembang aplikasi dengan memberikan otorisasi khusus di berbagai aplikasi. OAuth berada dalam pengembangan *IETF* OAuth *WG* dan didasarkan pada usulan *WRAP* OAuth. *WRAP (Web Resource Authorization Protocol)* adalah profil OAuth yang memiliki sejumlah kemampuan penting yang tidak tersedia di versi OAuth sebelumnya. Spesifikasi terbaru dari OAuth disumbangkan kepada *IETS* OAuth *WG* dan merupakan dasar dari terciptanya OAuth versi 2 (Kaur dan Aggarwal, 2013).

Dengan banyaknya aplikasi *web* yang mengadopsi kolaborasi penggunaan jaringan sosial, pengembang aplikasi memiliki kesempatan untuk menghubungkan pengguna dengan aplikasi dimanapun mereka berada. Yang dapat meningkatkan efektivitas dan efisiensi terjadinya transaksi pengolahan data di dalam sistem tersebut. OAuth menyediakan kemampuan terhubung dengan sistem aplikasi secara aman, sehingga pengguna tidak perlu menyerahkan sandi akun pentingnya (Tapiador, 2012). Ada beberapa fungsionalitas menguntungkan yang tersedia pada OAuth meliputi (Brail *et al.*, 2012).

a) Mendapatkan akses ke grafik sosial media seperti teman yang berasal dari *Facebook*, orang-orang yang mengikuti (*following*) di *Twitter*, ataupun *list contact* yang berasal dari *Google Contact*.
b) Berbagi informasi mengenai kegiatan seorang pengunjung pada sebuah *web* aplikasi dengan menandai postingan mereka di *Facebook* ataupun *tweet* dari *Twitter* mereka.
c) Mengakses penggunaan *Google Docs* atau akun *Dropbox* untuk menyimpan data *online* mereka dengan berbagai file pilihan.

17. OAuth 1.0

OAuth *1.0* dilakukan oleh terlibatnya dari 3 peran berikut yakni : *client*, *server*, dan *resource owner*. Ketiga peran ini akan hadir dalam setiap alur kerja OAuth. Versi asli dari spesifikasi yang digunakan yang berbeda istilah untuk peran ini : konsumen (*client*), penyedia layanan (*server*), dan *user* (pemilik sumber daya). 3 (tiga) peran yang terlibat aktif ini disebut dengan *3-Legged*. Jika di contohkan dan di ilustrasikan perannya seperti berikut ini. Pada Gambar 8 menunjukkan keterlibatan 3 (tiga) peran aktif pada lalu lintas protokol OAuth 1.0 (Brail *et al.*, 2012).



Gambar 8 Tiga Peran Aktif pada *OAuth 1.0* (Brail *et al.*, 2012)

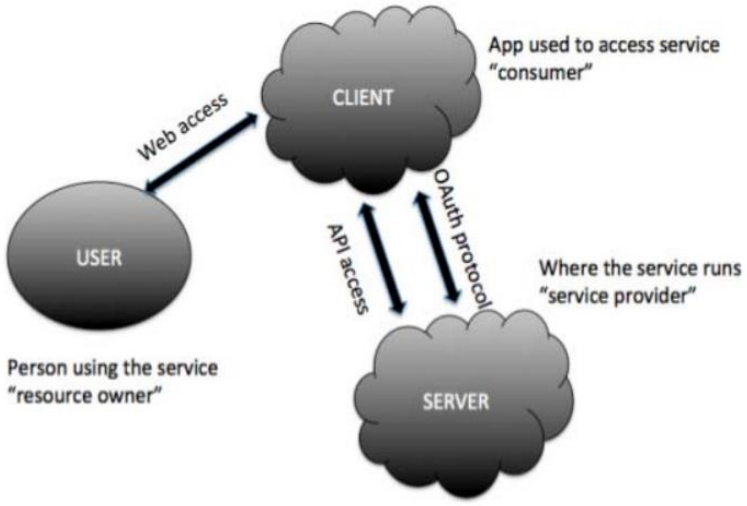
Bagaimana Bob menggunakan sumber daya layanannya yang ada di *bit.ly* melalui *account twitter* nya (Brail *et al.*, 2012).

a) Bob mengunjungi dan mengakses *bit.ly*, yang sudah menggunakan layanan dan tersedia di *twitter*. Bob sudah memiliki *account* baik itu untuk *bit.ly* ataupun *twitter*.
b) Pada sistem layanan *bit.ly* menggunakan OAuth credential sebagai proses awal otentikasi kepemilikan Bob. *Bit.ly* mengarahkan Bob untuk sementara ke *twitter.com* untuk proses login (*bit.ly* tanpa pernah sedikitpun mengetahui *account* Bob di *twitter.com*). Halaman otentikasi twitter menanyakan apakah aplikasi *bit.ly* dapat mengakses *twitter*. Pada Gambar 9 di bawah ini menunjukkan otorisasi *bit.ly* menggunakan *account twitter.com*.

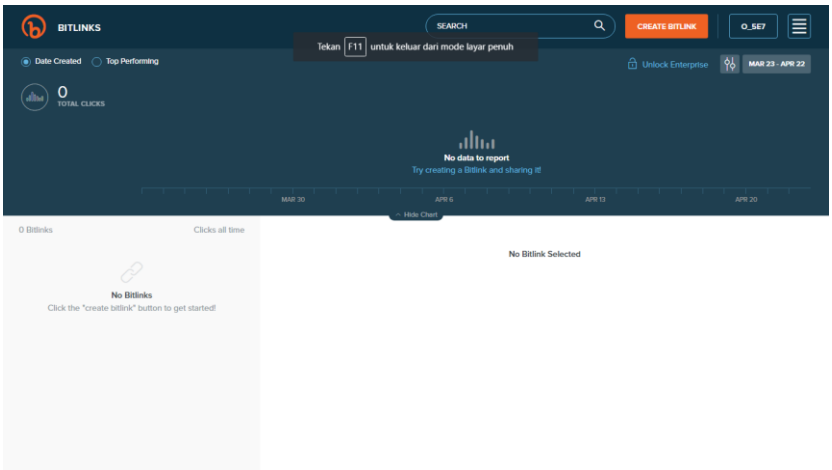
Gambar 9 Otorisasi Akses Layanan *Bit.Ly* Menggunakan *Account Twitter.com*

c) Jika proses login berhasil, *bit.ly* menggunakan *credential* OAuth (*token*) yang merupakan *valet key*, mengizinkan *bit.ly* menggunakan *twitter* Bob.

d) *Bit.ly* menyimpan *credential* penting Bob sepanjang Bob menggunakan *account* nya *bit.ly* mengizinkan Bob menggunakan layanan pada *bit.ly* dan hanya *bit.ly* mengakses *twitter.com*. Gambar 10 di bawah ini menjelaskan lalu lintas transaksi yang terjadi pada sistem otentikasi OAuth *1.0*. Pada Gambar 11 terlihat *interface* (antar muka) daripada *bit.ly* setelah sukses login menggunakan *credential* otentikasi OAuth *account twitter*.



Gambar 10 Lalu Lintas Transaksi *OAuth 1.0*



Gambar 11 Antar Muka Layanan Bit.Ly

18. OAuth 2.0

Terdapat 4 peran utama dalam mekanisme kerja OAuth 2 yakni (Hardt, 2012).

a) *Resource Server*

Resource server (sumber daya *server*) yang digunakan oleh pengguna yang memiliki *API* (*Application Programming Interface*) dan dilindungi oleh OAuth. *Resource server* merupakan penerbit *API* yang memegang dan memiliki kekuasaan pengaturan data seperti foto, video, kontak, atau kalender.

b) *Resource Owner*

Memosisikan diri sebagai pemilik sumber daya (*resource owner*), yang merupakan pemilik dari aplikasi. Pemilik sumber daya memiliki kemampuan untuk mengakses sumber daya *server* dengan aplikasi yang sudah tersedia.

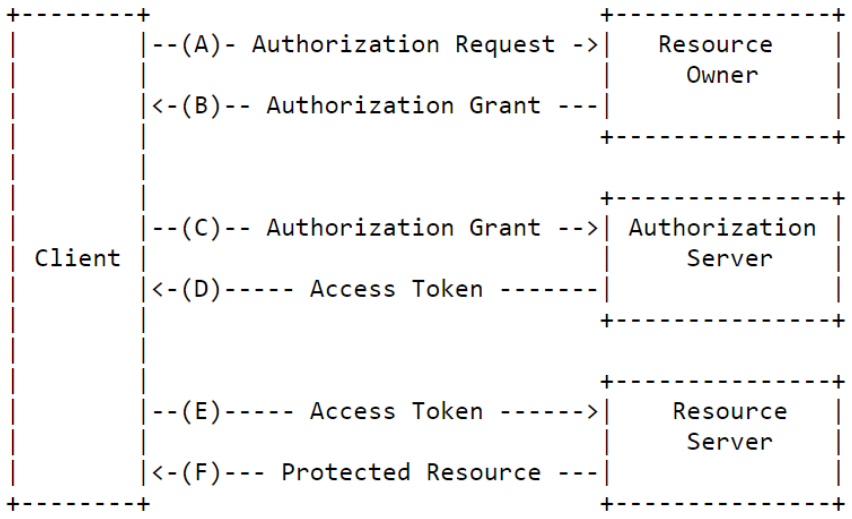
c) *Client*

Sebuah aplikasi yang membuat permintaan *API* pada *resource server* yang telah diproteksi untuk kepentingan pemilik *resource owner* dengan melakukan otorisasi.

d) *Authorization Server*

Authorization server (otorisasi *server*) mendapat persetujuan dari pemilik sumber daya (*resource owner*) dengan melakukan dan memberikan akses token kepada *client* untuk mengakses sumber daya yang diproteksi yang sudah tersedia pada *resource server*.

Mekanisme kerja OAuth 2 terbentuk dari peran aktif 4 (empat) bagian yang terdiri dari : *client*, *resource owner*, *authorization server*, *resource server* (Kaur dan Aggarwal, 2013). Gambar 12 menunjukkan mekanisme kerja OAuth *2* (Hardt, 2012), sedangkan pada Tabel 2 menunjukkan keterangan dari tugas dan fungsi 4 (empat) komponen OAuth.



Gambar 12 Mekanisme Kinerja *OAuth 2* (Hardt, 2012)

Tabel 2 Tugas dan Fungsi 4 (Empat) Komponen OAuth *2* (Hardt, 2012)

Kode	Keterangan
A	*Client* melakukan permintaan otorisasi dari *resource owner*. Permintaan otorisasi dapat dilakukan langsung menuju *resource owner*, atau jika tidak langsung melalui perantara *authorization server*.
B	*Client* mendapatkan persetujuan otorisasi yang merupakan credential mewakili otorisasi kepemilikan *client*. Pemberian otorisasi ini tergantung pada metode yang digunakan oleh *client* dan jenis yang didukung oleh *authorization server*.
C	*Client* melakukan permintaan akses token dengan otentikasi kepada *authorization server*, *client* mendapatkan penyajian hibah dan bentuk otorisasi dari *authorization server*.

Tabel 3 Tugas dan Fungsi 4 (Empat) Komponen OAuth *2* Lanjutan (Hardt, 2012)

Kode	Keterangan
D	Otorisasi *server* (*authorization server*) melakukan otentikasi kepada *client* dan memvalidasi pemberian otorisasi kepada *client*, jika sesuai dan berlaku, otorisasi *server* membagikan akses token.
E	*Client* melakukan permintaan sumber daya yang sudah diproteksi dari *resource server*, melakukan tindakan otentikasi dengan menghadirkan akses token.
F	*Resource server* memvalidasi akses token, jika valid dan sesuai, akan melayani permintaan *client* untuk menggunakan aplikasi yang sudah terlindungi.

19. Hibah Otorisasi (Authorization Grant)

Hibah otorisasi adalah mandat mewakili sumber daya pemilik otorisasi (*Resource Owner*) untuk mengakses sumber daya yang dilindungi yang digunakan oleh client (pengunjung) untuk mendapatkan akses token. Ada beberapa metode hibah otorisasi :*authorization code*, *implicit*, *resource owner*, *password credential* (Kaur dan Aggarwal, 2013).

g. *Authorization Code*

Kode otorisasi diperoleh dengan menggunakan otorisasi *server* sebagai perantara antara *client* dan *resource owner*. *client* melakukan permintaan otorisasi langsung dari *resource owner*, diarahkan menuju ke otorisasi *server* melalui *user-agent* (*web browser*) yang pada gilirannya mengarahkan pemilik sumber daya ke client dengan kode otorisasi (Hardt, 2012).

h. *Implicit*

Hibah otorisasi implisit adalah hibah otorisasi yang disederhanakan dan dioptimalkan untuk *client* yang diimplementasikan dalam

browser menggunakan bahasa *scripting JavaScript*. Dalam aliran otorisasi *implicit*, *client* diberikan langsung akses token sebagai hasil dari otorisasi pemilik sumber daya (*resource owner*) yang kemudian digunakan untuk mendapatkan akses token (Hardt, 2012).

a. *Resource Owner Password Credential*

Sumber daya pemilik *password credential* seperti (*username* dan *password*) dapat digunakan langsung sebagai hibah otorisasi untuk mendapatkan akses token. *Credential* yang digunakan berdasarkan kepercayaan tingkat tinggi antara pemilik sumber daya dan *client*. Jenis hibah ini membutuhkan akses *client* langsung ke pemilik sumber daya *credential* yang digunakan untuk satu permintaan dan ditukar dengan akses token (Hardt, 2012).

i. *Client Credential*

Kredensial klien adalah bentuk lain dari otentikasi klien yang digunakan sebagai hibah otorisasi terbatas pada sumber daya yang dilindungi di bawah pengawasan dan pengaturan klien. Atau ke sumber daya terlindungi yang sebelumnya telah di atur dengan *server* otorisasi. Kredensial klien digunakan sebagai otorisasi hibah ketika klien bertindak sebagai pemilik sumber daya (*resource owner*) atau meminta akses ke sumber daya yang dilindungi berdasarkan otorisasi yang sudah diatur dengan *server* otorisasi (*resource server*) (Hardt, 2012).

20. Access Token dan Refresh Token

j. *Access Token*

Access token adalah *credential* yang digunakan untuk mengakses sumber daya yang terlindungi. Akses token adalah kumpulan string yang mewakili suatu kuasa yang diberikan kepada *client*. Setiap token menyatakan batasan dan jangka waktu akses, yang diberikan oleh *resource owner* (pemilik sumber daya), yang berasal dari *resource server* (sumber *server*) dan *authorization server* (otorisasi *server*) (Parecki, 2012). Token dapat menunjukkan pengenal yang digunakan untuk mengambil informasi otorisasi atau mungkin data diri yang diverifikasi yaitu berupa tanda string yang terdiri dari beberapa rangkaian data (Kaur dan Aggarwal, 2013). Akses token dapat memiliki format yang berbeda baik dalam struktur dan metode pemanfaatan.

k. *Refresh Token*

Refresh token adalah token *credential* yang digunakan untuk mendapatkan token akses yang baru. Me*refresh* kembali akses token yang dikeluarkan otorisasi *server* (*authorization server*) dan digunakan untuk mendapatkan akses token terbaru ketika akses token tidak menjadi sah atau akses token memiliki ruang lingkup yang lebih pendek yang diterbitkan oleh *authorization server*. *Refresh token* merupakan string yang mewakili otorisasi yang diberikan kepada client oleh pemilik sumber daya. Tidak seperti

akses token, refresh token hanya digunakan untuk otorisasi *server* dengan tidak mengirimkan ke pemilik sumber daya (*resource owner*) (Parecki, 2012).

21. Jenis Client Profil

OAuth 2 mendefinisikan beberapa jenis client profil penting sebagai berikut.

l. *Server Side Web Application*

Adalah salah satu dari *client profil* OAuth *2* yang berjalan pada sisi *web server*. Aplikasi *web* yang telah diakses oleh pemilik sumber daya ataupun pengguna, melakukan permintaan *API* (*Application Programming Interface*) memanggil penggunaan satu bahasa program yang berjalan pada sisi *server* (Byod, 2012).

m. *Client Side User Agent Based Application*

Adalah jenis *client profil* yang berjalan pada *web browser* pengguna, dimana client mendapatkan hak akses kode aplikasi ataupun permintaan *API*. Aplikasi dapat berupa JavaScript yang terdapat di dalam sebuah halaman *web* bisa berupa *browse extension*, atau menggunakan teknologi *plugin* seperti Flash (Byod, 2012).

n. *Resource Owner Password Flow*

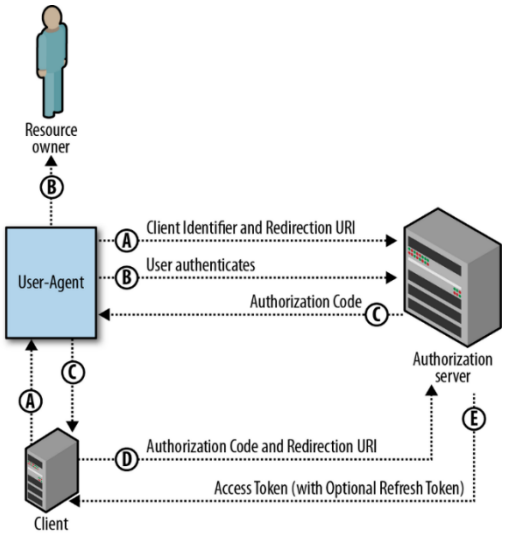
Adalah jenis *client profil* yang dimana *resource owner* (pemilik sumber daya) memiliki kepercayaan tingkat tinggi terhadap client.

Kelemahan dari *client profil* berikut adalah otorisasi *server* harus berhati - hati saat mengizinkan hibah ke dalam aliran sistem yang bekerja, mengingat kepercayaan penuh yang sudah diberikan kepada *client* (Byod, 2012).

o. *Workflow Client Profil*

Workflow client profil pada OAuth *2* berdasarkan pada mekanisme kerja yang berjalan pada sisi *client side* ataupun *server side*. Di bawah ini adalah model kerja yang otentikasi OAuth *2* yang berjalan pada sisi *server side*, *client side*, *resource owner password flow* dan *client credential* (Byod, 2012).

- *Server Side Web Application Flow*

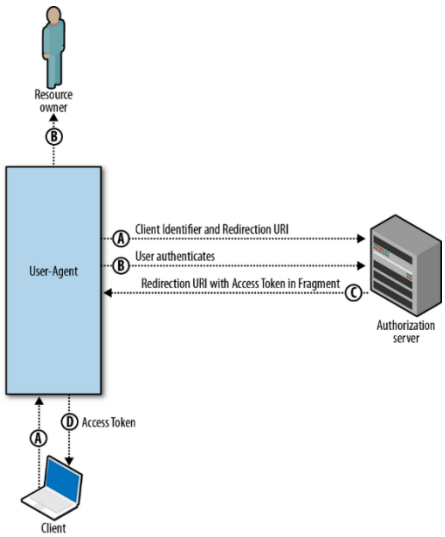


Gambar 13 *Server Side Web Application Flow* (Byod, 2012)

Tabel 4 *Server Side Web Application Flow* (Byod, 2012)

Kode	Keterangan
A	*Client* memulai aliran menuju aplikasi pemilik sumber daya (*resource owner*) lewat perantara user-agent langsung menuju titik akhir otorisasi.
B	Otorisasi *server* mengotentikasi *resource owner* dan menetapkan apakah pemilik sumber daya memberikan atau menolak permintaan akses *client*.
C	Dengan asumsi client mendapatkan kuasa melakukan akses, otorisasi *server* mengarahkan ulang user-agent kembali ke *client* menggunakan *redirect url* (*redirect url* termasuk kode otorisasi).
D	*Client* meminta akses token dari otorisasi *server* termasuk di dalamnya kode otorisasi (*code authorization*) yang sudah diterima sebelumnya. Ketika melakukan permintaan, *client* mengotentikasi dengan *server* otorisasi. *Client* termasuk url *direction* digunakan untuk memperoleh kode otorisasi untuk verifikasi.
E	Otorisasi *server* mengotentikasi *client*, memvalidasi kode otorisasi dan memastikan url redirect di terima sesuai dengan url yang digunakan untuk mengarahkan *client*, jika valid otorisasi *server* merespon kembali dengan akses token dan opsional *refresh token*.

- *Client Side Web Application Flow*

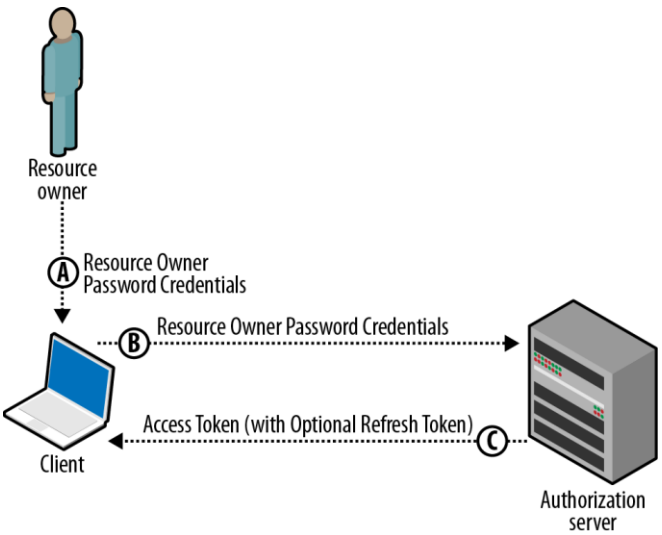


Gambar 14 *Client Side Web Application Flow* (Byod, 2012)

Tabel 5 Keterangan *Client Side Web Application Flow* (Byod, 2012)

Kode	Keterangan
A	*Client* memulai aliran dengan mengarahkan pemilik sumber daya (*resource owner*) lewat perantara *user-agent* menuju titik akhir otorisasi. Otorisasi *Server* akan mengirim kembali ke *user-agent* setelah akses diberikan atau ditolak.
B	Otorisasi *Server* mengotentikasi pemilik sumber daya, dan menetapkan apakah pemilik sumber daya memberikan atau menolak permintaan akses *client*.
C	Dengan asumsi pemilik sumber daya memberikan akses, otorisasi *server* mengarahkan ulang *user-agent* kembali ke *client* menggunakan *redirect url* sebelumnya. Pengalihan *url* termasuk akses token di dalam *fragmen url*.
D	*User-agent* mengikuti instruksi pengalihan dengan membuat permintaan sumber daya *client web hosting* dengan tetap mempertahankan informasi *fragment* lokal agar *user-agent* dapat mengeksekusi *scripts* yang disediakan *web-host* sumber daya *client* lokal yang mengekstrak akses token dan lolos ke *client*.

- *Resource Owner Password Flow*



Gambar 15 *Resource Owner Password Flow* (Byod, 2012)

Tabel 6 Keterangan *Resource Owner Password Flow* (Byod, 2012)

Kode	Keterangan
A	Pemilik sumber daya (*resource owner*) menyediakan *client* dengan nama pengguna (*username*) dan sandi (*password*)
B	*Client* meminta sebuah akses token dari *authorization server* (otorisasi *server*) dengan memasukkan *credential* yang telah diterima dari pemilik sumber daya (*resource owner*). Saat melakukan permintaan tersebut, *client* mengotentikasi dengan otorisasi *server* (*authorization server*).
C	*Server* otorisasi mengotentikasi *client* dan memvalidasi *credential* dari pemilik sumber daya, dan jika berlaku, mendapatkan sebuah akses token.

22. Proses Roles Authentication

p. *Web Server Application*

Aplikasi *web* yang berjalan pada sisi *server* adalah jenis paling umum yang dikembangkan oleh pemilik aplikasi (*resource owner*) yang didukung penuh pada OAuth *Server*. Aplikasi *web* pada bahasa pemrograman *server-side* seperti *php*, *asp.net*, *jsp* melakukan proses kerjanya pada sisi *server* yang tidak dipublikasikan untuk umum (Brail *et al.*, 2012). Proses yang terjadi dan berlaku saat seorang pengunjung melakukan kegiatan otentikasi dengan mengeklik tombol ”*Log In*” adalah pengunjung akan menerima rangkaian link pada alamat *browser* seperti pada Gambar 16 (Parecki, 2012).

```
https://oauth2server.com/auth?response_type=code&
  client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zyx
```

Gambar 16 Proses Otentifikasi *Web Server Application* (Parecki, 2012)

Beberapa parameter *query* yang terdapat pada rangkaian link alamat *browser* adalah sebagai berikut (Chiragsh, 2012).

- *client-id*

  Nilai yang diberikan saat mendaftarkan suatu aplikasi *web* yang dimiliki oleh *resource owner.*

- *redirect_uri*

  Lokasi kemana pengguna harus kembali setelah menyetujui akses untuk aplikasi.

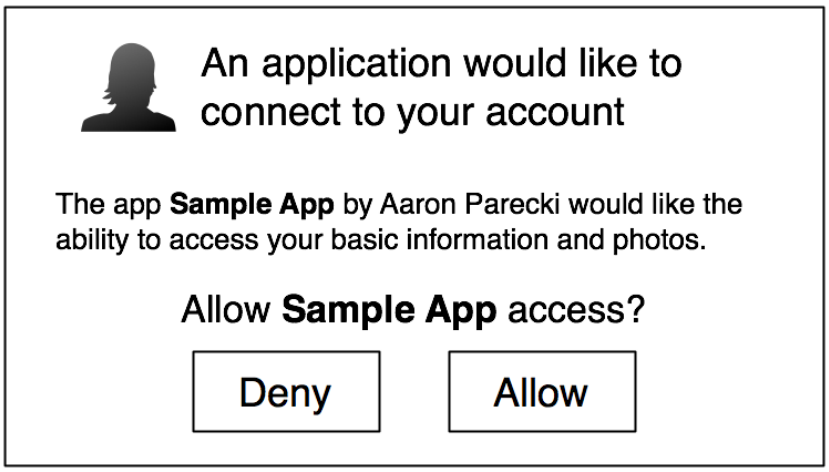
- *scope*

  Data aplikasi meminta akses menuju dan merujuk pada *web developer* penyedia OAuth. Jika berasal dari *google api* berarti merujuk ke halaman *auth google api* contoh: *https://www.googleapis.com/auth/tasks*.

- *response_type*

  Adalah kode *server side web application flow* yang mengidentifikasikan sebuah kode otorisasi yang akan dikembalikan lagi ke aplikasi setelah user menerima dan menyetujui permintaan otorisasi.

Pada sisi halaman *browser*, pengunjung di terbitkan suatu halaman pemberian izin hibah, untuk disetujui oleh pengunjung agar dapat dilakukan otorisasi oleh OAuth *Server* (*authorization server*). Gambar 17 memperlihatkan permintaan persetujuan dari pengunjung (Parecki, 2012).

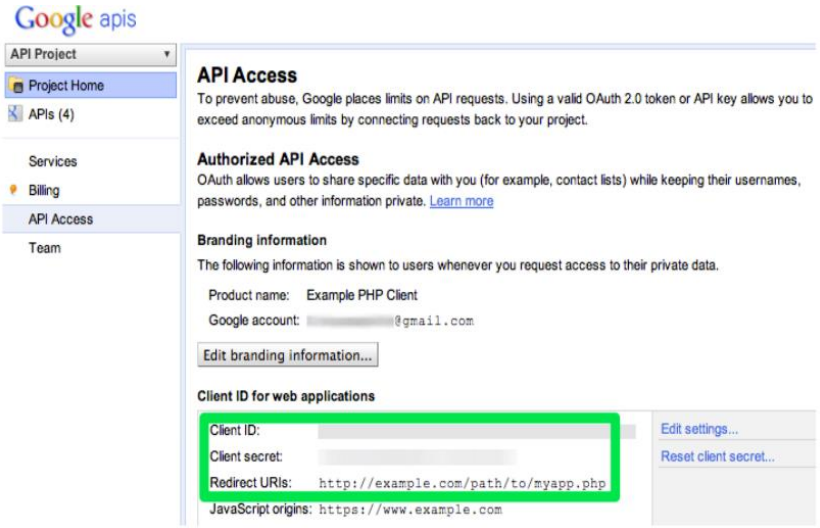


Gambar 17 Proses Otentifikasi *Web Server Application* 2 (Parecki, 2012)

Jika pengunjung melakukan proses tindakan ”*Allow*”, maka sistem yang bekerja akan mengarahkan kembali pada aplikasi *web* dengan sebuah *Auth Code*. *Auth Code* untuk suatu aplikasi *web* memiliki kode *unique* dan berbeda, yang setiap aplikasi jika didaftarkan pada salah satu penerbit OAuth seperti Google akan mendapatkan auth code tertentu. Pada Gambar 18 memperlihatkan sebuah halaman dengan *auth code*. Pada Gambar 19 adalah auth code yang didapat saat didaftarkan pada layanan OAuth yang tersedia pada *Google API* (Chiragsh, 2012).

```
https://oauth2client.com/cb?code=AUTH_CODE_HERE
```

Gambar 18 Proses Otentifikasi *Web Server Application* - 3 (Parecki, 2012)



Gambar 19 *Auth Code* dari *Google API*

authorization server (otorisasi *server*) kemudian mengubah auth code untuk sebuah akses token, dan alamat pada *browser* akan terlihat seperti pada Gambar 20 (Parecki, 2012).

```
POST https://api.oauth2server.com/token
    grant_type=authorization_code&
    code=AUTH_CODE_HERE&
    redirect_uri=REDIRECT_URI&
    client_id=CLIENT_ID&
    client_secret=CLIENT_SECRET
```

Gambar 20 Halaman *Auth Code* dari *Google API*

Beberapa parameter *query* yang terdapat pada rangkaian link alamat *browser* adalah sebagai berikut (Parecki, 2012).

- *client_id*

  Nilai *id* yang diberikan saat mendaftarkan suatu aplikasi.

- *client_secret*

Kepercayaan yang bersifat rahasia diberikan saat mendaftar aplikasi.

- *grant_type*

  Nilai suatu authorization_code, mengidentifikasikan penukaran sebuah kode otorisasi pada suatu akses token.

- *code*

  Kode otorisasi yang dikirimkan ke aplikasi.

- *redirect_uri*

  Lokasi yang telah terdaftar dan digunakan pada permintaan awal menuju otorisasi.

Kemudian otorisasi *server* menggantikannya dengan sebuah akses token seperti yang terlihat pada Gambar 21 dan jika terdapat error pada akses token akan terlihat seperti pada Gambar 22.

Gambar 21 Proses Otentikasi *Web Server Application* – 4 (Parecki, 2012)

Gambar 22 Proses Otentikasi *Web Server Application* – 5 (Parecki, 2012)

Adapun beberapa kondisi *error* yang terjadi pada alamat *url* (*uniform resource locator*) dan ditampilkan pada halaman *browser* seperti (Byod, 2012).

- *invalid_request*

  Adalah permintaan yang tidak *valid*, parameter mengalami kekurangan atau nilai parameter yang tidak sesuai atau tidak didukung.

- *unauthorized_client*

  Client tidak mempunyai kewenangan untuk meminta kode otorisasi menggunakan metode mendapatkan token.

- *unsupported_response_type*

  Otorisasi *server* tidak mendukung mendapatkan kode otorisasi menggunakan salah satu metode mendapatkan token.

- *invalid_scope*

  Cakupan yang diminta tidak valid, tidak diketahui dan tidak sesuai (cacat).

- *server_error*

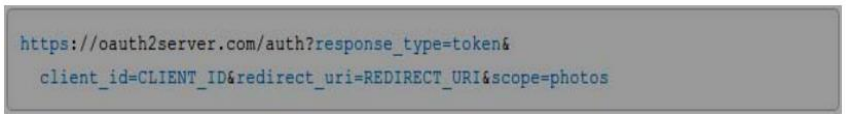
  Otorisasi *server* mengalami masalah sehingga tidak dapat mencegah masalah dan memenuhi permintaan penyesuaian token.

- *temporarily_unavailable*

  Keberadaan dan kondisi dari *authorization server* (otorisasi *server*) yang tidak dapat menangani permintaan karena *overloading* atau *maintenance server*.

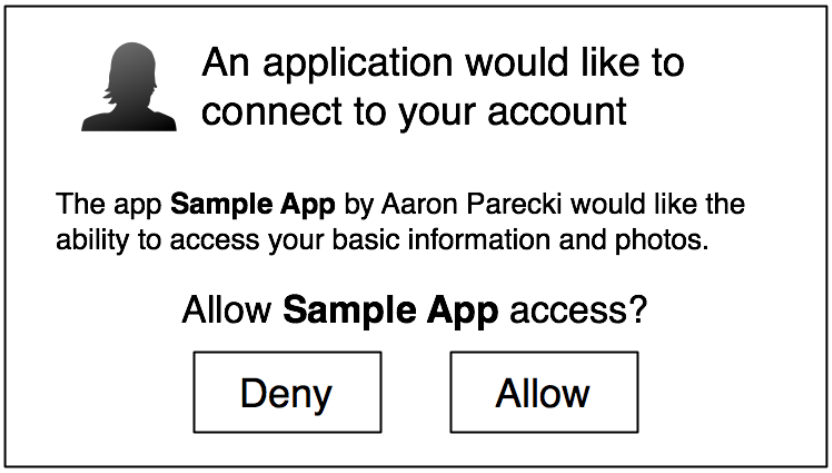
q. *Browser Based Application*

Aplikasi *browser* dijalankan sepenuhnya dalam *browser* setelah memuat suatu kode sumber dari suatu halaman *web*. Karena seluruh kode sumber tersedia untuk dan pada *browser*, sehingga kerahasiaan dari rahasia client, tidak digunakan pada aplikasi ini (Parecki, 2012). Proses yang tercipta apabila seorang pengunjung melakukan kegiatan otentikasi dengan mengeklik tombol "*Log In*" pada aplikasi *browser* adalah seperti yang ditunjukkan pada Gambar 23 di bawah ini.



Gambar 23 Proses Otentikasi *Browser Based Application* (Parecki, 2012)

Kemudian pengunjung akan diarahkan pada suatu halaman pemberian izin hibah, agar pengunjung dapat melakukan pemberian izin hibah yang akan di proses kembali oleh otorisasi *server* (*authorization server*). Gambar 24 berikut menunjukkan halaman *browser* persetujuan pemberian hibah oleh pengunjung (Parecki, 2012).



Gambar 24 Proses Otentikasi *Browser Based Application* - 2 (Parecki, 2012)

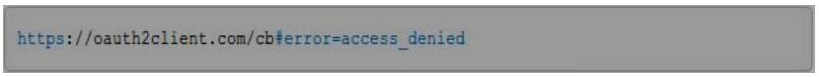
Jika pengunjung melakukan tindakan dengan menekan tombol ”*Allow*”, layar *browser* akan diarahkan menuju aplikasi dengan menghadirkan sebuah akses token. Pada Gambar 24 adalah proses yang terdapat alamat *HTTP* (*Hypertext Transfer Protocol*) saat pengunjung menekan tombol ”Allow” dengan menampilkan akses token.

```
https://oauth2client.com/cb#token=ACCESS_TOKEN
```

Gambar 25 Proses Otentikasi *Browser Based Application* - 3 (Parecki, 2012)

Selesai akses token diterima pada *browser* dengan memanggil kumpulan kode *JavaScript* yang bisa mengeluarkan kode token akses dari fragment (setelah tanda #) dan mulai membuat permintaan OAuth *API* (Parecki, 2012). Jika terdapat kesalahan, halaman akan

menerima kode kesalahan dalam fragment url yang terdapat pada Gambar 26 di bawah ini.



Gambar 26 Proses Otentikasi *Browser Based Application* - 4 (Parecki, 2012)

M. PostgreSQL

PostgreSQL adalah *open-source*, *client/server*, *relational database*. PostgreSQL menawarkan gabungan fitur unik dibandingkan dengan *database* komersial utama seperti Sybase, Oracle, dan DB2. Salah satu kelebihan utama PostgreSQL adalah *open source*. PostgreSQL tidak dimiliki oleh perusahaan tunggal manapun. PostgreSQL dikembangkan, dipelihara, dipatahkan, dan diperbaiki oleh sekelompok pengembang sukarelawan di seluruh dunia. Hal ini lah yang membuat PostgreSQL gratis dan tidak perlu membayar biaya perawatan apapun. PostgreSQL mampu berjalan di atas berbagai sistem operasi, termasuk Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OSX, Solaris, Tru64) dan juga Windows (Douglas., 2005).

23. Sejarah PostgreSQL

PostgreSQL, yang awalnya disebut Postgres, diciptakan di UCB oleh seorang profesor ilmu komputer bernama Michael Stonebraker, yang kemudian menjadi CTO dari Informix Corporation. Stonebraker memulai Postgres pada tahun 1986 sebagai proyek lanjutan untuk

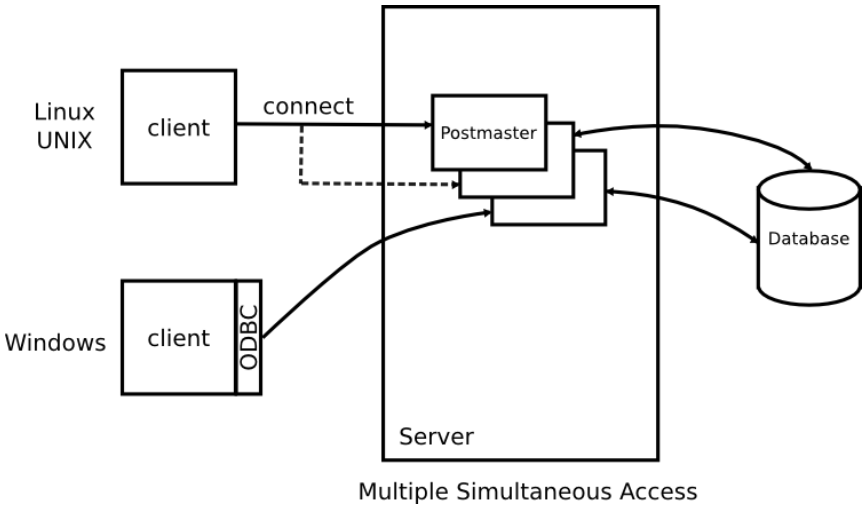
pendahulunya, Ingres, yang sekarang dimiliki oleh Computer Associates. Nama Postgres jadi diputar dari pendahulunya (seperti pada "setelah Ingres"). Ingres, yang dikembangkan dari tahun 1977 sampai 1985, telah menjadi latihan dalam menciptakan sistem *database* sesuai dengan teori RDBMS klasik. Postgres, yang dikembangkan antara tahun 1986-1994, adalah sebuah proyek yang dimaksudkan untuk memecahkan masalah baru dalam konsep *database* seperti eksplorasi teknologi "relasional objek". RDBMSs yang tersedia secara komersial. Saat ini, basis pengguna PostgreSQL lebih besar dari sebelumnya dan mencakup kelompok perusahaan besar yang cukup besar yang menggunakannya di lingkungan yang menuntut. Beberapa perusahaan seperti Afilias dan Fujitsu telah memberikan kontribusi signifikan terhadap pengembangan PostgreSQL (PostgreSQL Global Development Group, 2009).

24. Arsitektur PostgreSQL

Salah satu kekuatan PostgreSQL berasal dari arsitekturnya. Kesamaan dengan sistem *database* komersial, PostgreSQL dapat digunakan di lingkungan *client/server*. Ini memiliki banyak manfaat bagi pengguna dan pengembang. Inti dari instalasi PostgreSQL adalah proses *database server*. Ini berjalan di *server* tunggal. Aplikasi yang perlu mengakses data yang tersimpan dalam *database* diharuskan melakukannya melalui proses *database*. Program klien ini tidak dapat mengakses data secara

langsung, walaupun komputer tersebut berjalan pada komputer yang sama dengan proses *server* (Mathew dan Stones, 2005).

PostgreSQL terdiri atas proses (program) yang saling terkait, yaitu proses *server* untuk mengelola *database file*, menerima koneksi ke *database* dari *client application*, dan mengeksekusi permintaan dari *client* dan program *client user application* (*front-end*) yang berfungsi untuk melakukan operasi *database*. *Client* dan *server* dapat berada pada *host* yang berbeda. *Server* PostgreSQL dapat menangani koneksi bersamaan dari *client*. Untuk dapat mengoneksikan *client* secara bersamaan, PostgreSQL memulai dengan membuat proses (program) yang baru untuk masing-masing koneksi. Maka, *client* dan proses *server* yang baru dapat berkomunikasi tanpa adanya intervensi oleh proses PostgreSQL yang asli. Dengan demikian, proses dari master *server* masih tetap berjalan dan menunggu koneksi *client* (PostgreSQL Global Development Group, 2009).



Gambar 27 *Architecture* PostgreSQL (Mathew dan Stones, 2005)

Pada Gambar 27, dapat kita lihat beberapa *client* terhubung ke *server* melalui sebuah jaringan. Pada PostgreSQL, jaringan yang dibutuhkan seperti TCP/IP, *Local Area Network* (LAN) atau mungkin jaringan internet. Masing-masing *client* menghubungkan ke *database server* utama (*Postmaster*). *Postmaster* dapat membuat proses *server* baru yang spesifik untuk melayani permintaan *client* tersebut (Mathew dan Stones, 2005).

Eksekusi *query* pada PostgreSQL terdiri atas beberapa tahapan sampai dengan mendapatkan hasilnya (PostgreSQL Global Development Group, 2009).

a) Sebuah koneksi dari *application* program ke *server* dari PostgreSQL terlebih dahulu harus dibuat. *Application* program mengirimkan *query* dari pengguna ke *server* dan menunggu untuk menerima hasil yang dikirim kembali oleh *server* ke *application* program.
b) Tahap berikutnya adalah *parser* yang berfungsi memeriksa *query* yang dikirimkan oleh *application* program untuk mengoreksi *syntax* dan membuat sebuah *query tree*. *Syntax* diperiksa agar sistem dapat mengetahui apakah *syntax* yang dituliskan oleh pengguna tersebut benar.
c) Kemudian *rewrite System* mengambil *query tree* yang telah dibuat pada tahap *parser* dan memeriksa setiap aturan yang sesuai (disimpan dalam katalog sistem) untuk diterapkan pada *query tree*. *Rewrite System* melakukan transformasi yang dimasukkan ke dalam sebuah aturan *syntax* dan menulis ulang *query* dari pengguna ke

dalam sebuah *query* yang nantinya akan mengakses ke Tabel dasar.

d) *Planner* atau *optimizer* mengambil *query tree* yang telah ditulis ulang dan membuat *query plan* yang akan dimasukkan ke dalam eksekutor. *Planner* atau *optimizer* melakukannya dengan terlebih dahulu menciptakan semua *path* yang memungkinkan dan akhirnya mengarah ke hasil yang sama. Pertama adalah *sequential scan* dan yang lainnya menggunakan pengindeksan. Selanjutnya biaya untuk eksekusi dari masing-masing *path* diestimasikan dan dipilih *path* yang biayanya paling murah. *Path* yang termurah diperluas ke dalam *complete plan* yang kemudian eksekutor dapat menggunakannya.

e) Eksekutor akan mengulangi langkah di atas dengan kembali melewati *plan tree* dan mengambil baris-baris dengan cara yang diwakili oleh *plan*. Eksekutor memanfaatkan sistem penyimpanan ketika melakukan *scanning Tabel*, melakukan *sort* dan *join*, mengevaluasi kualifikasi dan akhirnya baris-baris yang sesuai dengan *query* akan diproses sebagai *output*.

25. Fitur PostgreSQL

PostgreSQL adalah salah satu *server database* terpopuler yang tersedia. Berikut adalah beberapa fitur yang terdapat dalam distribusi PostgreSQL standar (Mathew dan Stones, 2005).

a) *Object-relational*

Di PostgreSQL, setiap Tabel mendefinisikan sebuah kelas. PostgreSQL menerapkan warisan di antara Tabel (atau, jika Anda suka, di antara kelas). Fungsi dan operator bersifat polimorfik.

b) *Standards compliant*

Sintaks PostgreSQL mengimplementasikan sebagian besar standar SQL92 dan banyak fitur SQL99. Dimana perbedaan sintaks terjadi, mereka paling sering dikaitkan dengan fitur unik untuk PostgreSQL.

c) *Open source*

Tim pengembang internasional mengelola PostgreSQL. Anggota tim datang dan pergi, namun anggota inti telah meningkatkan kinerja dan rangkaian fitur PostgreSQL sejak setidaknya 1996. Salah satu keuntungan dari sifat open source PostgreSQL adalah bahwa bakat dan pengetahuan dapat direkrut seperlunya. Fakta bahwa tim internasional memastikan bahwa PostgreSQL adalah produk yang dapat digunakan secara produktif dalam bahasa alami, tidak hanya bahasa Inggris.

d) *Transaction processing*

PostgreSQL melindungi data dan mengordinir beberapa pengguna bersamaan melalui pemrosesan transaksi penuh. Model transaksi yang digunakan oleh PostgreSQL didasarkan pada *multi-version concurrency control* (MVCC). MVCC memberikan kinerja yang jauh lebih baik daripada yang Anda temukan dengan produk lain yang mengkoordinasikan banyak pengguna melalui penguncian Tabel, halaman, atau tingkat baris.

e) *Referential integrity*

PostgreSQL menerapkan integritas referensial lengkap dengan mendukung hubungan utama dan kunci utama serta *trigger*. Aturan bisnis dapat dinyatakan dalam *database* daripada mengandalkan alat eksternal.

f) *Multiple procedural languages*

Trigger dan prosedur lainnya dapat ditulis dalam beberapa bahasa prosedural. Kode sisi *server* paling sering ditulis dalam PL / pgSQL, bahasa prosedural yang mirip dengan PL / SQL Oracle. Anda juga bisa mengembangkan kode sisi *server* di Tcl, Perl, bahkan bash (open source linux / Unix shell).

g) *Multiple-client APIs*

PostgreSQL mendukung pengembangan aplikasi klien dalam banyak bahasa. Dalam menjelaskan bagaimana antarmuka ke PostgreSQL dari C, C ++, ODBC, Perl, PHP, Tcl / Tk, dan Python.

h) *Unique data types*

PostgreSQL menyediakan beragam tipe data. Selain tipe numerik, string, dan data biasa, Anda juga akan menemukan tipe geometris, tipe data Boolean, dan tipe data yang dirancang khusus untuk menangani alamat jaringan.

i) *Extensibility*

Salah satu fitur yang paling penting dari PostgreSQL adalah bahwa dapat dikembangkan. Jika tidak menemukan sesuatu yang Anda butuhkan, biasanya dapat menambahkannya sendiri. Misalnya, dapat menambahkan jenis data baru, fungsi dan operator baru, dan

bahkan bahasa prosedural dan klien baru. Ada banyak paket kontribusi yang tersedia di Internet. Sebagai contoh, *Refractions Research, Inc*. telah mengembangkan seperangkat tipe data geografis yang dapat digunakan untuk data model spasial (GIS) yang efisien.

26. Psycopg2

Begitu banyak bahasa pemrograman yang dapat menggunakan *database* PostgreSQL. Banyak orang yang berargumen bahwa PostgreSQL adalah *database open source* yang memiliki *library Application Programmable Interface* (API) terbesar untuk berbagai bahasa pemrograman (Wiki PostgreSQL, 2017).

Psycopg2 adalah adaptor untuk *database* PostgreSQL yang paling populer untuk bahasa pemrograman Python. Fitur utama dari psycopg2 adalah implementasi dari spesifikasi Python DB API 2.0 yang lengkap dan beberapa *thread* dapat berbagi koneksi yang sama. Psycopg2 dirancang untuk aplikasi yang berat dengan *multithread* yang dapat membuat kursor komputer tidak dapat bergerak dan membuat sejumlah perintah *INSERT* yang banyak dalam waktu yang bersamaan ataupun perintah *UPDATE* (Psycopg, 2017).

N. Firebase

Database adalah koleksi data yang terorganisir. *Database* dapat disimpan secara lokal di komputer atau dapat di simpan *cloud storages*. Setiap aplikasi Android, IOS atau aplikasi web memiliki *database* sendiri. Dalam aplikasi Android, kita bisa membuat *database* menggunakan SQLite, *shared preferences*, *websites* atau beberapa situs penyimpanan berbasis *cloud*. Ide dasar pembuatan *database* adalah menyimpan data secara sistematis dan mengambil data bila diperlukan. Firebase juga merupakan *database backend* untuk aplikasi Android, IOS dan *web*. Firebase adalah google yang disediakan API untuk membuat *database* dan ambil dari itu secara *real time* dengan hanya beberapa baris kode. Data disimpan sebagai JSON dan dapat diakses dari semua platform. Firebase adalah layanan berbayar dan dengan mendapatkan 200 MB ruang penyimpanan secara gratis (Singh, 2006).

Setelah API firebase dimasukkan ke dalam aplikasi Android atau IOS bisa memberi firebase fitur dengan baris kode sederhana. Fitur yang disediakan Firebase tercantum di bawah ini (Singh, 2006).

a) *Analytics*: Fitur ini memungkinkan pengembang aplikasi memahami bagaimana pengguna menggunakan aplikasinya. SDK menangkap *event* dan properti sendiri dan juga memungkinkan pengguna untuk melakukan kustomisasi pengambilan data. Dasbor memberikan rincian seperti pengguna yang paling aktif atau fitur aplikasi yang paling banyak digunakan. Selain itu memberikan data ringkas bagi pengguna.

b) *Authentication*: Fitur *auth* di firebase memberikan pengguna mengizinkan hanya pengguna yang berwenang mengakses aplikasi. Firebase menyediakan *login* melalui Gmail, Github, Twitter Facebook dan juga mengizinkan pengembang dalam kustomisasi otentikasi.

c) *Messaging*: *Cloud messaging* firebase memungkinkan pengguna dalam menyampaikan pesan ke berbagai platform tanpa biaya. Pesan juga digunakan untuk tujuan notifikasi.

d) *Real-time Database*: *Database* di firebase adalah *database* berbasis *cloud* dan tidak memerlukan *query* berbasis SQL untuk menyimpan dan mengambil data. *Database* sangat andal dan supercepat artinya data di*update* dan disinkronkan dalam waktu singkat dan data tetap terjaga bahkan user kehilangan koneksi internet.

e) *Storage*: Firebase juga menyediakan fasilitas penyimpanan. Ini dapat menyimpan dan mengambil konten seperti gambar, video dan audio langsung dari SDK klien. Mengunggah dan mengunduh dilakukan di bagian *background*. Penyimpanan data aman dan satu-satunya pengguna yang berwenang dapat mengaksesnya.

f) *Hosting*: Firebase juga digunakan untuk keperluan hosting. Firebase memberikan konten *web* dengan sangat cepat dan konten selalu terkirim dengan aman.

g) *Crash Reporting*: Fitur pelaporan *crash* di firebase membuat laporan kesalahan di aplikasi setelah diluncurkan. Kesalahan dikelompokkan ke dalam kelompok yang berbeda sesuai dengan

seberapa parah kesalahannya. Pengguna juga dapat membuat acara khusus untuk menangkap langkah-langkah yang mengarah pada perampokan aplikasi.

O. Android

Android adalah sebuah sistem operasi untuk perangkat *mobile* yang menyertakan *middleware (virtual machine)* dan sejumlah aplikasi utama. Android merupakan modifikasi dari kernel Linux (Andry, 2011).

Pada awalnya sistem operasi ini dikembangkan oleh sebuah perusahaan bernama Android, Inc. Dari sinilah awal mula nama Android muncul. Android Inc. adalah sebuah perusahaan *start-up* kecil yang berlokasi di Palo Alto, California, Amerika Serikat yang didirikan oleh Andy Rubin bersama Rich Miner, Nick Sears, dan Chris White. Pada bulan Juli 2005, perusahaan tersebut diakuisisi oleh Google dan para pendirinya bergabung ke Google. Andy Rubin sendiri kemudian diangkat menjadi Wakil Presiden divisi *Mobile* dari Google (Andry, 2011).

Tujuan pembuatan sistem operasi ini adalah untuk menyediakan *platform* yang terbuka, yang memudahkan orang mengakses Internet menggunakan telepon seluler. Android juga dirancang untuk memudahkan pengembang membuat aplikasi dengan batasan yang minim sehingga kreativitas pengembang menjadi lebih berkembang (Andry, 2011).

Sebagai *open source* dan bebas dalam memodifikasi, di dalam android tidak ada ketentuan yang tetap dalam konfigurasi *Software* dan *Hardware*. Fitur-fitur yang didapat dalam Android antara lain (Lee, 2011):

- *storage* - menggunakan SQLite, *database* yang ringan, untuk sebuah penyimpanan data;
- *connectivity* - mendukung GSM/EDGE, IDEN, CDMA, EV-DO, UMTS;
- *bluetooth* (termasuk A2DP dan AVRCP), WiFi, LTE, dan WiMax;
- *messaging* - mendukung SMS dan MMS;
- *web browser* - berbasiskan *open-source* WebKit, bersama mesin;
- *chrome's v8 javascript;*
- *media support* – termasuk mendukung untuk beberapa media berikut : H.263, H.264 (dalam bentuk 3GP or MP4), MPEG-4 SP, AMR, AMRWB (dalam bentuk 3GP), AAC, HE-AAC (dalam bentuk MP4 atau 3GP), MP3, MIDI, Ogg Vorbis, WAV, JPEG, GIF, dan BMP;
- *hardware support* - sensor akselerasi, Kamera, Kompas Digital, Sensor Kedekatan, GPS;
- *multi-touch* - mendukung *multi-touch screens;*
- *multi-tasking* - mendukung aplikasi *multi-tasking;*
- *flash-support* - android 2.3 mendukung Flash 10.1;
- *tethering* - mendukung pembagian dari koneksi internet sebagai *wired/wireless hotspot;*
- *play store* - katalog aplikasi yang dapat di-*download* dan diinstal pada telepon seluler secara *online*, tanpa menggunakan PC (*Personal*

Computer);

- serta lingkungan pengembangan yang kaya, termasuk *emulator*, peralatan *debugging*, dan *plugin* untuk Eclipse IDE.

27. Arsitektur Android

Arsitektur android dapat digambarkan seperti pada Gambar 28 dan secara garis besar arsitektur android dapat dijelaskan sebagai berikut (Safaat, 2012).

a) *Application* dan *Widgets*

Application dan *Widgets* ini adalah *layer* dimana kita berhubungan dengan aplikasi saja, dimana biasanya kita *download* aplikasi kemudian kita lakukan instalasi dan jalankan aplikasi tersebut. Di *layer* terdapat aplikasi inti termasuk klien email, program SMS, kalender, peta, *browser*, kontak, dan lain-lain. Hampir semua aplikasi ditulis menggunakan bahasa pemrograman Java.

b) *Application Framework*

Android adalah "*Open Development Platform*" yaitu android menawarkan kepada pengembang atau memberi kemampuan kepada pengembang untuk membangun aplikasi yang bagus dan inovatif. Pengembang bebas untuk mengakses perangkat keras, akses informasi *resource*, menjalankan *service background*, mengatur alarm, dan menambah status *notifications*, dan sebagainya. Pengembang memiliki akses penuh menuju API *framework* seperti yang dilakukan oleh aplikasi kategori inti.

Arsitektur aplikasi dirancang supaya kita dengan mudah dapat menggunakan kembali komponen yang sudah digunakan (*reuse*). Sehingga bisa kita simpulkan *Application Frameworks* ini adalah *layer* dimana para pembuat aplikasi melakukan pengembangan/pembuatan aplikasi yang akan dijalankan di sistem operasi android, karena pada *layer* inilah aplikasi dapat dirancang dan dibuat, seperti *content providers* yang berupa sms dan panggilan telepon.

Komponen-komponen yang termasuk di dalam *Application Frameworks* adalah sebagai berikut:

- *views;*
- *content provider;*
- *resource manager;*
- *notification manager;*
- *activity manager;*
- serta *libraries*.

c) *Libraries* ini adalah *layer* dimana fitur-fitur android berada, biasanya para pembuat aplikasi mengakses *libraries* untuk menjalankan aplikasinya. Berjalan di atas Kernel, layer ini meliputi berbagai *library* C/C++ inti seperti Libc SSL, serta:

- *libraries* media untuk pemutaran media *audio* dan *video;*
- *libraries* untuk manajemen tampilan;
- *libraries Graphics* mencakup SGL dan OpenGL untuk grafis 2D dan 3D;

- *libraries SQLite* untuk dukungan *database;*
- *libraries* SSL dan *Web*Kit terintegrasi dengan *web browser* dan *security;*
- *libraries LiveWebcore* mencakup *modern web browser* dengan *engine embedded web view;*
- serta *libraries* 3D yang mencakup implementasi OpenGL ES1.0 API's.

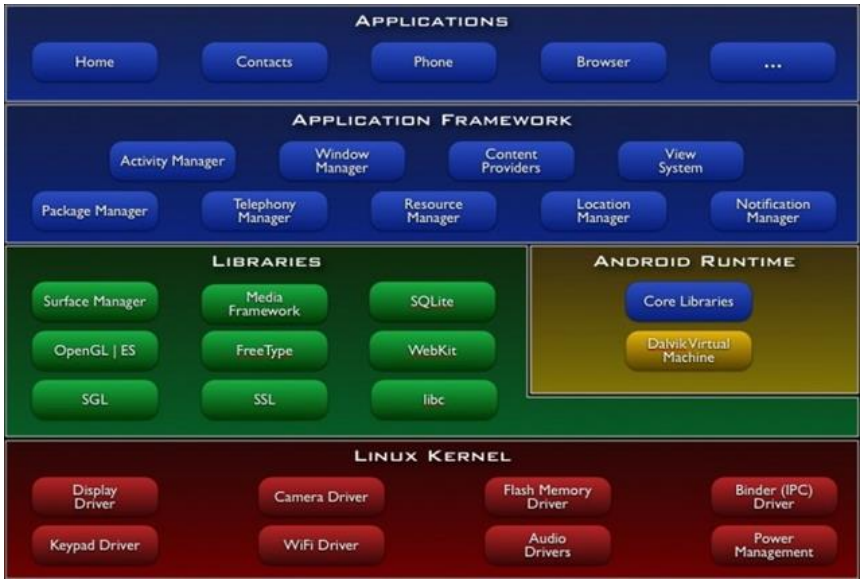
d) Android *Run Time*

Layer yang membuat aplikasi android dapat dijalankan dimana dalam prosesnya menggunakan implementasi Linux. *Dalvik Virtual Machine* (DVM) merupakan mesin yang membentuk dasar kerangka aplikasi Android. Di dalam Android *Run Time* dibagi menjadi dua bagian yaitu: (a) *Core Libraries* adalah aplikasi Android dibangun dalam bahasa Java, sementara Dalvik sebagai virtual mesinnya bukan *Virtual Machine Java*, sehingga diperlukan sebuah *libraries* yang berfungsi untuk menerjemahkan bahasa Java/C yang ditangani oleh *Core Libraries*. (b) *Dalvik Virtual Machine* adalah Virtual mesin berbasis *register* yang dioptimalkan untuk menjalankan fungsi-fungsi secara efisien, dimana merupakan pengembangan yang mampu membuat Linux Kernel untuk melakukan *threading* dan manajemen tingkat rendah.

e) Linux Kernel

Linux Kernel adalah *layer* dimana inti dari sistem operasi android itu berada. Berisi *file-file* sistem yang mengatur sistem *processing,*

memory, resource, drivers, dan sistem-sistem operasi Android lainnya. Linux Kernel yang digunakan android adalah Linux Kernel *release* 2.6.



Gambar 28 Arsitektur Android (Andry, 2011)

28. Android SDK

Android SDK adalah *tools* API (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada *platform* android menggunakan bahasa pemrograman Java. Android merupakan sub set perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang dirilis oleh Google. Saat ini disediakan Android SDK (*Software Development Kit*) sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi netral, android member membuka kesempatan untuk membuat aplikasi

yang kita butuhkan yang bukan merupakan aplikasi bawaan *Handphone* atau *Smartphone* (Developers, 2014).

29. Android Studio

Android Studio adalah lingkungan pengembangan terpadu (IDE) resmi untuk pengembangan *platform* Android, hal itu disampaikan pada tanggal 16 Mei 2013 oleh Google, Android Studio sudah tersedia secara bebas dibawah lisensi Apache 2.0. Android Studio pada awalnya tahap *preview* versi 0.1 yang dipakai pada tanggal 1 Mei 2013 dan memasuki tahap Beta pada bulan Juni 2014 dan mulai stabil dirilis pada Desember 2014 dengan versi 1.0, berdasarkan *jetBrains 'IDEA* Intellij *Software*, Android Studio dirancang khusus untuk pengembangan Android yang tersedia untuk Windows, Mac OS X, dan Linux sebagai pengganti Eclipse. Android Studio adalah *official* IDE yang digunakan untuk pengembangan aplikasi Android berdasarkan Intellij IDEA. Dengan kemampuan di atas yang diharapkan dari Intellij, Android Studio ini menawarkan (Developers, 2014):

- *flexible Gradle* yang berbasis membangun system;
- membangun varian dan beberapa berkas turunan apk;
- kode *template* untuk membantu dalam membangun fitur aplikasi standar;
- *rich layout editor* dengan dukungan untuk drag dan pengubahan tema penurunan;
- *link tools* yang digunakan untuk menangkap kinerja, kegunaan,

kompatibilitas versi , dan masalah lainnya;

- *proGuard* dan aplikasi *signing capabilities*.
- dan *built-in support* yang digunakan untuk *Google Cloud Platform*, sehingga memudahkan untuk mengintegrasikan *Google Cloud Messaging* dan *App Engine*.

30. Fundamental Aplikasi

Aplikasi Android ditulis dalam bahasa pemrograman Java, kode Java dikompilasi bersama dengan data *file resource* yang dibutuhkan oleh aplikasi dimana prosesnya di-*package* oleh *tools* yang dinamakan "*apt tools*" ke dalam paket Android sehingga menghasilkan *file* dengan ekstensi apk (*Android Package*). *File* apk itulah yang sebenarnya kita sebut dengan aplikasi yang dapat diinstal di perangkat *mobile* nantinya. Ada empat jenis komponen pada aplikasi Android yaitu (Safaat, 2012).

a) *Activites*

Suatu *activity* akan menyajikan *User Interface* (UI) kepada pengguna, sehingga pengguna dapat melakukan interaksi. Sebuah aplikasi Android bisa jadi hanya memiliki satu *activity*, tetapi umumnya aplikasi memiliki banyak *activity* tergantung pada tujuan aplikasi dan desain dari aplikasi tersebut. Satu *activity* biasanya akan dipakai untuk menampilkan aplikasi atau yang bertindak sebagai *user interface* saat aplikasi diperlihatkan kepada *user*. Untuk pindah dari satu *activity* ke *activity* yang lain kita dapat melakukan dengan satu *event* misalnya klik tombol, memilih opsi atau menggunakan

triggers tertentu. Secara hierarki sebuah *windows activity* dinyatakan dengan *method Activity*.setContentView(). *ContentView* adalah objek yang berada pada *root* hierarki.

b) *Service*

Service tidak memiliki visual *user interface* (UI), tetapi *service* berjalan secara *background*, sebagai contoh dalam memainkan musik, *service* mungkin memainkan musik atau mengambil data dari jaringan, tetapi setiap *service* haruslah berada dalam kelas induknya. Misalnya *media player* sedang memutar lagu dari *list* yang ada, aplikasi ini akan memiliki dua atau lebih *activity* yang memungkinkan *user* untuk memilih lagu atau menulis SMS sambil *player* sedang jalan. Untuk menjaga musik tetap dijalankan, *activity player* dapat menjalankan *service* untuk membuat aplikasi tetap berjalan. *Service* dijalankan pada *thread* utama dari proses aplikasi.

c) *Broadcast Recevier*

Broadcast Recevier berfungsi menerima dan bereaksi untuk menyampaikan notifikasi. *Broadcast Recevier* tidak memiliki *user interface* (UI), tetapi memiliki sebuah *activity* untuk merespon informasi yang mereka terima, atau mungkin menggunakan *Notification Manager* untuk memberitahu kepada pengguna, seperti lampu latar atau *vibrating* (getaran) perangkat, dan lain sebagainya.

d) *Content Provider*

Content provider membuat kumpulan aplikasi data secara spesifik sehingga bisa digunakan oleh aplikasi lain. Data disimpan dalam *file*

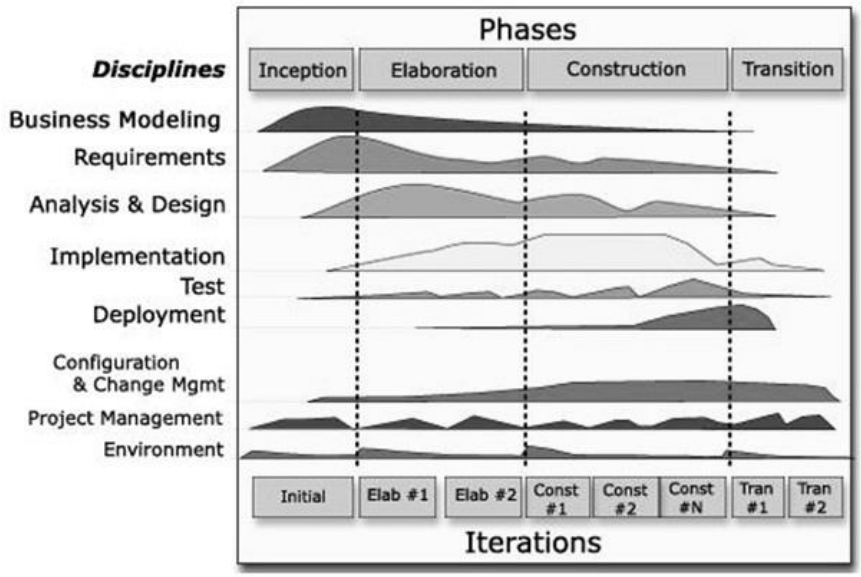
System seperti *database SQLite*. *Content Provider* menyediakan cara untuk mengakses data yang dibutuhkan oleh suatu *activity*. Misalnya ketika kita menggunakan aplikasi yang membutuhkan peta atau aplikasi yang membutuhkan cara untuk mengakses data kontak untuk navigasi, maka disinilah fungsi *content provider*.

P. Metodologi Pengembangan Sistem

Adapun metodologi yang digunakan dalam pembuatan aplikasi SIAKAD menggunakan RESTful *Web Service* dan Otentikasi dengan OAuth *2* berbasis android ini, antara lain meliputi : *unified process* (UP) dan desain menggunakan *Unified Modeling Language* (UML).

31. *Unified Process* (UP)

Unified process adalah salah satu model pengembangan *software* yang populer yang digunakan untuk membangun sistem yang *object-oriented* (Larman, 2002). *Unified process* mengombinasikan pendekatan umum terbaik, seperti siklus iteratif dan pengembangan dengan risiko yang terkendali, menjadi sebuah deskripsi yang terdokumentasi dengan baik dan bersifat kohesif. *Unified process* merupakan dasar dari beberapa model pemrosesan software lain, seperti: RUP (*Rational Unified Process*), OpenUP (Open *Unified Process*), dan lain-lain (Kroll dan MacIsaac, 2006).



Gambar 29 Arsitektur *Unified Process* (Kroll dan MacIsaac, 2006)

Siklus u*nified process* membagi sebuah proyek menjadi 4 fase besar antara lain (Kroll dan MacIsaac, 2006).

a) *Inception*, memperkirakan visi, meninjau risiko-risiko yang terdapat dalam bisnis dan menjadikannya permasalahan dalam bisnis, membuat ruang lingkup sistem, dan estimasi ketidakpastian.

b) *Elaboration*, merevisi visi yang ada, mengurangi risiko terbesar dengan cara menangani tugas-tugas tersulit yang ada agar estimasi biaya dan penjadwalan dapat diperbarui, dan mendesain, mengimplementasikan, *testing*, dan membuat garis besar inti arsitektur, mengidentifikasi kebutuhan dan ruang lingkup yang paling besar.

c) *Construction*, membangun keseluruhan sistem mulai dari elemen terbesar hingga yang terkecil secara bertahap. Akhir dari fase ini adalah sebuah sistem *software* tahap *beta* yang sudah

terdokumentasi dan dapat digunakan oleh pengguna untuk dicoba.

d) *Transition*, testing sistem dan memenuhi sisa kebutuhan pengguna yang masih belum terpenuhi sebelum dilepas ke pasaran.

Seperti yang terlihat pada Gambar 29, setiap fase pada *unified process* memiliki iterasinya sendiri-sendiri dimana dari setiap iterasi tersebut akan menghasilkan sistem yang bekerja sampai pada tahap tertentu sehingga memungkinkan pengguna melihat peningkatan yang terjadi.

32. Unified Modeling Language (UML)

Unified Modeling Language (UML) adalah keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek (OO). Definisi ini merupakan definisi yang sederhana. Pada kenyataannya, pendapat orang – orang tentang UML berbeda satu sama lain. Hal ini dikarenakan oleh sejarahnya sendiri dan oleh perbedaan persepsi tentang apa yang membuat sebuah proses rancang – bangun perangkat lunak efektif.

Unified Modeling Language (UML) merupakan standar yang relatif terbuka yang dikontrol oleh *Object Management Group* (OMG), sebuah konsorsium terbuka yang terdiri dari banyak perusahaan. OMG dibentuk untuk membuat standar – standar yang mendukung *interoperabilitas*, khususnya interoperabilitas sistem 41 berorientasi objek. OMG mungkin lebih dikenal dengan standar – standar COBRA (*Common Object Request Broker Architecture*).

UML lahir dari penggabungan banyak bahasa permodelan grafis berorientasi objek yang berkembang pesat pada akhir 1980-an dan awal 1990-an. UML dibuat oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson di bawah bendera Rational Software Corp. UML menyediakan notasi-notasi yang membantu memodelkan sistem dari berbagai perspektif. UML tidak hanya digunakan dalam permodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan permodelan (Fowler, 2004).

UML dideskripsikan oleh beberapa diagram, yaitu sebagai berikut.

a. *Use Case* Diagram

Use case Diagram digunakan untuk menggambarkan sistem dari sudut pandang pengguna sistem tersebut (*user*), sehingga pembuatan *use case* diagram lebih dititikberatkan pada fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian. Sebuah *use case* diagram merepresentasikan sebuah interaksi antara aktor dengan sistem (Fowler, 2004).

Komponen-komponen dalam *use case* diagram (Fowler, 2004) :

a) Aktor

Pada dasarnya aktor bukanlah bagian dari *use case* diagram, namun untuk dapat terciptanya suatu *use case* diagram diperlukan aktor, dimana aktor tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat atau 42 sistem lain) yang berinteraksi dengan sistem yang dibuat. Sebuah aktor mungkin hanya memberikan informasi inputan pada sistem,

hanya menerima informasi dari sistem atau keduanya menerima dan memberi informasi pada sistem. Aktor hanya berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case*. Aktor digambarkan dengan *stick pan* seperti yang terdapat pada Gambar 30.

Gambar 30 Contoh Aktor (Fowler, 2004)

b) *Use Case*

Use case adalah Gambaran fungsionalitas dari suatu sistem, sehingga pengguna sistem paham dan mengerti kegunaan sistem yang akan dibangun. Bentuk *use case* dapat terlihat pada Gambar 31.

Gambar 31 *Use Case* (Fowler, 2004)

Ada beberapa relasi yang terdapat pada *use case* diagram:

- *association*, menghubungkan link antar element;
- *generalization*, disebut juga pewarisan (*inheritance*), sebuah elemen dapat merupakan spesialisasi dari elemen lainnya;

- *dependency*, sebuah element bergantung dalam beberapa cara ke element lainnya;
- dan *aggregation*, bentuk *association* dimana sebuah elemen berisi elemen lainnya.

Tipe relasi yang mungkin terjadi pada *use case* diagram:

- <<*include*>>, yaitu kelakuan yang harus terpenuhi agar sebuah *event* dapat terjadi, dimana pada kondisi ini sebuah *use case* adalah bagian dari *use case* lainnya;
- <<*extend*>>, kelakuan yang hanya berjalan di bawah kondisi tertentu seperti menggerakkan peringatan;
- dan <<*communicated*>>, merupakan pilihan selama asosiasi hanya tipe *relationship* yang dibolehkan antara aktor dan *use case*.

b. *Activity* Diagram

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktivitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktivitas lainnya (Fowler, 2004). Notasi *Activity* diagram dapat dilihat pada Tabel 7.

Tabel 7 Notasi *Activity* Diagram (Meildy, 2014)

Simbol	Keterangan

	*Activity*
	Titik awal
	Titik akhir
	Pilihan untuk mengambil keputusan

Tabel 7 Notasi *Activity* Diagram Lanjutan (Meildy, 2014)

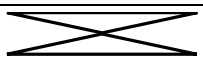
Simbol	Keterangan
	*Fork*; Digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel atau untuk menggabungkan dua kegiatan paralel menjadi satu
	*Rake*; Menunjukkan adanya dekomposisi
	Tanda waktu
	Tanda pengiriman
	Aliran akhir (*flow final*)

Diagram ini sangat mirip dengan *flowchart* karena memodelkan *workflow* dari satu aktivitas ke aktivitas lainnya atau dari aktivitas ke status. Pembuatan *activity* diagram pada awal pemodelan proses dapat membantu memahami keseluruhan proses. *Activity* diagram juga digunakan untuk menggambarkan interaksi antara beberapa *use case* (Fowler, 2004).

c. *Class* Diagram

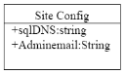
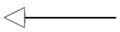
Class adalah sebuah spesifikasi yang akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain

berorientasi objek. *Class* diagram menggambarkan struktur dan deskripsi *Class*, *package* dan objek beserta hubungan satu sama lain seperti pewarisan, asosiasi, dan lain-lain. Notasi *Class* diagram dapat dilihat pada Tabel 8.

Hubungan antar *Class* (Fowler, 2004).

a) Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain.
b) Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas").
c) Pewarisan, yaitu hubungan hierarki antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metode *class* asalnya serta bisa menambahkan fungsionalitas baru. Sehingga *class* tersebut disebut anak dari *class* yang diwarisinya.
d) Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-class dari satu class kepada class lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence* diagram yang akan dijelaskan kemudian

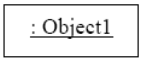
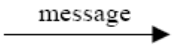
Tabel 8 Notasi *Class* Diagram (Meildy, 2014)

Simbol	Nama	Keterangan
	*Class*	*Class* adalah blok-blok pembangun pada pemrograman berorientasi obyek. Sebuah *class* digambarkan sebagai sebuah kotak yang terbagi atas 3 bagian. Bagian atas adalah bagian nama dari *class*. Bagian tengah mendefinisikan *property*/atribut *class*. Bagian akhir mendefinisikan *method* dari sebuah *class*.
	Association	Sebuah asosiasi merupakan sebuah *relationship* paling umum antara 2 *class*, dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 *class*. Garis ini bisa melambangkan tipe-tipe *relationship* dan juga dapat menampilkan hukum-hukum multiplisitas pada sebuah relationship (Contoh: *One-to-one*, *one-to-many*, *many-to-many*).
	*Composition*	Jika sebuah *class* tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *composition* terhadap *class* tempat dia bergantung tersebut. Sebuah *relationship composition* digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/*solid*.
	Dependency	Kadang kala sebuah *class* menggunakan *class* yang lain. Hal ini disebut *dependency*. Umumnya penggunaan *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain. Sebuah *dependency* dilambangkan sebagai sebuah panah bertitik-titik.
	*Aggregation*	*Aggregation* mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut sebagai relasi "mempunyai sebuah" atau "bagian dari". Sebuah aggregation digambarkan sebagai sebuah garis dengan sebuah jajaran genjang yang tidak berisi/tidak *solid*.
	*Generalization*	Sebuah relasi *generalization* sepadan dengan sebuah relasi *inheritance* pada konsep berorientasi obyek. Sebuah *generalization* dilambangkan dengan sebuah panah dengan kepala panah yang tidak solid yang mengarah ke kelas "*paren*t"-nya/induknya.

d. *Sequence* Diagram

Menggambarkan interaksi antara sejumlah objek dalam urutan waktu. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antar objek yang terjadi pada titik tertentu dalam eksekusi sistem (Fowler, 2014). Notasi *sequence* diagram dapat dilihat pada Tabel 9.

Tabel 9 Notasi *Sequence* Diagram (Meildy, 2014)

Simbol	Nama	Keterangan
	*Object*	*Object* merupakan *instance* dari sebuah *class* dan dituliskan tersusun secara horizontal. Digambarkan sebagai sebuah *class* (kotak) dengan nama obyek di dalamnya yang diawali dengan sebuah titik koma
	*Actor*	*Actor* juga dapat berkomunikasi dengan *object*, maka *actor* juga dapat diurutkan sebagai kolom. Simbol *actor* sama dengan simbol pada *actor use case* diagram
	Lifeline	*Lifeline* mengindikasikan keberadaan sebuah *object* dalam basis waktu. Notasi untuk *lifeline* adalah garis putus-putus vertikal yang ditarik dari sebuah obyek
	*Activation*	*Activation* dinotasikan sebagai sebuah kotak segi empat yang digambar pada sebuah *lifeline*. *Activation* mengindikasikan sebuah obyek yang akan melakukan sebuah aksi
	*Message*	*Message*, digambarkan dengan anak panah *horizontal* antara *activation*. *Message* mengindikasikan komunikasi antara *object-object*.

Q. Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah perangkat lunak secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum (Clune dan Rood, 2011). Singkat kata, pengujian adalah aktivitas untuk menemukan dan menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya.

33. Teknik Pengujian Perangkat Lunak

Ada dua macam pendekatan kasus uji yaitu *white box* dan *black box*. Pendekatan *white box* adalah pengujian untuk memperlihatkan cara kerja dari produk secara rinci sesuai dengan spesifikasinya (Jiang dan Lu, 2012). Jalur logika perangkat lunak akan dites dengan menyediakan kasus uji yang akan mengerjakan kumpulan kondisi dan pengulangan secara spesifik. Sehingga melalui penggunaan metode ini akan dapat memperoleh kasus uji yang menjamin bahwa semua jalur independen pada suatu model telah digunakan minimal satu kali, penggunaan keputusan logis pada sisi benar dan salah, pengeksekusian semua loop dalam batasan dan batas operasional perekayasa, serta penggunaan struktur data internal guna menjamin validitasnya (Pressman, 2001).

Pendekatan *black box* merupakan pendekatan pengujian untuk mengetahui apakah semua fungsi perangkat lunak telah berjalan semestinya sesuai dengan kebutuhan fungsional yang telah didefinisikan (Jiang dan Lu, 2012). Kasus uji ini bertujuan untuk menunjukkan fungsi perangkat lunak tentang cara beroperasinya. Teknik pengujian ini

berfokus pada domain informasi dari perangkat lunak, yaitu melakukan kasus uji dengan mempartisi domain input dan output program.

Metode *black box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi *input* yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian ini berusaha menemukan kesalahan dalam kategori fungsi-fungsi yang tidak benar atau hilang, kesalahan *interface*, kesalahan dalam struktur data atau akses basis data eksternal, kesalahan kinerja, dan inisialisasi dan kesalahan terminal (Pressman, 2001).

Teknik pengujian yang digunakan dalam penelitian ini terbagi atas pengujian fungsional dan pengujian *non* fungsional. Adapun pengujian fungsional menggunakan metode *black box* yaitu *Equivalence Partitioning* dan pengujian non fungsional menggunakan angket yang penyusunan bentuk jawaban dari pertanyaan menggunakan skala likert.

34. Equivalence Partitioning

Equivalence partitioning (EP) merupakan metode black box testing yang membagi domain masukan dari program kedalam kelas-kelas sehingga *test case* dapat diperoleh. *Equivalence partitioning* berusaha untuk mendefinisikan kasus uji yang menemukan sejumlah jenis kesalahan, dan mengurangi jumlah kasus uji yang harus dibuat. Kasus uji yang didesain untuk *equivalence partitioning* berdasarkan pada evaluasi dari kelas ekuivalensi untuk kondisi masukan yang menggambarkan kumpulan keadaan yang valid atau tidak. Kondisi

masukan dapat berupa spesifikasi nilai numerik, kisaran nilai, kumpulan nilai yang berhubungan atau kondisi *boolean*.

Kesetaraan kelas dapat didefinisikan menurut panduan berikut (Pressman, 2001) :

- jika masukan kondisi menentukan kisaran, satu sah dan dua diartikan tidak *valid* kesetaraan kelas;
- jika masukan membutuhkan nilai, kondisi tertentu satu sah dan dua tidak valid kesetaraan kelas diartikan;
- jika masukan kondisi menentukan anggota dari set, satu sah dan satu tidak valid kesetaraan kelas diartikan;
- dan jika kondisi yang input, boolean satu sah dan satu tidak valid kelas diartikan.

35. Skala Likert

Menurut Likert (Azwar, 2011), sikap dapat diukur dengan metode *rating* yang dijumlahkan (*Method of Summated Ratings*). Metode ini merupakan metode penskalaan pernyataan sikap yang menggunakan distribusi respons sebagai dasar penentuan nilai skalanya. Nilai skala setiap pernyataan tidak ditentukan oleh derajat *favourable* nya masing-masing akan tetapi ditentukan oleh distribusi respon setuju dan tidak setuju dari sekelompok responden yang bertindak sebagai kelompok uji coba (pilot study) (Azwar, 2011). Skala Likert, yaitu skala yang berisi lima tingkat preferensi jawaban dengan pilihan sebagai berikut: 1 =

sangat tidak setuju; 2 = tidak setuju; 3 = ragu–ragu atau netral; 4 = setuju; 5 = sangat setuju. Selanjutnya, penentuan kategori interval tinggi, sedang, atau rendah digunakan rumus sebagai berikut :

$$I = \frac{NT - NR}{K}$$

Keterangan :
I = interval;
NT = total nilai tertinggi;
NR = total nilai terendah;
K = kategori jawaban (Yitnosumarto, 2006).

R. Penelitian Terkait

Hasil penelitian yang relevan dengan penelitian ini adalah.

a) Implementasi RESTful *Web Services* Pada Sistem Informasi Wisata Alam dan Wisata Kuliner DIY oleh Fajar (2015). Pada penelitian ini bertujuan untuk memberikan solusi atas permasalahan yang ada pada Jogjanan, yaitu dengan mengganti arsitektur *web service* pada Jogjanan dengan menggunakan RESTful *web service*. Dengan arsitektur *web service* ini, diharapkan mampu menjadi jembatan informasi yang menghubungkan penyedia data dengan *client* yang membutuhkan informasi, khususnya informasi pariwisata yang lengkap dan akurat beserta informasi-informasi pendukungnya bagi Daerah Istimewa Yogyakarta.

b) Pemanfaatan *Web Services* dalam Sistem Informasi Daftar Pemilih Tetap oleh Allokendek, *et al* (2014). Dalam penelitian ini, *web service* digunakan untuk mengkomunikasikan dua aplikasi yang

berbeda yaitu SIAK DISDUKCAPIL Kabupaten Maros dan SIDP KPU Maros. Penelitian ini dilanjutkan dengan membuat suatu desain sistem dan implementasi berupa *prototype* sistem yang mengintegrasikan data dari *database* SIAK di DISDUKCAPIL dengan *database* KPU di Kab. Maros dengan memanfaatkan teknologi *web services*. Hasilnya adalah didapatnya DPT yang *valid* dan tidak terdapat data yang ganda serta adanya alternatif lain dalam media publikasi oleh KPU untuk mempublikasikan DPT kepada masyarakat, disamping tetap menggunakan media publikasi yang telah digunakan selama ini.

c) Perancangan dan Implementasi *Resource Server* dan *Authorization Server* Menggunakan Teknologi Otentikasi OAuth *2* oleh Yosrinal (2014). Pada penelitian ini digunakan 3 (tiga) aplikasi yang bersifat *single sign on*, terdiri atas 3 halaman *sign-in* berbasis otentikasi OAuth 2 dari pemilik sumber daya (*Resource Owner*) dengan menerapkan proses kerja otentikasi OAuth 2. Otentikasi OAuth 2 lewat peran *Authorization Server* akan memvalidasi *credential* dari *client* seusai keberadaannya pada basis data, di proses dengan mengeluarkan sebuah halaman otorisasi (*Authorize App*) untuk diarahkan ke halaman utama setiap aplikasi *web* masing-masing. Otorisasi akhir dari kebenaran *credential* seorang *client* adalah di hasilkan sebuah akses token yang bekerja pada url (*uniform resource locator*) pada masing - masing aplikasi *web*.

d) Pengembangan *Web Service* Pengurai Morfologi Bahasa Indonesia Pada *Language Grid* oleh Christian (2009). Dalam penelitian ini, akan dikembangkan sebuah *web service* yang memberikan layanan analisis morfologi bahasa Indonesia menggunakan program *Morphological Analyzer* yang telah dikembangkan sebelumnya oleh Femphy Pisceldo (Pisceldo, 2008). Langkah lebih lanjut adalah mengembangkan *wrapper* yang menggunakan teknologi *web service* agar layanan ini dapat diakses melalui infrastruktur *Language Grid*. Perancangan dari *web service* ini meliputi perancangan *web application* pada *server side* yang berkomunikasi langsung dengan program *Morphological Analyzer*, dan perancangan file WSDL yang mendefinisikan layanan yang disediakan. Selain itu, juga dilakukan perancangan *web application* pada *client side* untuk melakukan uji coba dari *web service* yang dihasilkan. Pada akhirnya, *web service* yang menyediakan layanan analisis morfologi ini berhasil dibuat, namun belum dilakukan *deployment* ke dalam infrastruktur *Language Grid* karena kendala teknis dan keterbatasan waktu pengembangan. *Web service* yang dihasilkan juga telah diuji coba dan telah sukses melewati uji coba tersebut.

III. METODOLOGI PENELITIAN

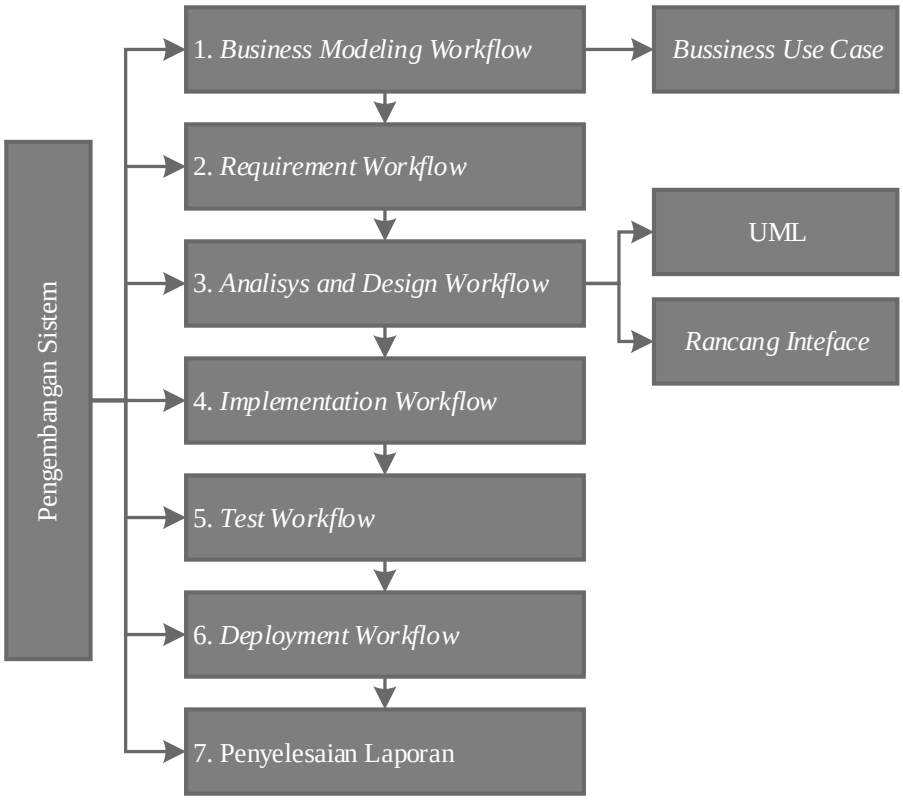
A. Waktu dan Tempat Penelitian

Penelitian ini dilakukan di Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Lampung yang berada di Jalan Soemantri Brojonegoro No.1 Gedung Meneng, Bandar Lampung. Penelitian ini dilaksanakan mulai bulan April 2017 sampai bulan September 2017.

B. Metodologi Penelitian

1. Alir Penelitian

Diagram Alir merupakan urutan alur kegiatan yang dilakukan dalam suatu penelitian. Penelitian penerapan Django REST Framework dan Teknologi Otentikasi 2.0 versi Android ini dilakukan dimulai dengan pengembangan sistem kemudian dilanjutkan dengan penyelesaian laporan. Bagan dari diagram alir penelitian penerapan Django REST Framework dan Teknologi Otentikasi 2.0 versi Android dapat dilihat pada Gambar 32.



Gambar 32 Diagram Alir Metodologi Penelitian

Metode yang digunakan pengembangan sistem informasi administrasi PKM adalah *Rational Unified Process* (RUP). Metode RUP memberikan keleluasaan pengembang dalam pengembangan sistem karena pada satu fase pengembang dapat melaksanakan lebih dari satu *workflow* pengembangan. *Workflow* pada penerapan Django REST Framework dan Teknologi Otentikasi 2.0 versi Android ini terdiri dari perencanaan *project management*, *business modeling*, *requirement*, *analysis* dan *design*, *implementation test*, *deployment*, *configuration and change management*, dan *environment workflow*. RUP memiliki empat fase tersebut yaitu *inception*, *elaboration*, *construction*, dan *transition*.

RUP dilakukan secara berurutan dan *iterative* dimana setiap iterasi dapat digunakan untuk memperbaiki iterasi berikutnya.

2. *Business Modeling Workflow*

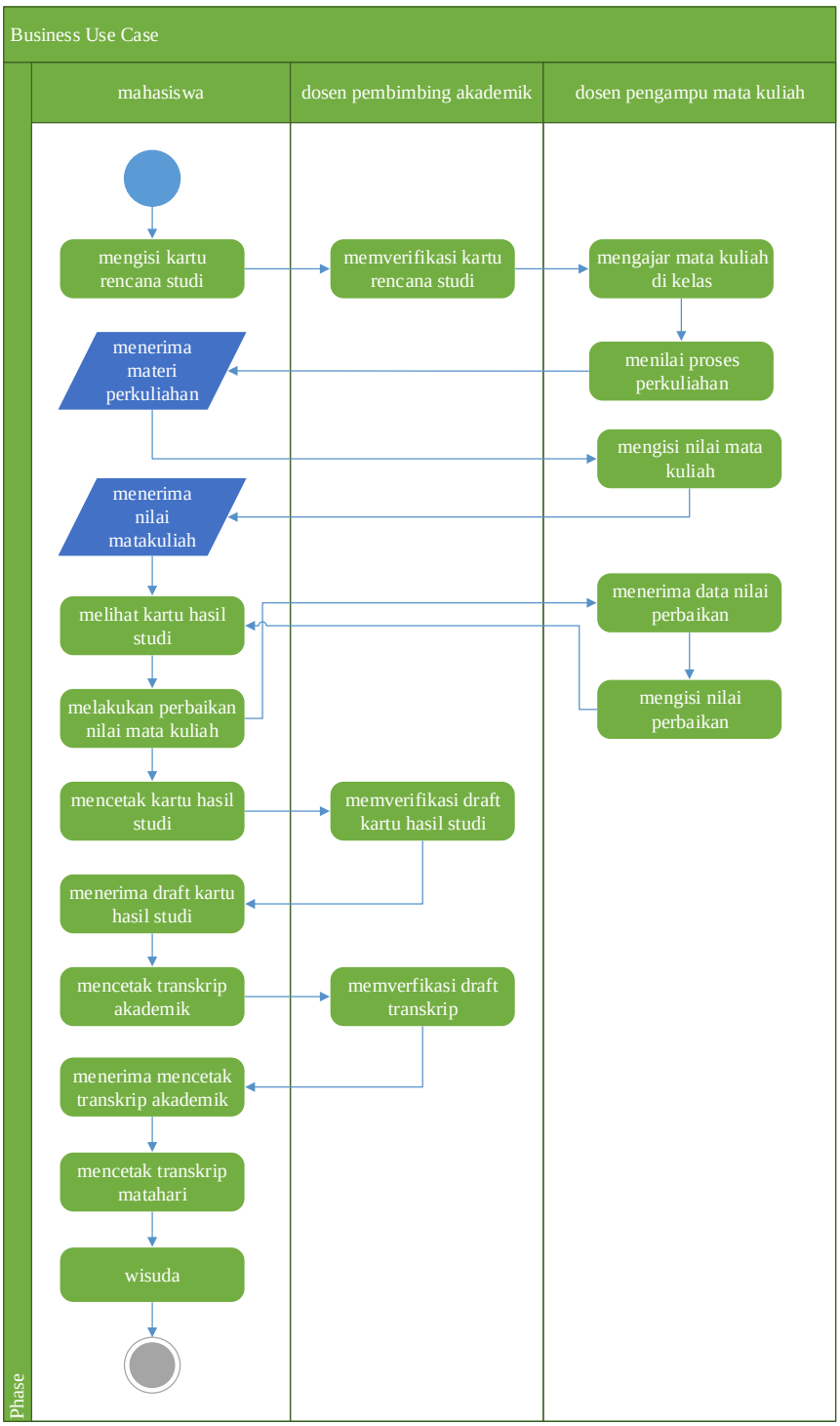
Business Modeling Workflow merupakan analisis untuk memahami bagaimana bisnis harus mendukung proses bisnis yang ada dalam organisasi. Kegiatan pengembangan yang dilakukan pada proses *business modelling workflow* yaitu menetapkan proses bisnis sistem informasi akademik Universitas Lampung. Pemodelan proses bisnis SIAKAD di Universitas Lampung dimulai dengan mengidentifikasi masalah kemudian membuat model proses bisnis SIAKAD di Universitas Lampung.

a. Indentifikasi Masalah

Tahap identifikasi masalah merupakan tahapan dasar dimana pada tahapan ini dilakukan pengidentifikasian dan penganalisan terhadap permasalahan-permasalahan yang ada. Tahapan ini akan menghasilkan perumusan masalah, tujuan penelitian, manfaat penelitian dan juga batasan-batasan permasalahan. Rumusan masalah merupakan kalimat pertanyaan untuk menunjukkan penelitian mengarah pada suatu permasalahan atau isu tertentu. Manfaat penelitian menguraikan manfaat dari aplikasi SIAKAD Unila menggunakan Django RESTful *Framework* dengan teknologi OAuth *2.0* untuk memudahkan mahasiswa dan dosen dalam mengakses sistem akademik. Sedangkan batasan masalah digunakan untuk membatasi pembahasan dan ruang lingkup penelitian.

b. *Business Use Case*

Tahap ini juga menerjemahkan proses bisnis dalam bentuk *business use case*. *Business use case* menjelaskan mengenai rancangan alur kegiatan pelayanan sistem informasi akademik di Universitas Lampung lingkup mahasiswa, dosen pembimbing akademik dan pengampu mata kuliah. Proses yang dijelaskan meliputi proses mengisi KRS (Kartu Rencana Studi), mengikuti perkuliah, memverifikasi rencana studi mahasiswa, proses menilai mata kuliah yang dilakuikan oleh dosen pengampu, mencetak KHS (Kartu Hasil Studi), memverfikasi hasil studi, mencetak tranksrip akhir, dan proses wisuda. Rancangan *business use case* sistem informasi akademik lingkup mahasiswa, dosen pembimbing akademik, dan dosen pengampu mata kulaih di Universitas Lampung dapat dilihat pada Gambar 33.



Gambar 33 *Business Use Case* Alur Pelayanan Sistem Informasi Akademik Universitas Lampung Lingkup Mahasiswa, Dosen Pembimbing Akademik dan Pengampu Matakuliah

3. *Requirement Workflow*

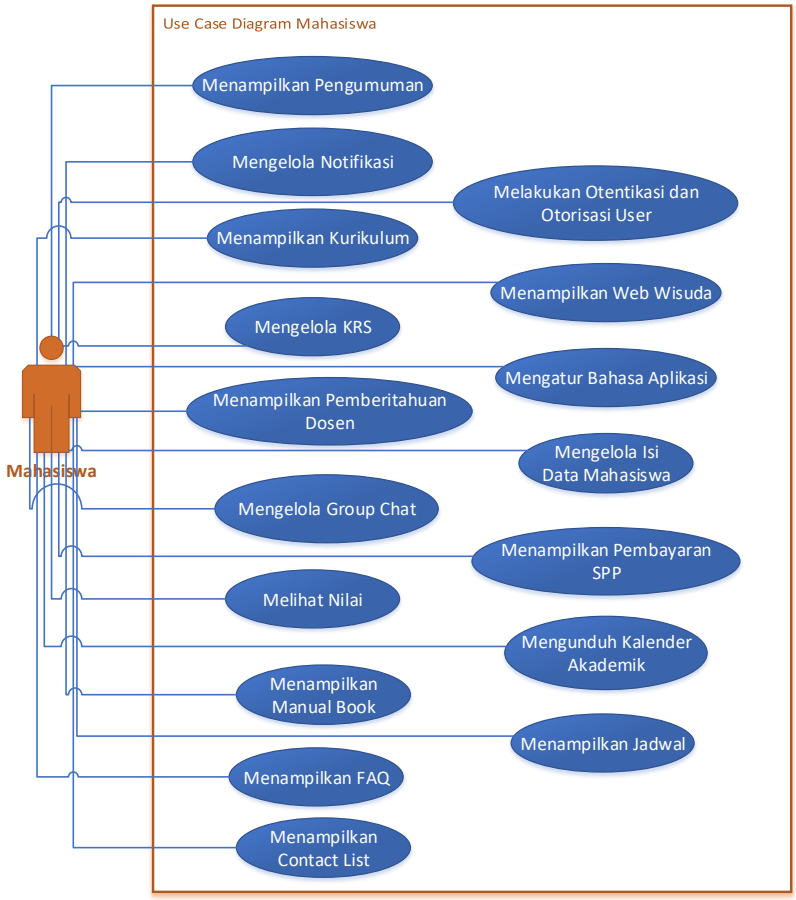
Kegiatan pengembangan yang dilakukan pada *requirement workflow* yaitu mengidentifikasi aktor, baik pengguna sistem lain yang berinteraksi dengan sistem informasi akademik Universitas Lampung. Tahap ini juga mengidentifikasi dan mengembangkan *use case* yang sesuai dengan kebutuhan aktor. Adapun aktor yang terlibat dalam proses bisnis ini yaitu mahasiswa, dosen (pembimbing akademik dan pengampu mata kuliah) di Univeristas Lampung.

Use case diagram digunakan untuk menggambarkan sistem dari sudut pandang pengguna sistem tersebut (*user*), sehingga pembuatan *use case* diagram ini lebih dititikberatkan pada fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian. Pada sistem ini terdapat dua *use case* diagram yaitu *use case* diagram untuk pengguna aplikasi (mahasiswa) dan *use case* diagram untuk dosen pengampu mata kuliah dan pembimbing akademik.

1) *Use case* Diagram untuk Mahasiswa

   Pada ini pengguna *role* mahasiswa dapat melakukan 17 interaksi antara lain menampilkan pengumuman, mengelola notifikasi, menampilkan kurikulum, mengelola KRS, menampilkan pemberitahuan pengguna dosen, mengelola forum, melihat nilai, menampilkan *manual book*, menampilkan FAQ, menampilkan *contact list*, mengunduh kalender akademik, menampilkan pembayaran SPP, mengolah isi data mahasiswa (IDM), mengatur bahasa aplikasi, menampilkan *web* wisuda, serta menampilkan

otentikasi dan otorisasi *user*. *Use case* untuk pengguna *role* mahasiswa disajikan pada Gambar 36.

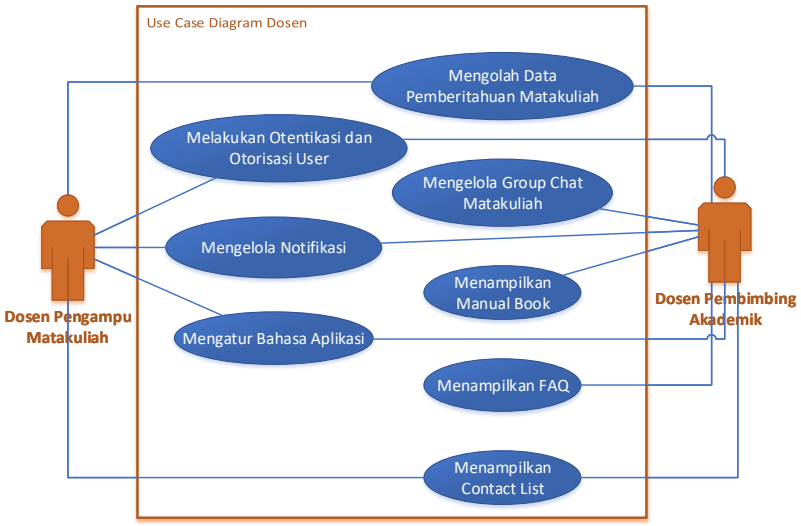


Gambar 34 *Use Case* Diagram Untuk Pengguna *Role* Mahasiswa

2) *Use Case* Diagram untuk Dosen Pengampu dan Pembimbing Akademik

Pada pengguna *role* dosen pengampu mata kuliah dan pembimbing akademik dapat melakukan 7 interaksi antara lain melakukan otentikasi dan otorisasi *user*, mengelola notifikasi, mengatur bahasa aplikasi, mengolah pemberitahuan mata kuliah, mengelola forum mata kuliah, menampilkan *manual book*, menampilkan FAQ, dan menampilkan *contact list*. *Use case* Diagram untuk pengguna *role*

Dosen Pengampu Mata Kuliah dan Pembimbing Akademik disajikan pada Gambar 37.



Gambar 35 *Use Case* Diagram Untuk Pengguna *Role* Dosen Pengampu Mata Kuliah dan Pembimbing Akademik

a. Metode Pengumpulan Data

Metode pengumpulan data yang digunakan pada penelitian ini adalah sebagai berikut.

1) Studi Literatur Studi literatur yang digunakan yaitu buku-buku, jurnal, prosiding dan internet yang menyajikan informasi tentang Web Service, REST, Django, Django REST Framwork, OAuth (Open Authorization) 1.0 dan 2.0, PostgreSQL, Android *Programming*.
2) Metode Interview Metode ini digunakan untuk mendapatkan informasi yang diperlukan dengan wawancara pihak-pihak terkait, yaitu tim pengelola SIAKAD Unila dan tim pengembang SIAKAD Univeristas Lampung.

b. Analisis Sistem

Kebutuhan pengembangan sistem informasi akademik Universitas Lampung ini adalah sebagai berikut.

- Data PKM Universitas Lampung sebelumnya yang mencangkup mahasiswa, mata kuliah, daftar kartu rencana studi, daftar kartu hasil studi, daftar pembayaran spp, daftar jadwal mata kuliah.
- Database yang dapat merekam data input sistem.
- IDE Android Studio untuk melakukan koding aplikasi Android.
- Virtual Server untuk membungkus paket-paket API Python yang dikembang melalui Django REST Framework (DRF).
- Menggunakan perangkat lunak seperti Web browser Mozilla Firefox, Sistem Operasi Windows 8 Enterprise 32-Bit.

c. Analisis *User Requirement*

Analisis kebutuhan aplikasi SIAKAD Unila adalah sebagai berikut.

- Aplikasi mampu menerapkan *web service*, otentikasi dan otorisasi pengguna dengan teknologi OAuth *2.0.*
- Aplikasi mampu melakukan *refresh token, registrasi* ID *device*, dan konfirmasi saat login aktif di perangkat lainnya.
- Aplikasi mampu menampilkan pengumuman dan *push notification* pengumuman masuk.
- Aplikasi mampu menampilkan jadwal kuliah dan *push notification* jadwal.
- Aplikasi mampu mengelola KRS (Kartu Rencana Studi) dan Nilai (termasuk Transkrip)

- Aplikasi mampu menampilkan ringkasan akademik mahasiswa.
- Aplikasi mampu menyimpan data *offline* pengguna.
- Aplikasi mampu merekomendasikan, menampilkan dan notifikasi komunikasi sosial antara mahasiswa dan dosen berdasarkan mata kuliah yang diambil oleh mahasiswa dan diampu oleh dosen.

4. *Analysis and Design Workflow*

Pada tahap ini akan dilakukan desain arsitektur, desain sistem, *Entity Relationship Diagram* (ERD), dan desain antarmuka. Perancangan atau desain sistem dalam penelitian ini menggunakan *Unified Modeling Language* (UML). Diagram-diagram yang dibuat dalam penelitian ini antara lain sebagai berikut.

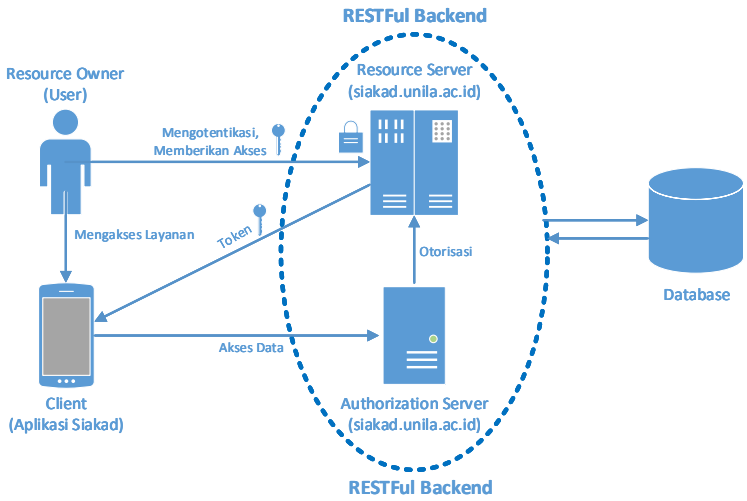
a. Desain Arsitektur

Teknologi yang digunakan dalam pembuatan sistem adalah teknologi *web service* dengan proses otentikasi menggunakan OAuth 2.0. *Web service* merupakan teknologi yang menyediakan integrasi proses dan data. Pada penelitain ini *web service* dibangun untuk menghubungkan aplikasi *mobile* Android sebagai *client* dengan terlebih dahulu melakukan proses otorisasi menggunakan *flow* protokol OAuth 2.0, seperti yang disajikan pada Gambar 35.

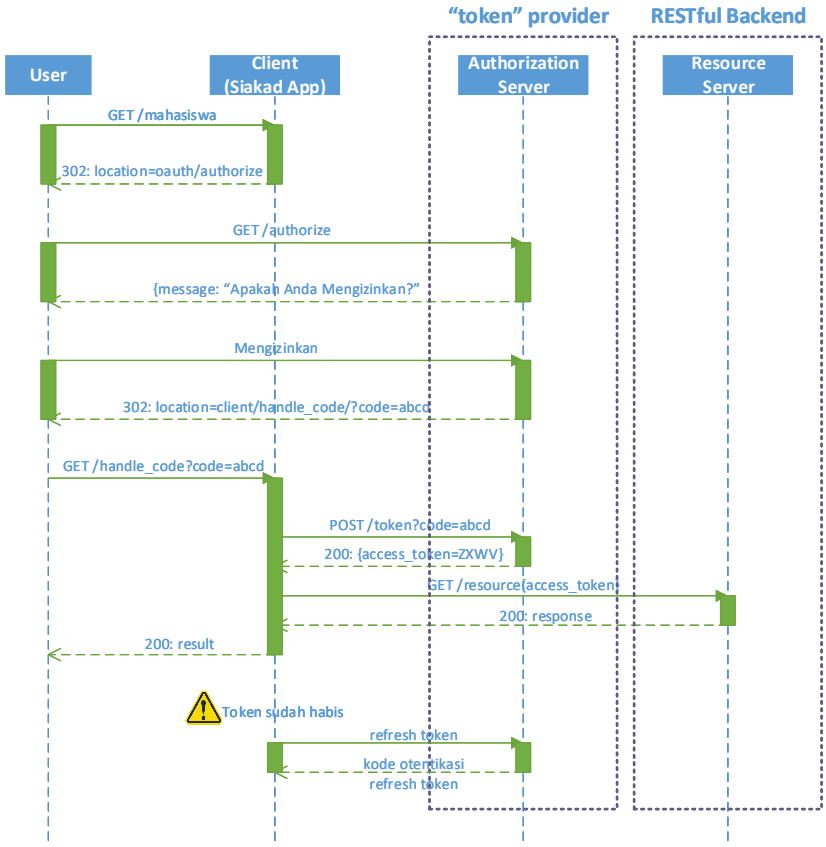
Web service dibangun dengan menggunakan RESTful *web service*. Dalam penelitain ini, RESTful yang dibangun menggunakan *framework* pada Django bernama DRF (*Django REST Framework*).

Dalam proses permintaan data (*request*), *client* terlebih dahulu memiliki token yang valid. Data yang didapat dari *web service* dikirim dalam format JSON.

Proses otentikasi pengguna dalam sistem yang dibangun memerlukan *grant* dari *resource server* (*unila.ac.id*) dengan terlebih dahulu melakukan proses otorisasi *user*. Pada proses ini membutuhkan izin dari *user* untuk melanjutkan otentikasi. Pada perancangan arsitektur ini, *resource server* dan *authorization server* menggunakan domain sama yaitu *unila.ac.id*. Desain arsitektur *web service* dapat dilihat pada Gambar 34.



Gambar 36 Perancangan Arisitektur *Web Service* dan *OAuth* 2.0



Gambar 37 Perancangan *Client Credentials Flow Web Service* dengan *OAuth 2.0*

Kode otorisasi untuk klien yang berinteraksi dapat menerima permintaan masuk dari *server* otorisasi (dapat bertindak sebagai *server* HTTP). Pengguna ingin mendapatkan respon data mahasiswa, melalui *client* pengguna terlebih dahulu melakukan proses otorisasi sebagai berikut.

- *Client* mengirimkan URL otorisasi dalam aplikasi. Dalam hal ini pengguna meng*input*kan *username* dan *password*.
- *Request* otorisasi diteruskan ke *Server* Otorisasi (*Authorization Server*), kemudian respon berupa pertanyaan "Apakah Anda mengizinkan?".

- Kemudian *Server* Otorisasi mengirimkan kode otorisasi ke *user*.
- *User* mengirimkan kode otorisasi ke *Server* Otorisasi untuk mendapatkan *access_token*.
- Respon yang diterima *Server* Otorisasi digunakan untuk mengambil data melalui *Resource Server*.
- *Resource Server* meneruskan respon ke *user* untuk mendapatkan data *mahasiswa.*

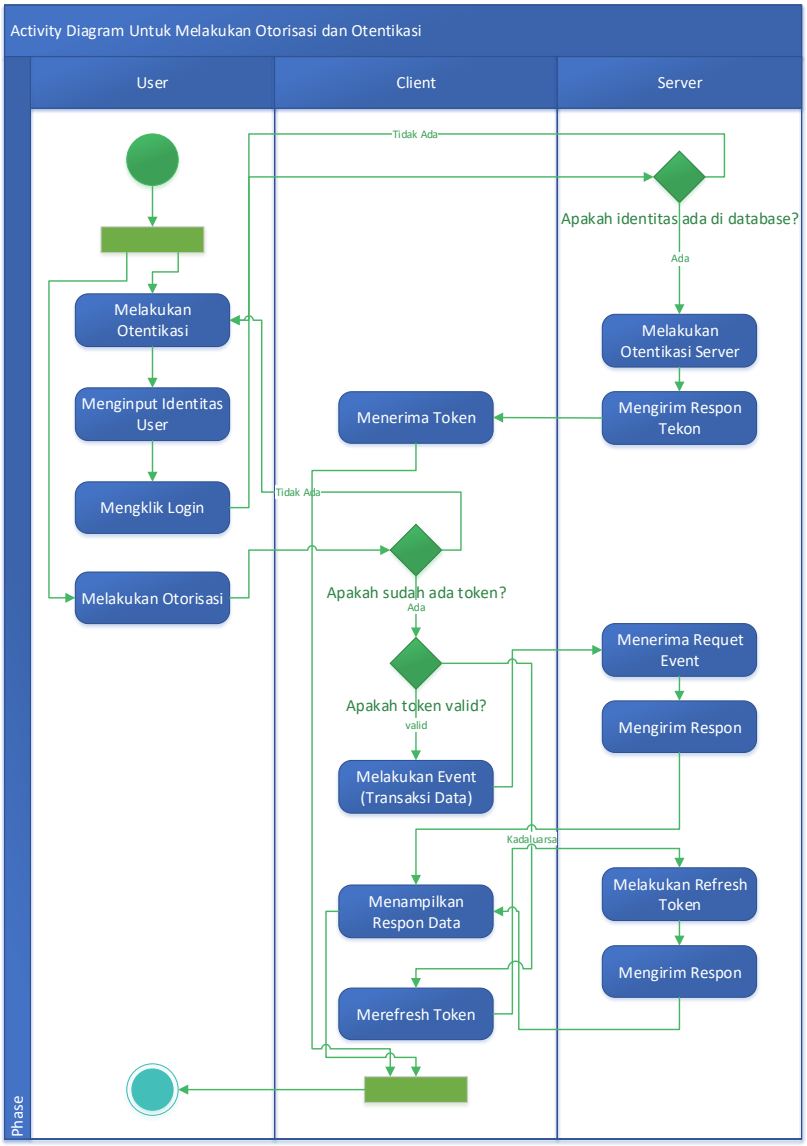
b. *Activity* Diagram

Activity diagram digunakan untuk menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktivitas yang dibentuk dalam satu operasi sehingga dapat juga untuk aktivitas lainnya. Diagram ini sangat mirip dengan *flowchart* karena memodelkan *workflow* dari satu aktivitas ke aktivitas lainnya atau dari aktivitas ke status. Pada aplikasi *Sistem Informasi Akademik Berbasis Mobile* terdapat 17 *activity* diagram yaitu sebagai berikut.

1) *Activity* Diagram Melakukan Otentikasi dan Otorisasi *User*

   Activity diagram dalam melakukan otentikasi dan otorisasi *user* dimulai dengan terlebih dahulu *user* melakukan otentikasi dalam aplikasi, *user* menginputkan data identikasi berupa *username* dan password, lalu *user* menekan tombol *login* pada aplikasi, kemudian *server* akan mengecek data identitas terhadap *database*, jika data identitas ditemukan maka *server* akan melakukan proses otentikasi *server* (*Server Authorization*) dengan menghasilkan respon token.

   Token ini yang akan digunakan dalam melakukan otorisasi dalam melakukan *request* data (data transaksi dengan *database*), hasil dari pada proses otorisasi adalah user dapat menerima respon. Apabila token sudah kadaluwarsa maka sistem akan melakukan *refresh* token ke *server*. *Activity* diagram melakukan otentikasi dan otorisasi dapat disajikan pada Gambar 38.

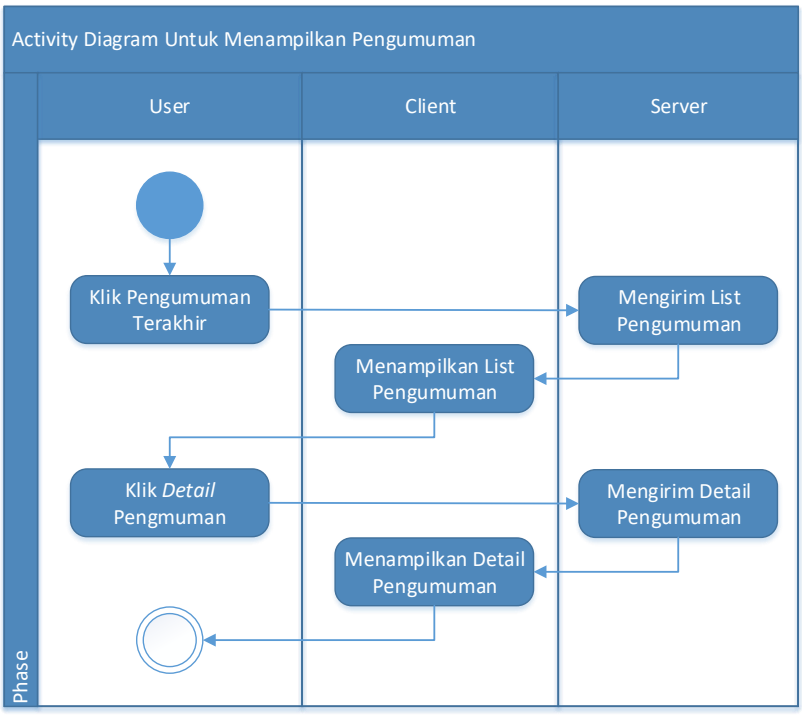


Gambar 38 *Activity* Diagram Melakukan Otentikasi dan Otorisasi *User*

2) *Activity* Diagram Menampilkan Pengumuman

Activity diagram menampilkan pengumuman dimulai dengan mengklik pengumuman terakhir yang ada dalam Halaman *Home*. Kemudian *server* akan mengirimkan *list* pengumuman, lalu *client* menampilkan *list* pengumuman. Ketika *user* mengklik detail *item* pengumuman maka *server* akan mencari *id*

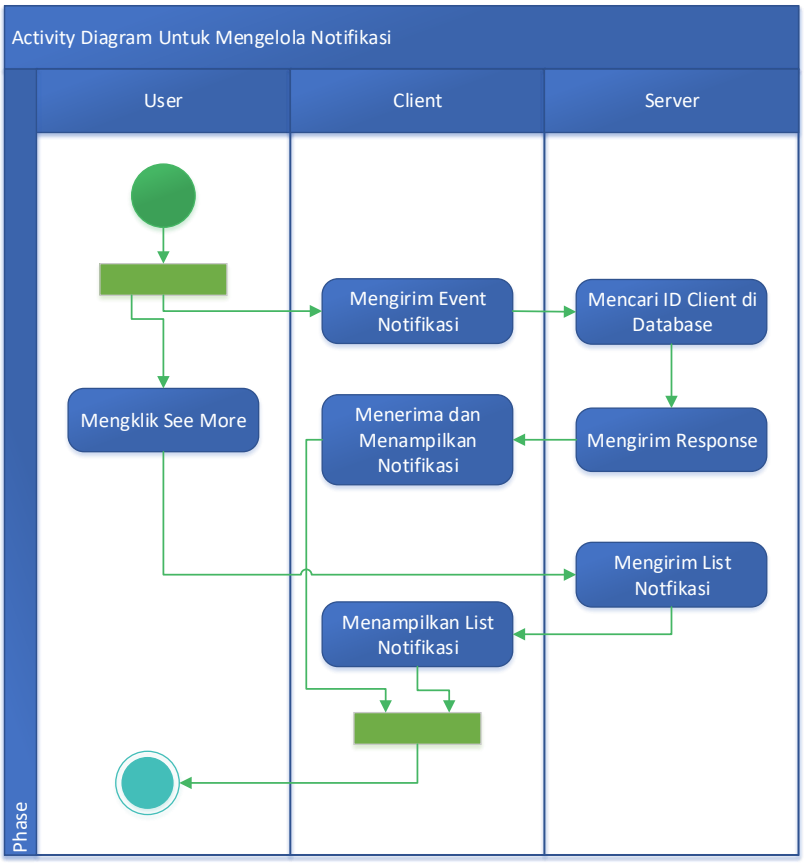
pengumuman, lalu *client* akan menampilkan detail pengumuman. *Activity* diagram untuk menampilkan pengumuman disajikan pada Gambar 39.



Gambar 39 *Activity* Diagram Menampilkan Pengumuman

3) *Activity* Diagram Mengelola Notifikasi

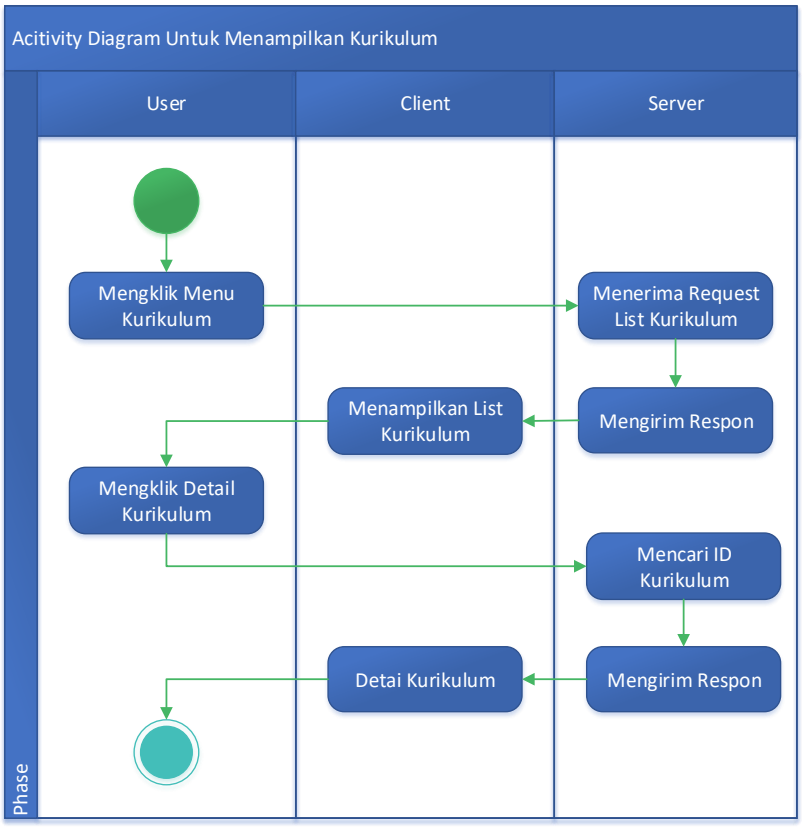
Activity diagram untuk mengelola notifikasi dimulai dengan *client* mengirimkan *event* notifikasi, lalu *server* mencari *id client* dalam *database* dan menghasilkan respon yang akan ditampilkan di *client. Client* juga dapat melihat semua *list* notifikasi dengan mengklik *button see more*. *Activity* diagram mengelola notifikasi disajikan pada Gambar 40.



Gambar 40 *Activity* Diagram Mengelola Notifikasi

4) *Activity* Diagram Menampilkan Kurikulum

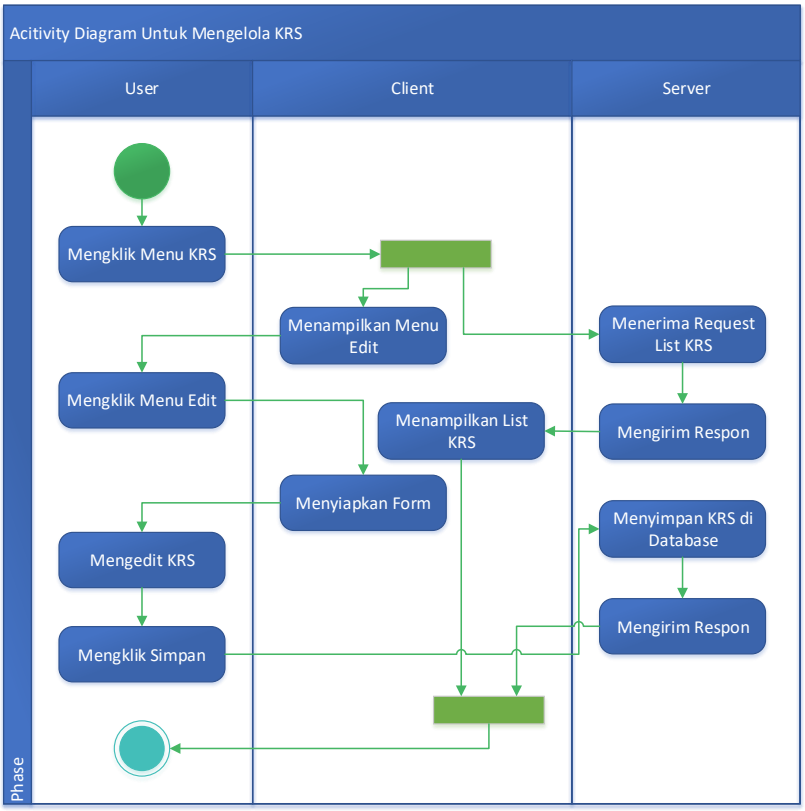
Activity diagram menampilkan kurikulum dimulai degan mengklik menu Kurikulum yang ada di Halaman Akademis, lalu *server* menerima *request list* kurikulum dengan mengirimkan respon untuk menampilkan *list* kurikulum pada *client*. Kemudian *user* mengklik *item list*, lalu *server* mencari *id* kurikulum dan mengirimkan respon, lalu *client* melihat detail dari kurikulum. *Activity* diagram menampilkan kurikulum disajikan pada Gambar 41.



Gambar 41 *Activity* Diagram Menampilkan Kurikulum

5) *Activity* Diagram Mengelola KRS

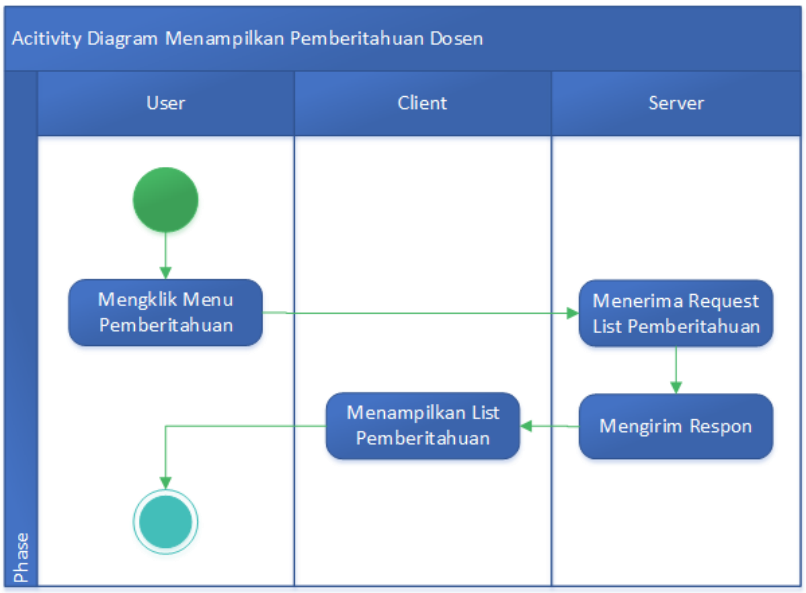
Activity diagram menampilkan kurikulum dimulai dengan mengklik menu KRS yang ada di Halaman Akademis, lalu *server* menerima *request list* KRS dengan menghasilkan respon yang akan diterima oleh *client* untuk menampilkan *list* KRS. *User* mengklik menu *edit*, lalu *client* menyiapkan *form* untuk meng*edit* kemudian *user* mengisi bagian yang akan diperbarui. Kemudian klik simpan untuk menyimpan hasil meng*edit* ke *server* dan *server* mengirimkan respon ke *client*. *Activity* diagram menampilkan kurikulum disajikan pada Gambar 42.



Gambar 42 *Activity* Diagram Mengelola KRS

6) *Activity* Diagram Menampilkan Pemberitahuan

Activity diagram menampilkan pemberitahuan dimulai dengan *user* mengklik *button* menu Pemberitahuan yang ada di Halaman Pesan, lalu *server* menerima *request* untuk menampilkan pemberitahuan dengan mengirimkan respon ke *client* untuk menampilkan *list* pemberitahuan. *Activity* diagram menampilkan pemberitahuan disajikan pada Gambar 43.

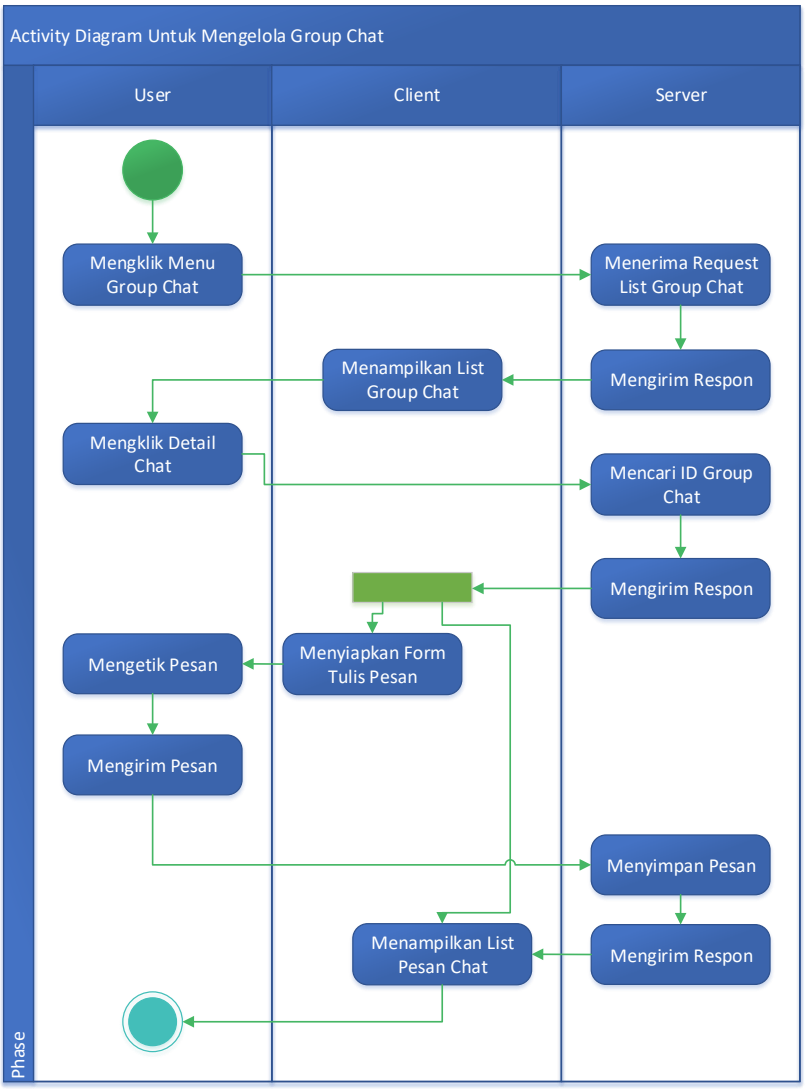


Gambar 43 *Activity* Diagram Menampilkan Pemberitahuan

7) *Activity* Diagram Mengelola Forum

Activity diagram mengelola forum dilakukan dengan mengklik *button* Forum yang ada di Halaman Pesan, lalu *server* menerima *request list* forum dan menghasilkan respon yang ditampilkan pada *client.* Kemudian *user* melakukan detail dari *chat,* lalu *server* mencari *id group* dan mengirim kan respon ke *client* untuk menampilkan *list chat.*

Kemudian *user* melakukan mengirim pesan dengan mengetik pesan pada *box* yang telah disediakan oleh *client*, lalu menekan *button* kirim untuk meneruskan pada *server* dan *server* akan mengirimkan pesan sudah terkirim. *Activity* diagram mengelola forum disajikan pada Gambar 44.

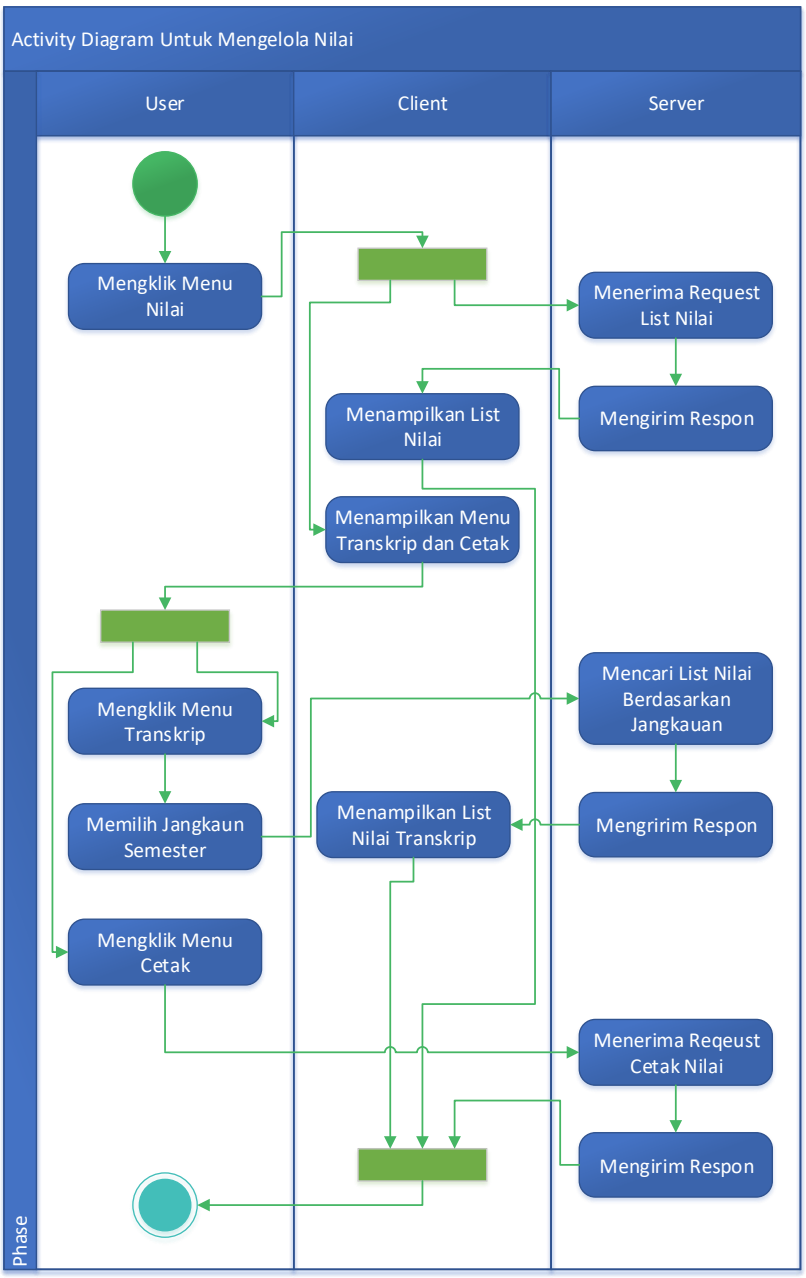


Gambar 44 *Activity* Diagram Mengelola Forum

8) *Activity* Diagram Melihat Nilai

Activity diagram melihat nilai dimulai dengan mengklik *button* menu Nilai yang terdapat pada Halaman Akademis, lalu *server* akan menerima *request* untuk menampilkan *list* nilai yang akan ditampilkan pada *client. User* mengklik menu transkrip, lalu memilih jangkauan semester, lalu *server* akan mencari *list* nilai berdasarkan jangkauan dengan mengirimkan respon *list* nilai transkrip. *User* juga dapa mencetap nilai dengan mengklik

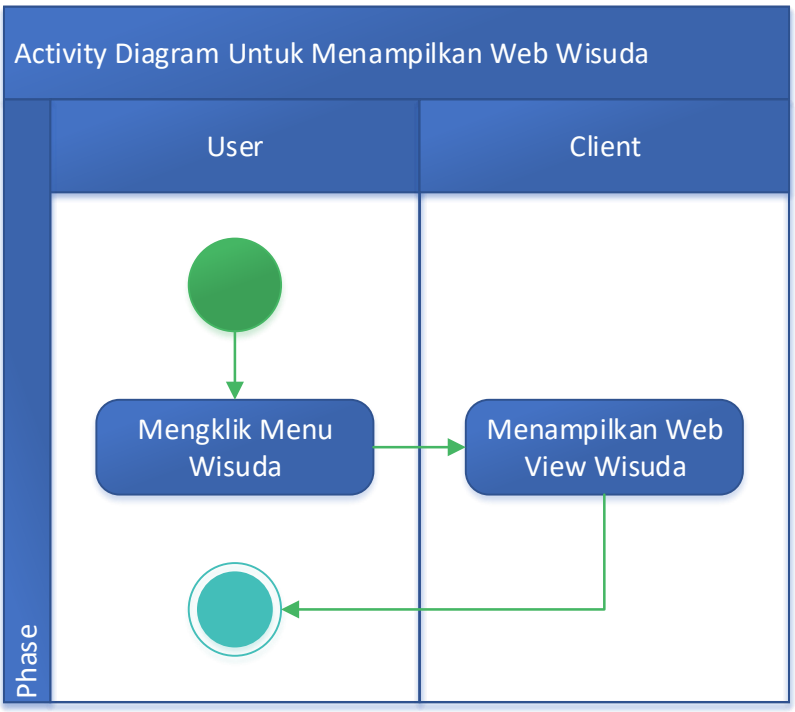
button cetak pada Halaman *List* Nilai, lalu *server* mencari *id* semester dengan menghasilkan respon untuk di*donwnload* oleh *user*. *Activity* diagram melihat nilai disajikan pada Gambar 45.



Gambar 45 *Activity* Diagram Melihat Nilai

9) *Activity* Diagram Menampilkan *Web* Wisuda

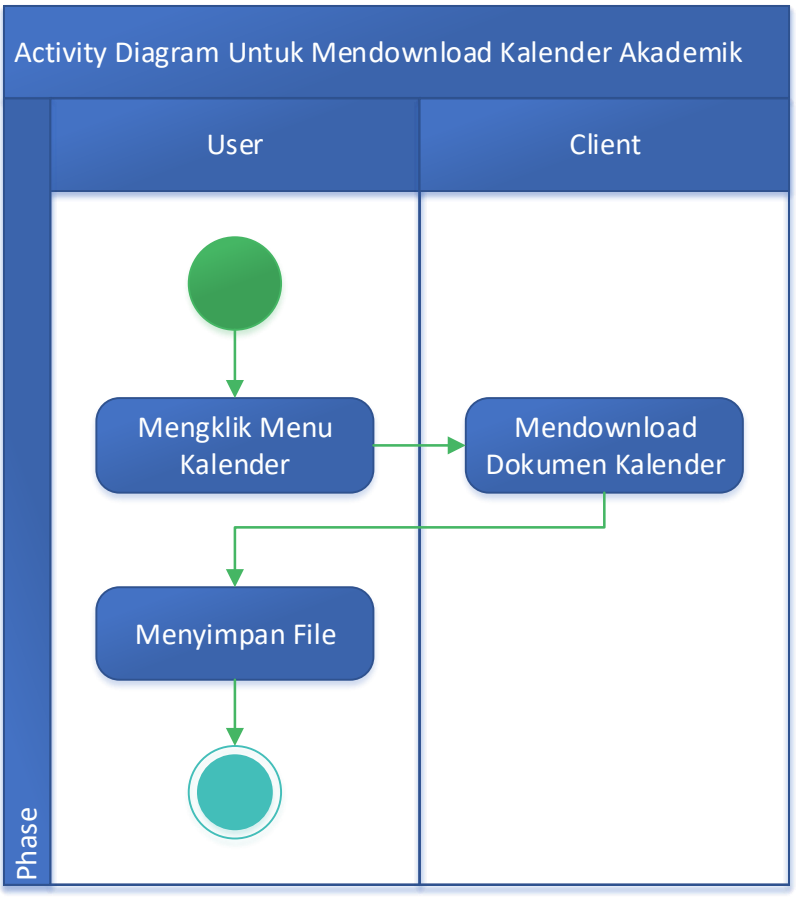
Activity diagram menampilkan *web* wisuda dimulai dengan mengklik *button* menu *Wisuda* yang ada pada Halaman *Home*, lalu *client* menampilkan *webview* wisuda. *Activity* diagram menampilkan wisuda disajikan pada Gambar 46.



Gambar 46 *Activity* Diagram Menampilkan *Web* Wisuda

10) *Activity* Diagram Mengunduh Kalender Akademik

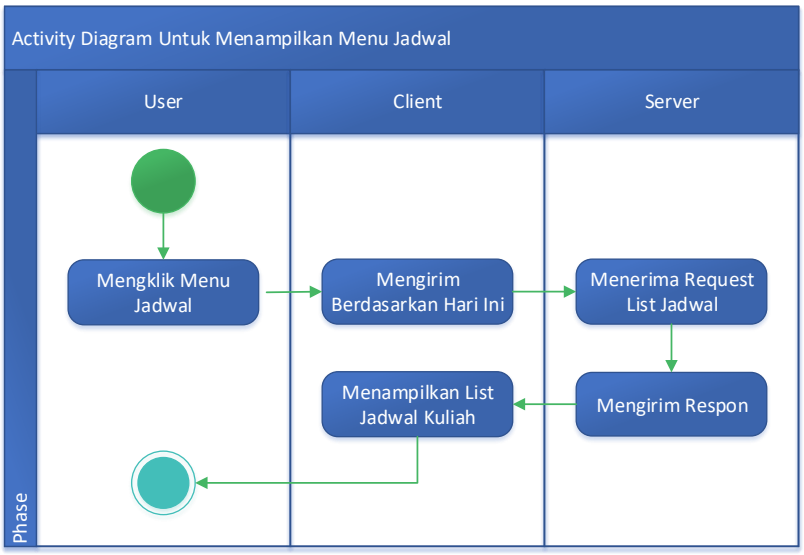
Activity diagram mengunduh kalender akademik dimulai dengan mengklik menu Kalender Akademik yang ada di Halaman *Home*, lalu *client* men*download* dokumen kalender akademik, lalu *user* menyimpan pada *file manager smartphone*. *Activity* diagram men*download file* kalender akademik disajikan pada Gambar 47.



Gambar 47 *Activity* Diagram Mengunduh Kalender Akademik

11) *Activity* Diagram Menampilkan Jadwal Kuliah

Activity diagram menampilkan jadwal kuliah dimulai dengan mengklik *button* menu Jadwal Kuliah pada Halaman *Home*, lalu *client* mengirimkan waktu, lalu *server* menerima *request list* jadwal dengan mengirimkan respon ke *client* untuk menampilkan *list* jadwal kuliah. *Activity* diagram menampilkan jadwal kuliah disajikan pada Gambar 48.

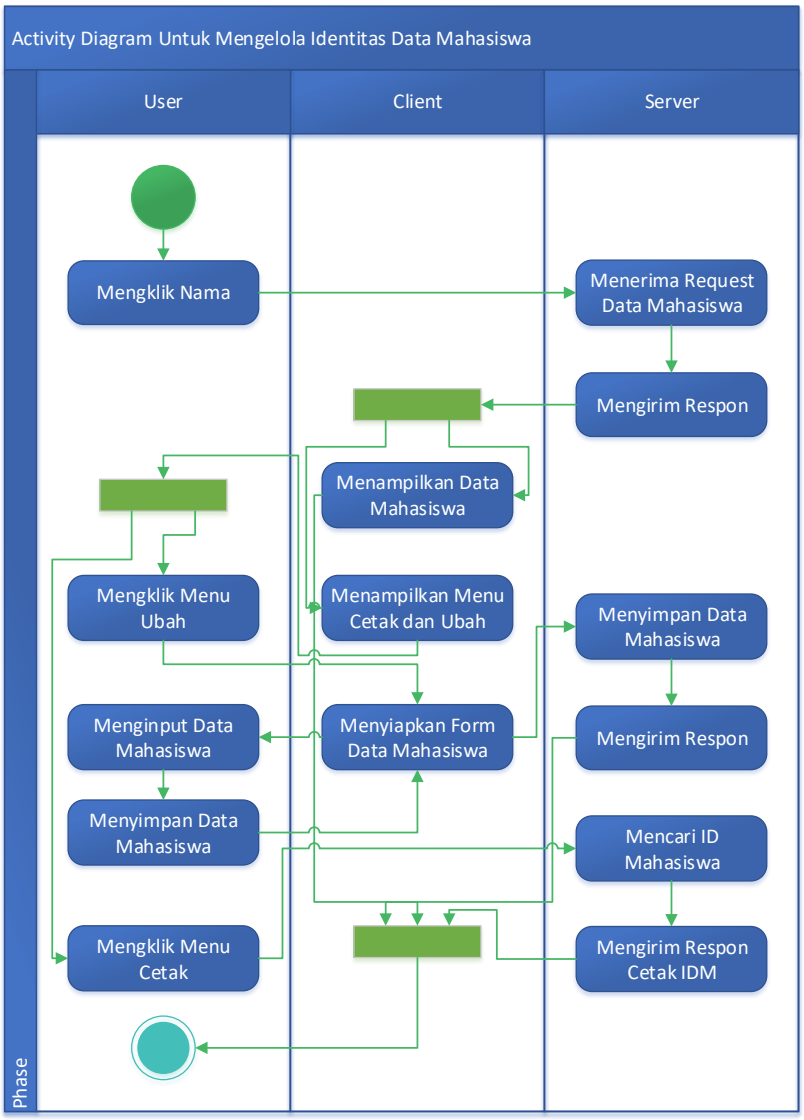


Gambar 48 *Activity* Diagram Menampilkan Jadwal Kuliah

12) *Activity* Diagram Mengelola Identitas Data Mahasiswa

Activity diagram mengelola identitas data mahasiswa dimulai dengan mengklik *link* Nama Mahasiswa pada Halaman Profil, lalu *server* menerima *request* data mahasiswa untuk ditampilkan pada *client.*

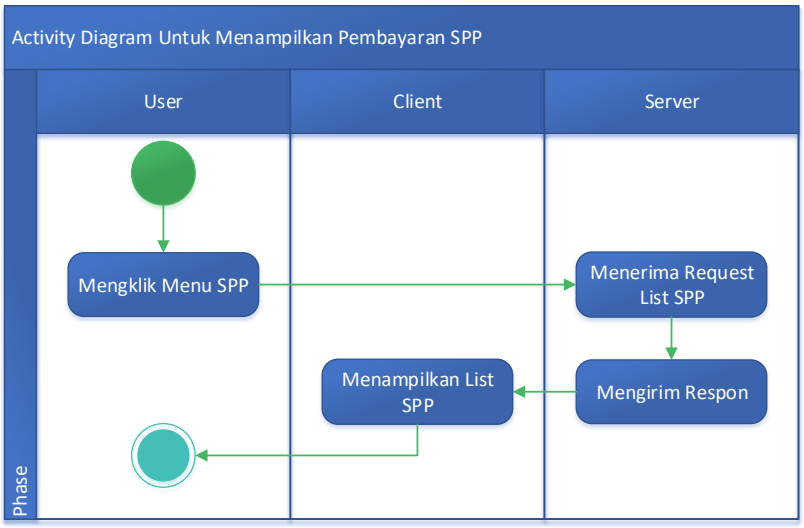
User dapat memperbarui Identitas Data Mahasiswa dengan mengklik menu *edit* dan mengisi *form* yang disediakan oleh *client*, lalu mengklik *update* untuk data disimpan oleh *server*. Selain itu *user* juga dapat mencetak Identitas Data Mahasiswa dengan mengklik menu cetak, lalu *server* akan mencari *id* mahasiswa dengan mengirimkan respon cetak pada *client. Activity* diagram mengelola identitas data mahasiswa disajikan pada Gambar 49.



Gambar 49 *Activity* Diagram Mengeloa Identitas Data Mahasiswa

13) *Activity* Diagram Menampilkan Pembayaran SPP

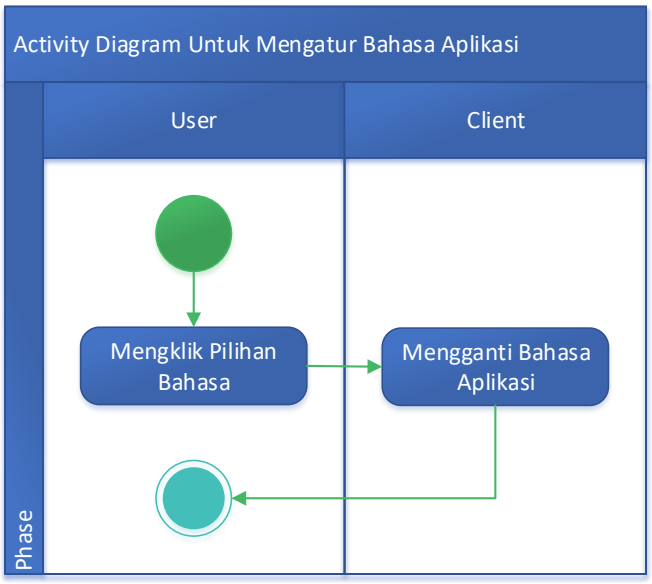
Activity diagram menampilkan pembayaran SPP dimulai dengan mengklik *button* menu SPP pada Halaman Akademis, lalu *server* menerima *request list* SPP dengan mengirimkan respon ke *client* untuk menampilkan *list* pembayaran SPP. *Activity* diagram menampilkan pembayaran SPP disajikan pada Gambar 50.



Gambar 50 *Activity* Diagram Menampilkan Pembayaran SPP

14) *Activity* Diagram Mengatur Bahasa Aplikasi

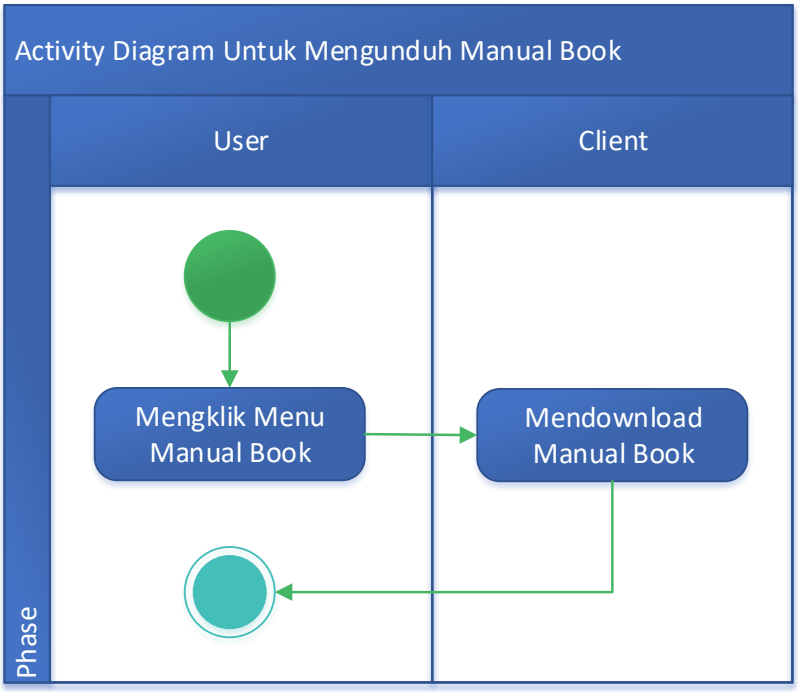
Activity diagram mengatur bahasa aplikasi dimulai dengan mengklik tombol Bahasa pada Halaman *Home*, lalu *client* akan mengganti bahasa aplikasi. *Activity* diagram mengatur bahasa aplikasi disajikan pada Gambar 51.



Gambar 51 *Activity* Diagram Mengatur Bahasa Aplikasi

15) *Activity* Diagram Mengunduh *Manual Book*

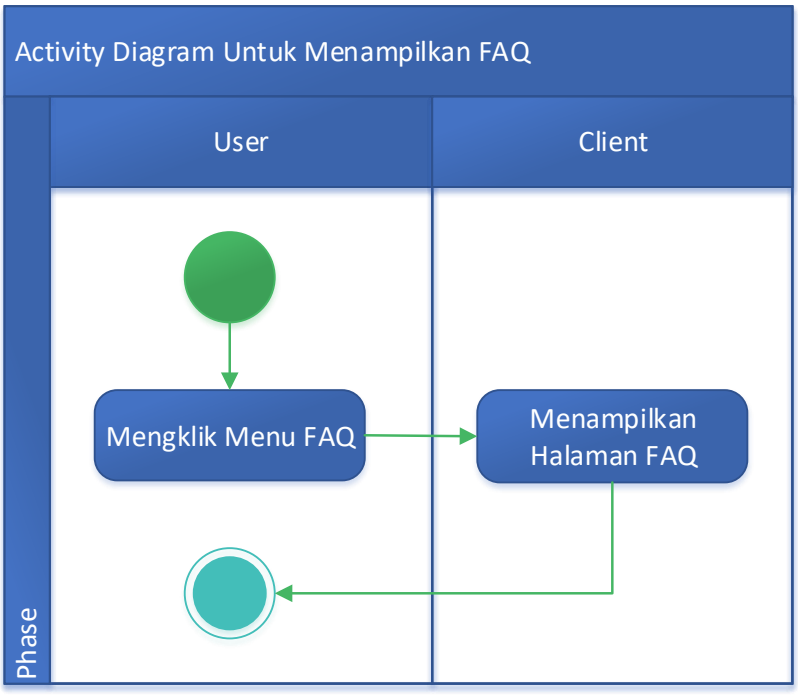
Activity diagram mengunduh *manual book* dimulai dengan mengklik *popup* pada *action bar* pada Halaman *Home*, lalu *client* akan mengunduh *manual book*. *Activity* diagram mengunduh *manual book* disajikan pada Gambar 52.



Gambar 52 *Activity* Diagram Mengunduh *Manual Book*

16) *Activity* Diagram Menampilkan FAQ

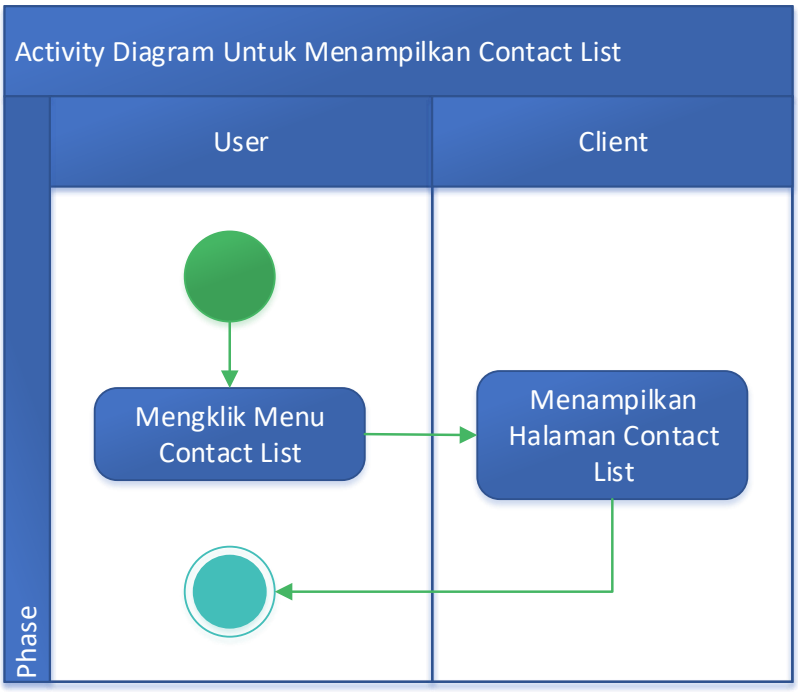
Activity diagram menampilkan FAQ dimulai dengan mengklik *popup* pada *action bar* pada Halaman *Home*, lalu *client* akan menampilkan FAQ. *Activity* diagram menampilkan FAQ disajikan pada Gambar 53.



Gambar 53 *Activity* Diagram Menampilkan FAQ

17) *Activity* Diagram Menampilkan *Contact List*

Activity diagram menampilkan *Contact List* dimulai dengan mengklik *popup* pada *action bar* pada Halaman *Home*, lalu *client* akan menampilkan *Contact List*. *Activity* diagram menampilkan *Contact List* disajikan pada Gambar 54.



Gambar 54 *Activity* Diagram Menampilakan *Contact List*

c. *Class* Diagram

Class diagram digunakan untuk mendeskripsikan jenis – jenis obyek dalam sistem dan berbagai macam hubungan statis yang terjadi. Pada aplikasi "Sistem Akademik Berbasis *Mobile*" terdapat 5 *package*, 23 *class*, dan 1 *interface*.

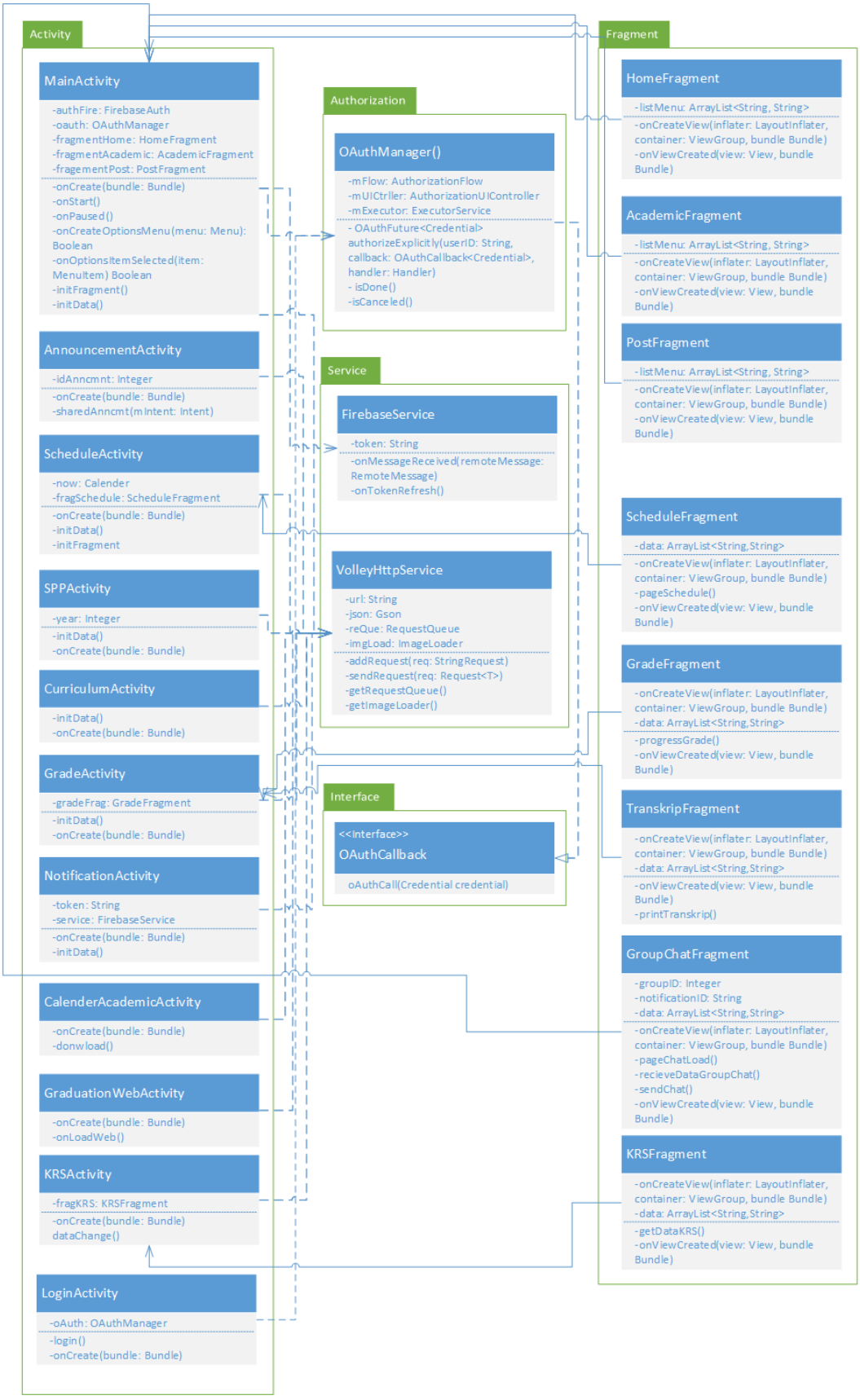
Kelima *package* yang dibangun diantaranya adalah *Activity Package* merupakan paket yang berisi *class – class activity* yang meng*extends class Activity*, pada paket ini dibangun *Activity* yang akan menampung seluruh aktifitas dari setiap fitur dari *client.* Paket ini yang terdapat 11 *class* diataranaya *MainActivity, AnnouncementActivity, ScheduleActivity, SPPActivity, KRSActivity* ,

CurriculumActivity, GradeActivity, NotificationActivity, LoginActivity, CalenderAcademicActivity, dan *GraduationActivity*.

Fragment Package merupakan paket yang menampung *class* yang meng*extends class Fragment*. Dimana *class Fragment* memiliki alir hidup hampir mirip dengan *class Activity*. Paket ini memiliki 8 *class* diantaranya *HomeFragment, AcademicFragment, PostAcademic, ScheduleAcademic, GradeFragment, TranskripFragment, GroupChatFragment,* dan *KRSFragment*.

Authentication Package merupakan paket yang menangani tentang proses otentikasi dan otorisasi yang merupakan rujukan bagi kelas *Activity* dan *Fragment* atau lainnya dalam menerapkan proses otorisasi. Paket *Authentication* terdapat 1 *class* yaitu OAuth*Manager.*

Service Package merupakan paket yang menangani tentang *service* yang diberikan aplikasi kepada *user*, *service* ini memungkinkan aplikasi berinteraksi dengan sistem lain. Paket *Service* terdapat 2 *class* diantaranya adalah *FirebaseService* dan *VolleyHttpService*. *Interface Package* terdapat 1 *interface* yaitu OAuth*Callback*. *Class* diagram yang ada di *client* (Android) disajikan pada Gambar 55.



Gambar 55 *Class* Diagram Pada *Client* (Android)

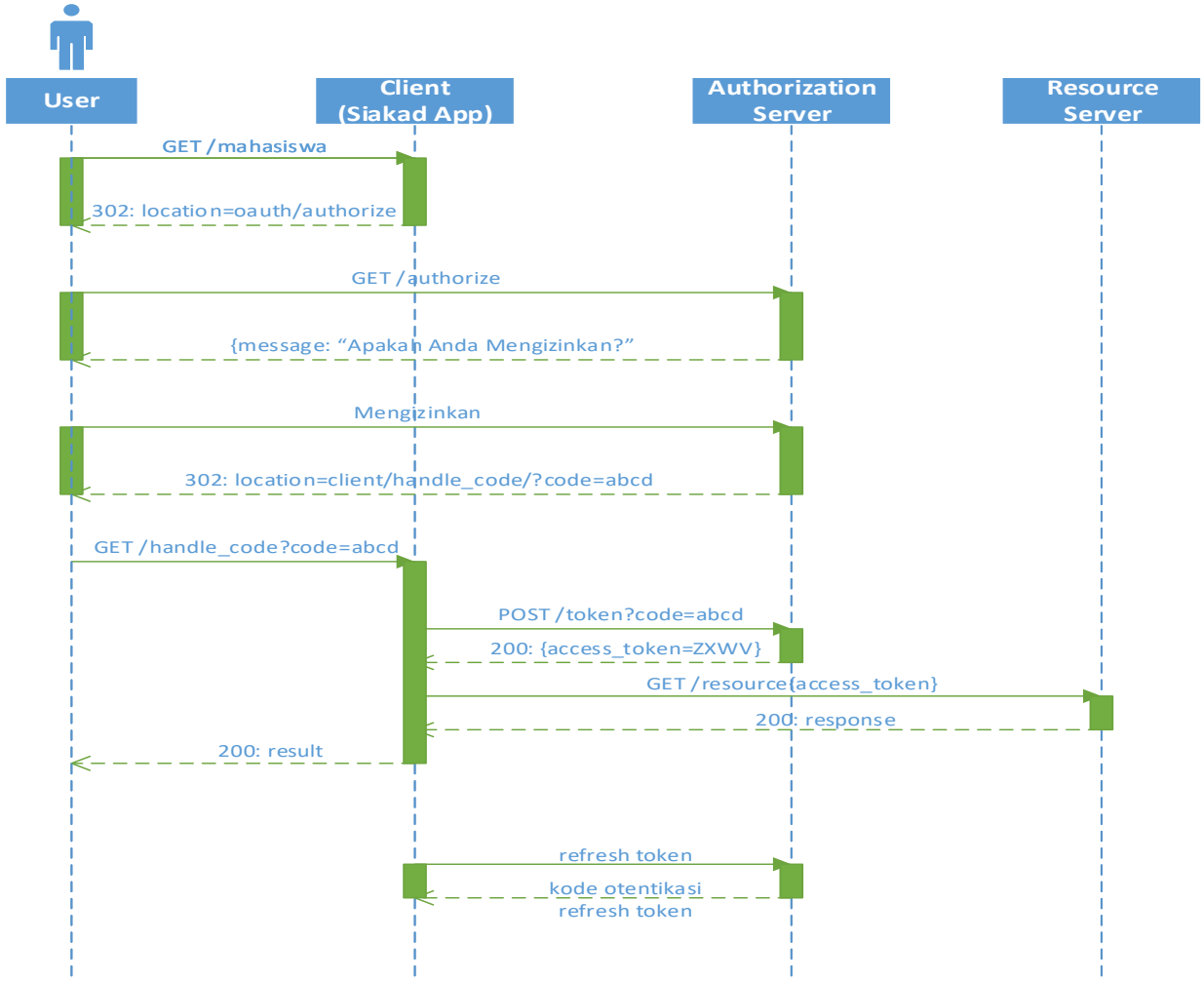
d. *Sequence* Diagram

Sequence diagram digunakan untuk menggambarkan interaksi antara sejumlah objek dalam urutan waktu. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara objek juga interaksi antar objek yang terjadi pada titik tertentu dalam eksekusi sistem. Pada aplikasi "Sistem Akademik Berbasis *Mobile*" terdapat 13 *sequence* diagram yang akan dibuat yaitu sebagai berikut.

1) *Sequence* Diagram Melakukan Otorisasi dan Otentikasi

   Untuk dapat mendapatkan respon dari setiap transaksi aliran data, maka *user* harus melakukan proses otentikasi. Proses otentikasi ini dimulai dengan *user* melakukan persetujuan setelah meng*inputkan* identitas pribadi ke *server*. Respon balik yang didapatkan oleh *user* adalah meneruskan ke *client* (Aplikasi) untuk melakukan proses otorisasi. *Client* akan mengirimkan kode akses ke *Authorization Server* untuk mendapatkan akses token.

   Ketika akses token sudah kadaluwarsa, maka *Client* akan me*refresh Authorization Server* untuk mendapatkan akses token kembali. Untuk lebih jelasnya *sequence* diagram untuk melakukan otentikasi dan otorisasi disajikan pada Gambar 56.

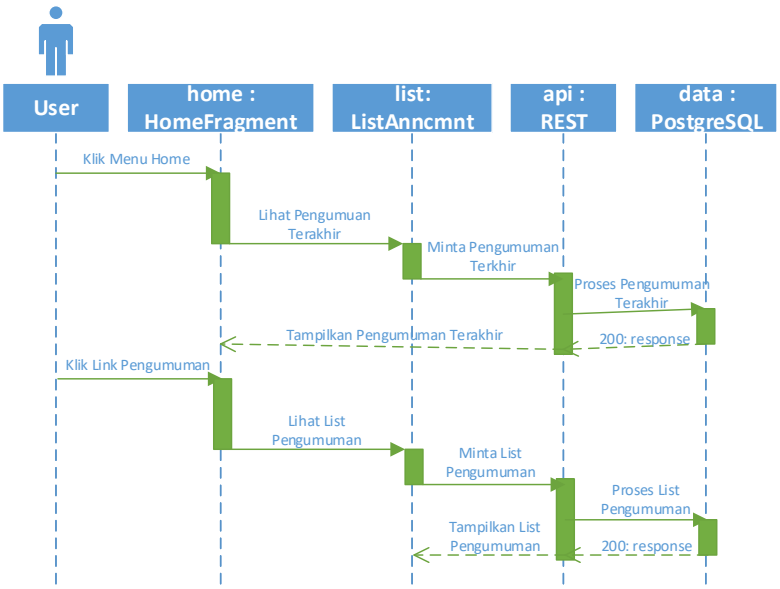


Gambar 56 *Sequence* Diagram Melakukan Otentikasi dan Otorisasi

2) *Sequence* Diagram Menampilkan Pengumuman

Untuk menampilkan pengumuman *user* harus mengklik menu *home* yang ada di *tab layout* Halaman *Home*, maka *client* akan meminta pengumuman terakhir ke *server* melalui API untuk menampilkan pengumuman terakhir dari *database*.

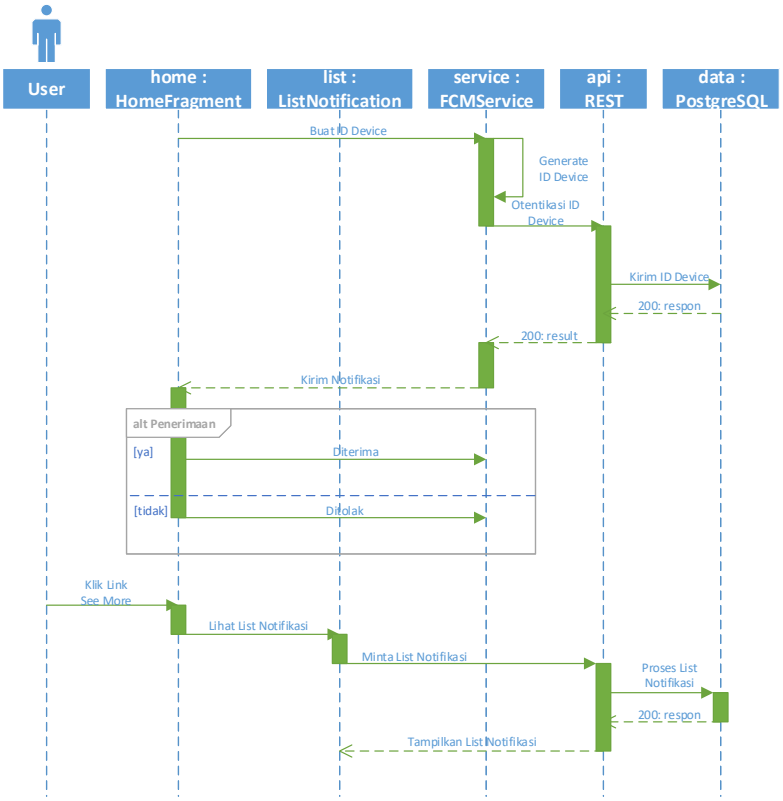
User dapat melihat *list* pengumuman dengan mengklik *link* pengumuman yang terdapat di Halaman *Home* aplikasi, maka *client* meminta *list* pengumuman ke *server* melalui API untuk menampilkan *list* pengumuman dari *database*, lalu menampilkan *list* pengumuman itu. Untuk lebih jelasnya *sequence* diagram untuk menampilkan pengumuman disajikan pada Gambar 57.



Gambar 57 *Sequence* Diagram Menampilkan Pengumuman

4) *Sequence* Diagram Mengelola Notifikasi

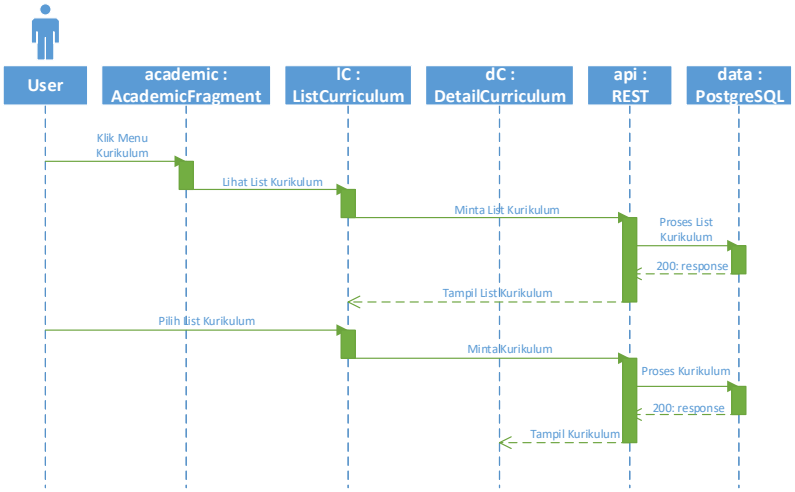
Untuk mengelola notifikasi *user* harus membuka aplikasi untuk mendaftarkan ID *Device* ke *database* dengan menghasilkan ID mengguna *Service* dari Firebase, lalu ID akan didaftarkan melalui API. Kemudian *Service* Firebase akan mengirimkan notifikasi ke *client,* jika notifikasi diterima oleh *client* maka notifikasi akan muncul, sebaliknya notifikasi tidak ditampilkan. *User* juga dapat melihat *list* notifikasi dengan mengklik *link see more* yang terdapat pada Halaman *Home,* maka *client* meminta *list* pengumuman ke *server* melalui API untuk menampilkan *list* notifikasi dari *database*, lalu menampilkan *list* notifikasi itu. Untuk lebih jelasnya *sequence* diagram untuk mengelola notifikasi disajikan pada Gambar 58.



Gambar 58 *Sequence* Diagram Mengelola Notifikasi

5) *Sequence* Diagram Menampilkan Kurikulum

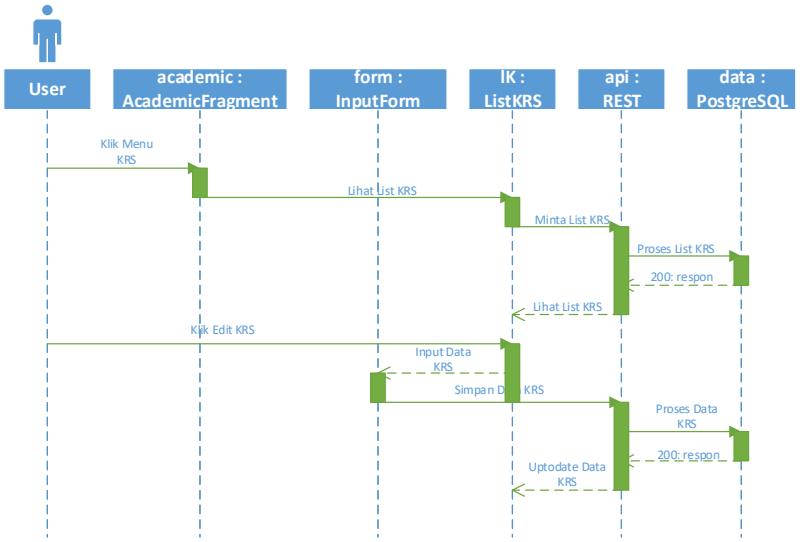
Untuk menampilkan kurikulum *user* harus mengklik menu Kurikulum yang ada di Halaman *Home*, maka *client* akan meminta *list* kurikulum ke *server* melalui API untuk menampilkan *list* kurikulum dari *database*. User dapat melihat detail kurikulum dengan mengklik salah satu kurikulum yang pada *list* kurikulum, maka sistem meminta detail kurikulum ke *server* melalui API untuk menampilkan detail kurikulum dari *database*, lalu menampilkan detail kurikulum itu. Untuk lebih jelasnya *sequence* diagram untuk menampilkan kurikulum disajikan pada Gambar 59.



Gambar 59 *Sequence* Diagram Menampilkan Kurikulum

6) *Sequence* Diagram Mengelola KRS

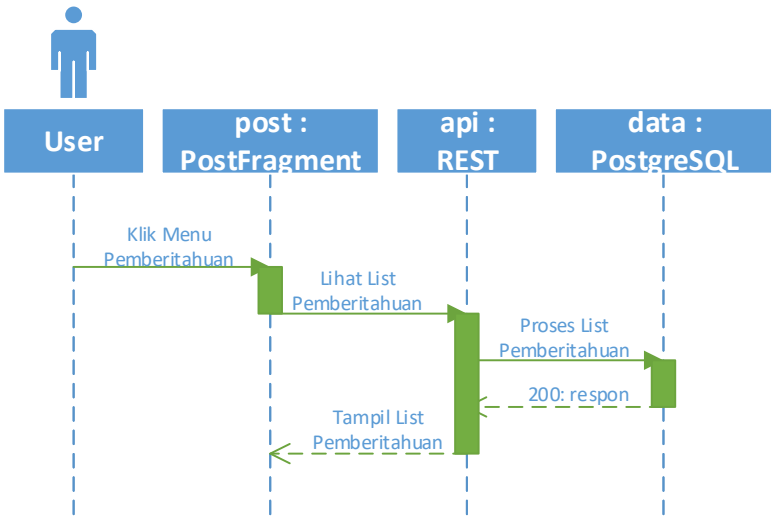
Untuk mengelola KRS *user* harus mengklik menu KRS yang terdapat di Halaman Akademik, maka *client* akan meminta *list* KRS ke *server* melalui API untuk menampilkan *list* KRS dari *database*. *User* dapat memperbarui KRS dengan mengklik tombol *edit*, sehingga *client* akan menampilkan *form input* KRS dan *user* meng*input*kan KRS yang akan di *edit* pada *form* yang tersedia dan menyimpan KRS tersebut ke dalam *database* melalui API. Untuk lebih jelasnya *sequence* diagram mengelola KRS disajikan pada Gambar 60.



Gambar 60 *Sequence* Diagram Mengelola KRS

7) *Sequence* Diagram Menampilkan Pemberitahuan Dosen

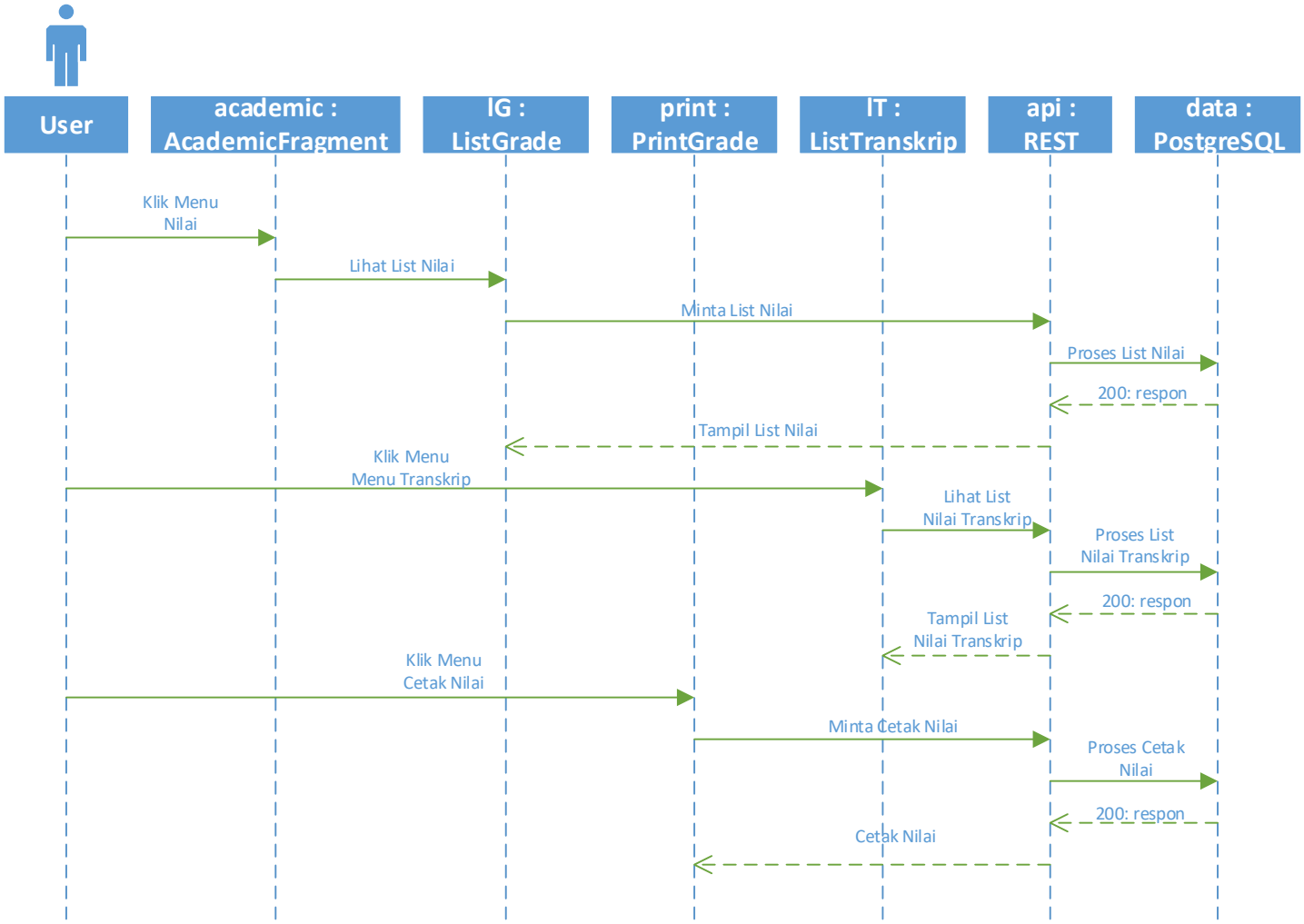
Untuk menampilkan pemberitahuan dosen *user* harus mengklik menu Pemberitahuan yang terdapat pada Halaman Pesan, maka *client* akan meminta *list* pemberitahuan ke *server* melalui API untuk menampilkan *list* pemberitahuan dari *database*. Untuk lebih jelasnya *sequence* diagram untuk menampilkan pemberitahuan dosen disajikan pada Gambar 61.



Gambar 61 *Sequence* Diagram Menampilkan Pemberitahuan Dosen

8) *Sequence* Diagram Melihat Nilai

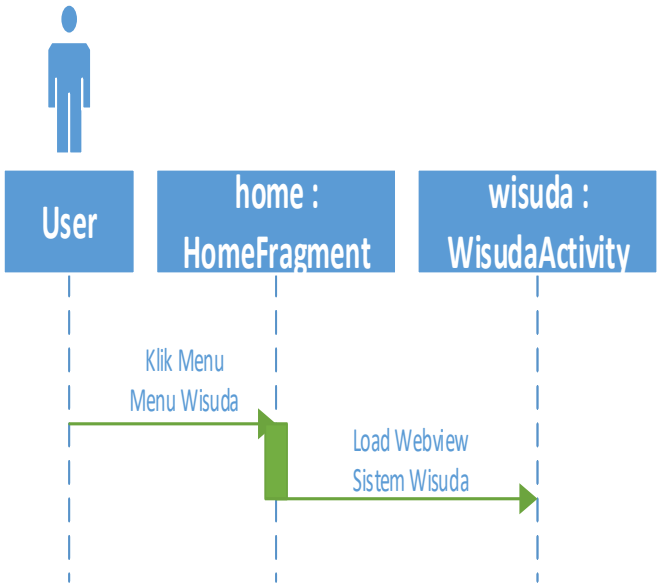
Untuk melihat nilai *user* harus mengklik menu Nilai yang terdapat pada Halaman Akademik, maka *client* akan meminta *list* nilai ke *server* melalui API untuk menampilkan *list* nilai dari *database*. User dapat melihat nilai transkrip dengan mengklik menu Transkrip yang terdapat pada Halaman Nilai, maka *client* akan meminta *list* nilai berdasarkan jangkauan semester ke *server* melalui API untuk menampilkan *list* nilai dari *database. User* dapat mencetak nilai dengan mengklik menu *Print* yang terdapat pada Halaman Nilai, maka *client* akan meminta *server* melalui API untuk mencetak nilai berdasarkan semester. Untuk lebih jelasnya *sequence* diagram melihat nilai disajikan pada Gambar 62.



Gambar 62 *Sequence* Diagram Melihat Nilai

9) *Sequence* Diagram Menampilkan *Web* Wisuda

Untuk menampilkan *web* wisuda *user* harus mengklik menu Wisuda yang terdapat di Halaman *Home*, maka *client* akan memuat halaman *web* wisuda untuk ditampilkan pada Halaman Wisuda. Untuk lebih jelasnya *sequence* diagram disajikan pada Gambar 63.

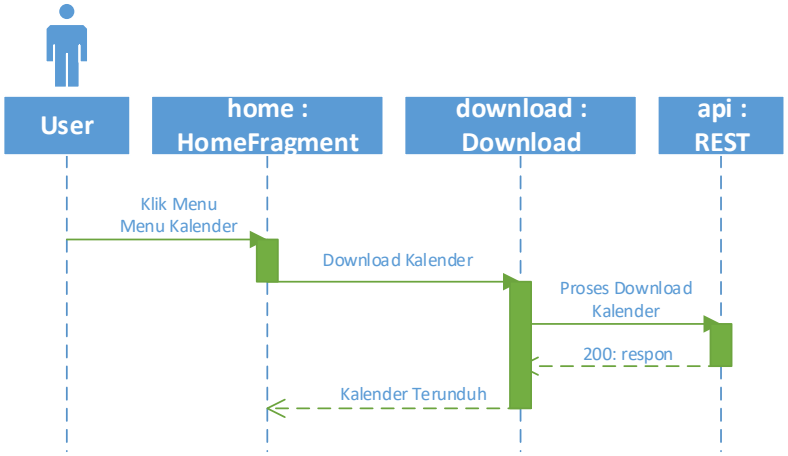


Gambar 63 *Sequence* Diagram Menampilkan *Web* Wisuda

10) *Sequence* Diagram Mengunduh Kalender Akademik

Untuk mengunduh kalender akademik *user* harus mengklik menu Kalender Akademik yang terdapat pada Halaman *Home*, maka *client* meminta *download* kalender akademik ke *server* melalui API untuk disimpan dalam *file manager*. Untuk lebih

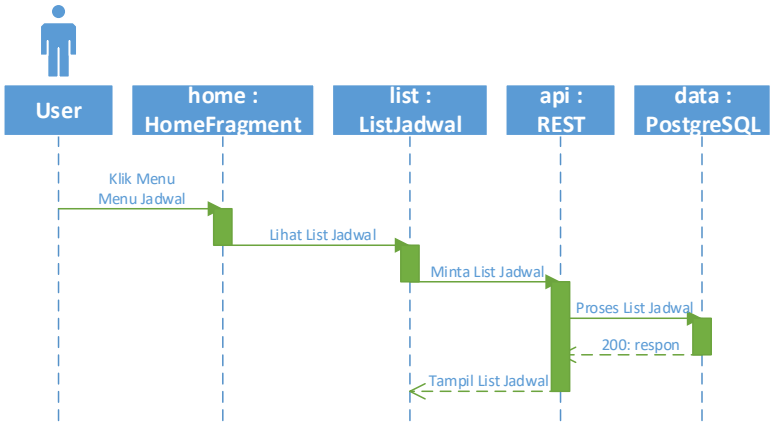
jelasnya *sequence* diagram mengunduh kalender akademik disajikan pada Gambar 64.



Gambar 64 *Sequence* Diagram Mengunduh Kalender Akademik

11) *Sequence* Diagram Menampilkan Jadwal

Untuk menampilkan menu jadwal *user* harus mengklik menu Jadwal Kuliah yang terdapat pada Halaman *Home*, maka *client* meminta *list* jadwal kuliah sesuai hari tersebut ke *server* melalui API untuk melihat *list* jadwal kuliah dari *database*. Untuk lebih jelasnya *sequence* diagram menampilkan jadwal disajikan pada Gambar 65.



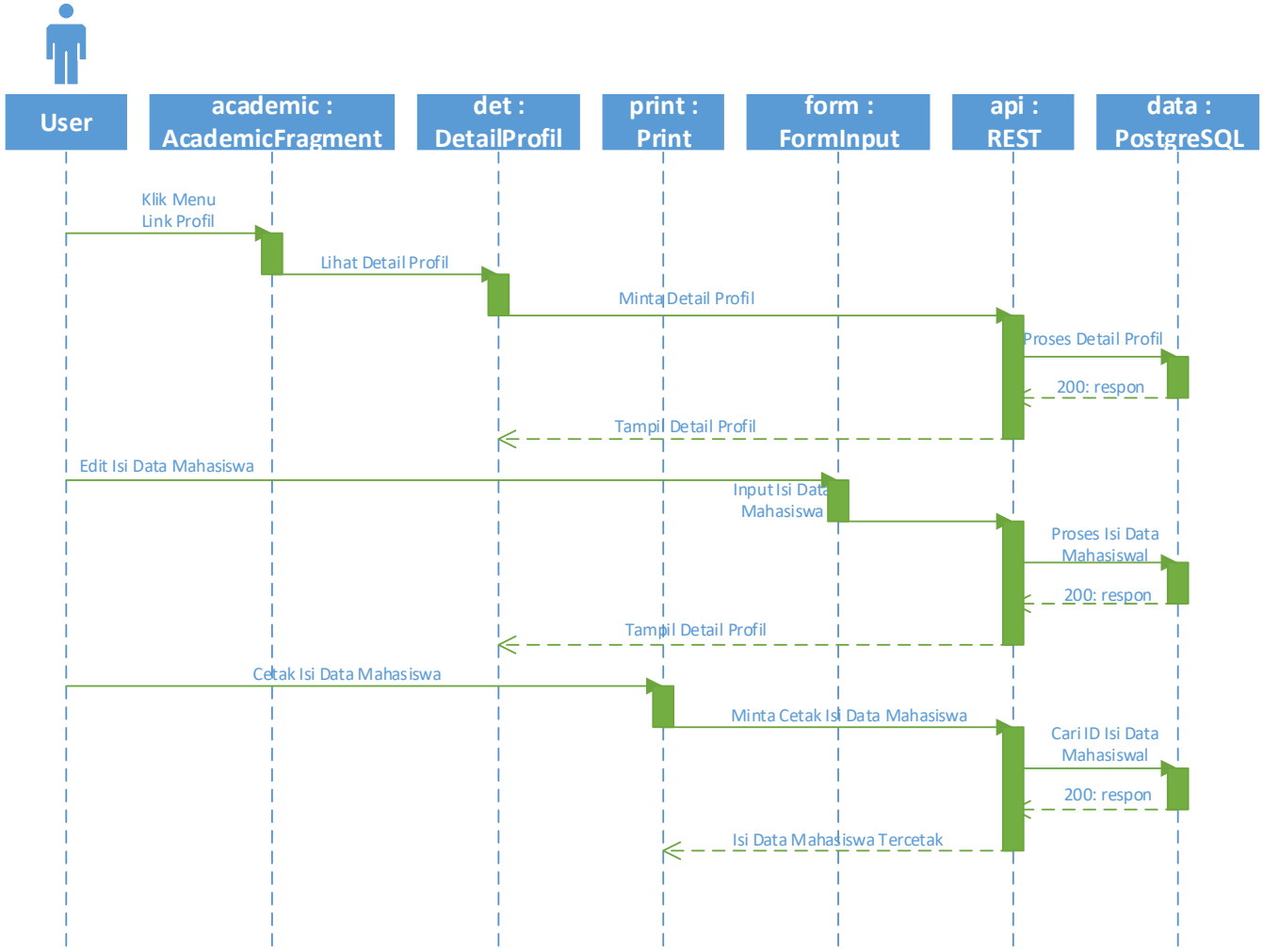
Gambar 65 *Sequence* Diagram Menampilkan Jadwal

12) *Sequence* Diagram Mengelola Isi Data Mahasiswa

Untuk mengelola isi data mahasiswa *user* harus mengklik *link* Profil Mahasiswa yang terdapat di Halaman Akademik, maka *client* akan meminta detail mahasiswa ke *server* melalui API untuk ditampilkan di Halaman Detail Profil.

User dapat melakukan *edit* isi data mahasiswa dengan mengklik menu *edit* yang terdapat pada Halaman Detail Profil, kemudian *client* akan menyediakan *form input* dan *user* meng*input*kan isi data mahasiswa dan menyimpan isi data mahasiswa ke *database* melalui API.

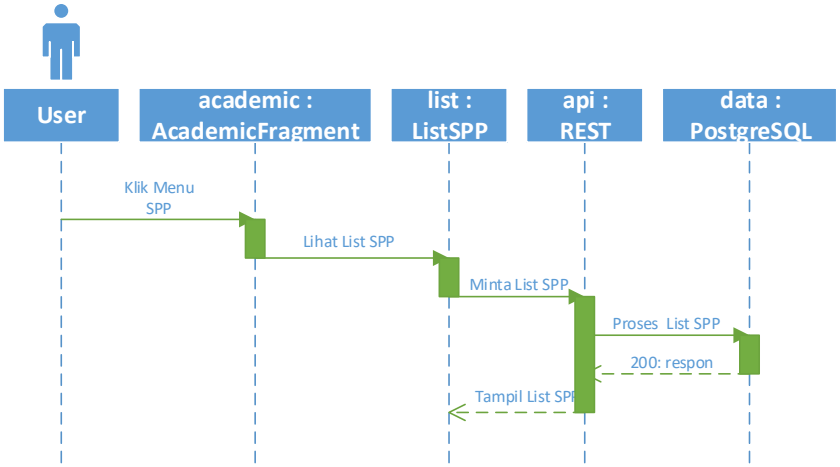
User dapat melakukan cetak isi data mahasiswa dengan mengklik menu cetak kemudian *client* meminta *server* untuk mencari *id* mahasiswa yang akan dicetak di *database*. Untuk lebih jelasnya *sequence* diagram mengelola isi data mahasiswa disajikan pada Gambar 66.



Gambar 66 *Sequence* Diagram Mengelola Isi Data Mahasiswa

13) *Sequence* Diagram Menampilkan Pembayaran SPP

Untuk menampilkan pembayaran SPP *user* harus mengklik menu SPP yang terdapat pada Halaman Akademik, maka *client* akan meminta *list* pembayaran SPP ke *server* untuk melalui API untuk menampilkan *list* pembayaran SPP dari *database*. Untuk lebih jelasnya *sequence* diagram menampilkan pembayaran SPP disajikan pada Gambar 67.

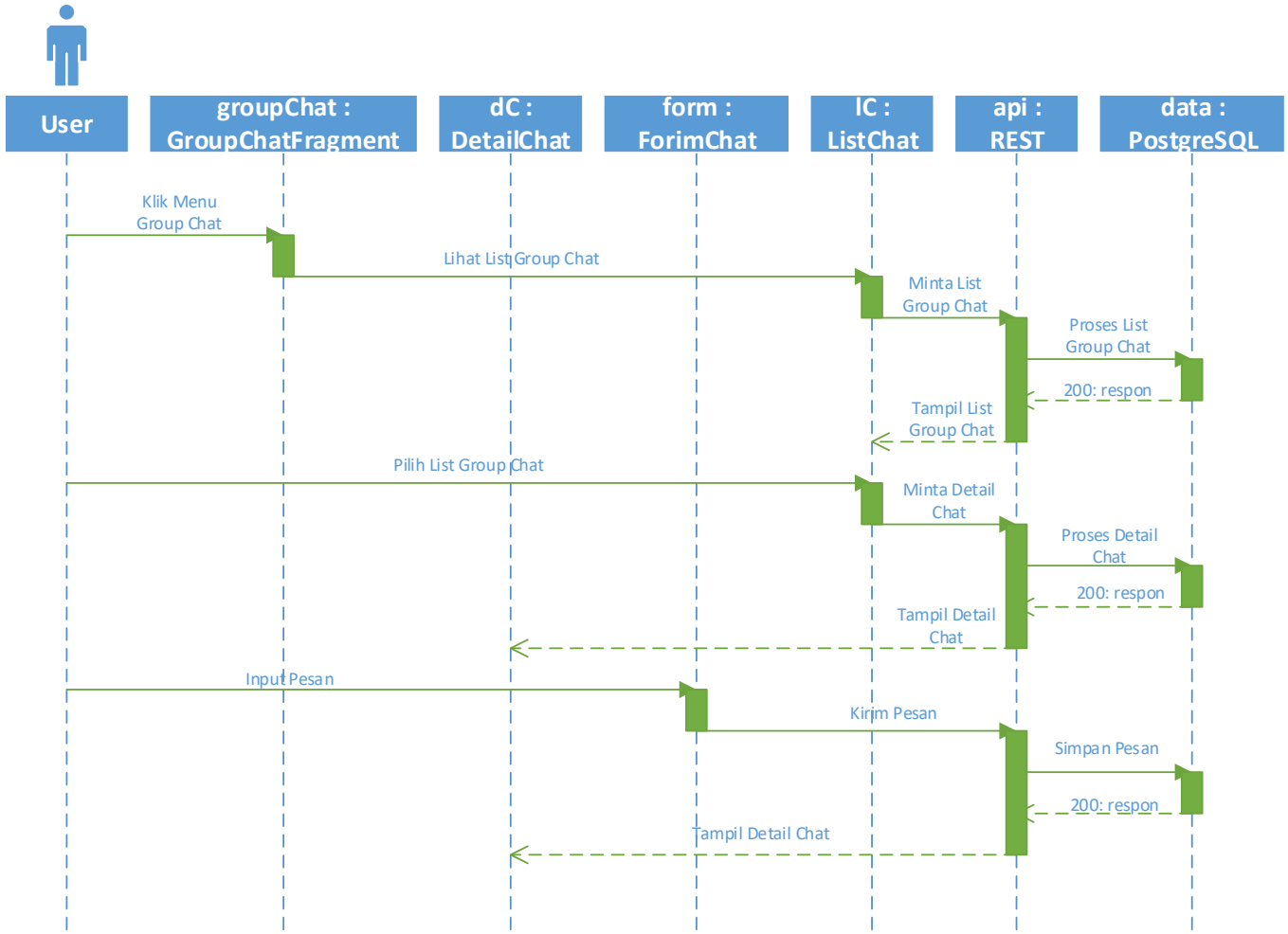


Gambar 67 *Sequence* Diagram Menampilkan Pembayaran SPP

14) *Sequence* Diagram Mengelola Forum

Untuk mengelola forum *user* harus mengklik menu Forum yang terdapat pada Halaman Pesan, maka *client* akan meminta *list group* chat ke *server* melalui API untuk menampilkan *list* forum dari *database*.

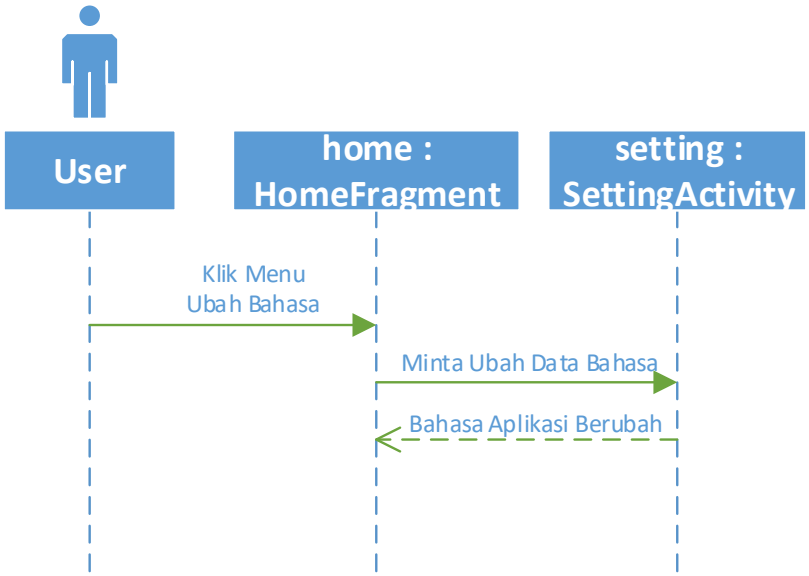
User dapat memilih forum dengan mengklik salah satu dan meminta *server* untuk melihat detail forum melalui API. *User* dapat mengirimkan pesan dengan meng*input*kan *form input* yang tersedia di Halaman Detail Forum dan menekan tombol "Kirim" untuk mengirimkan pesan ke Forum ke *server* melalui API. Untuk lebih jelasnya *sequence* diagram mengelola forum disajikan pada Gambar 68.



Gambar 68 *Sequence* Diagram Mengelola Forum

15) *Sequence* Diagram Mengatur Bahasa Aplikasi

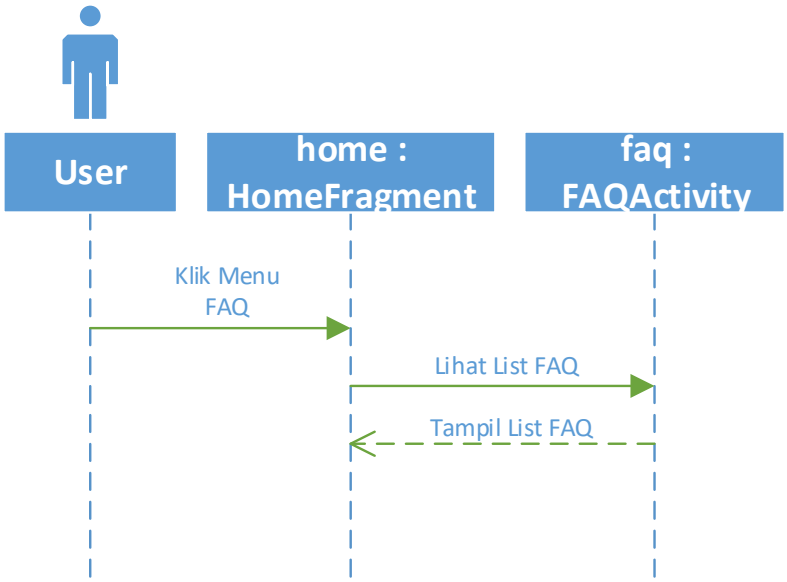
Untuk mengatur bahasa aplikasi *user* dapat mengklik menu Bahasa yang terdapat pada Halaman *Home*, maka *client* akan meminta pada sistem untuk mengubah bahasa aplikasi. Untuk lebih jelasnya *sequence* diagram mengubah bahasa aplikasi disajikan pada Gambar 69.



Gambar 69 *Sequence* Diagram Mengatur Bahasa Aplikasi

16) *Sequence* Diagram Menampilkan FAQ

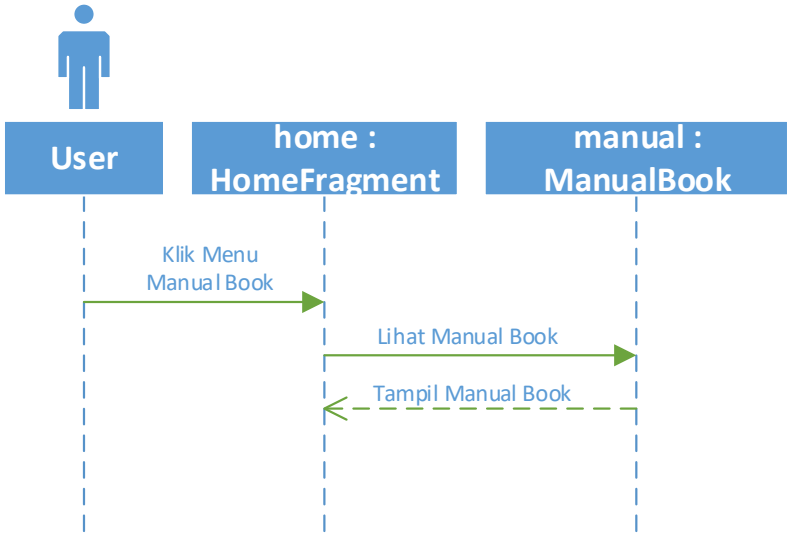
Untuk menampilkan FAQ *user* harus mengklik menu *popup More* untuk mengklik menu FAQ yang ada di Halaman *Home*, maka *client* akan meminta sistem untuk menampilkan *list* FAQ. Untuk lebih jelasnya *sequence* diagram menampilkan FAQ disajikan pada Gambar 70.



Gambar 70 *Sequence* Diagram Menampilkan FAQ

17) *Sequence* Diagram Menampilkan *Manual Book*

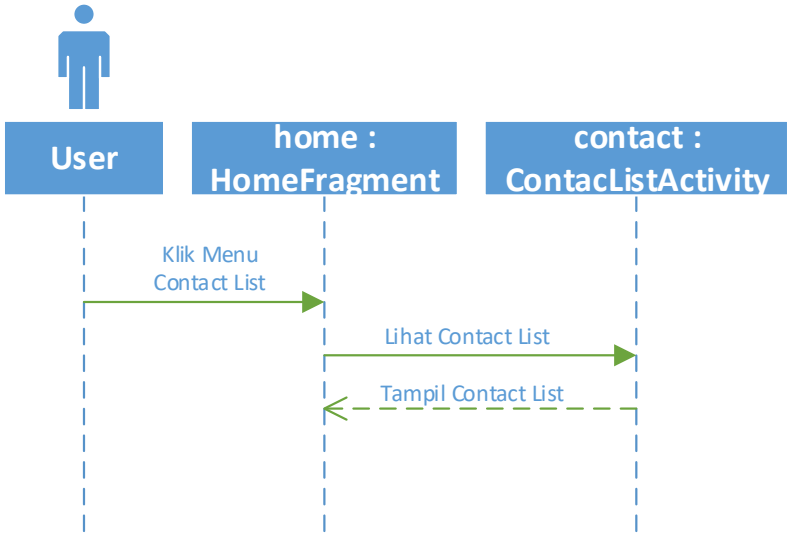
Untuk menampilkan *manual book user* harus mengklik menu *popup More* untuk mengklik menu *Manual Book* yang ada di Halaman *Home*, maka *client* akan meminta sistem untuk menampilkan *manual book*. Untuk lebih jelasnya *sequence* diagram menampilkan *manual book* disajikan pada Gambar 71.



Gambar 71 *Sequence* Diagram Menampilkan *Manual Book*

18) *Sequence* Diagram Menampilkan *Contact List*

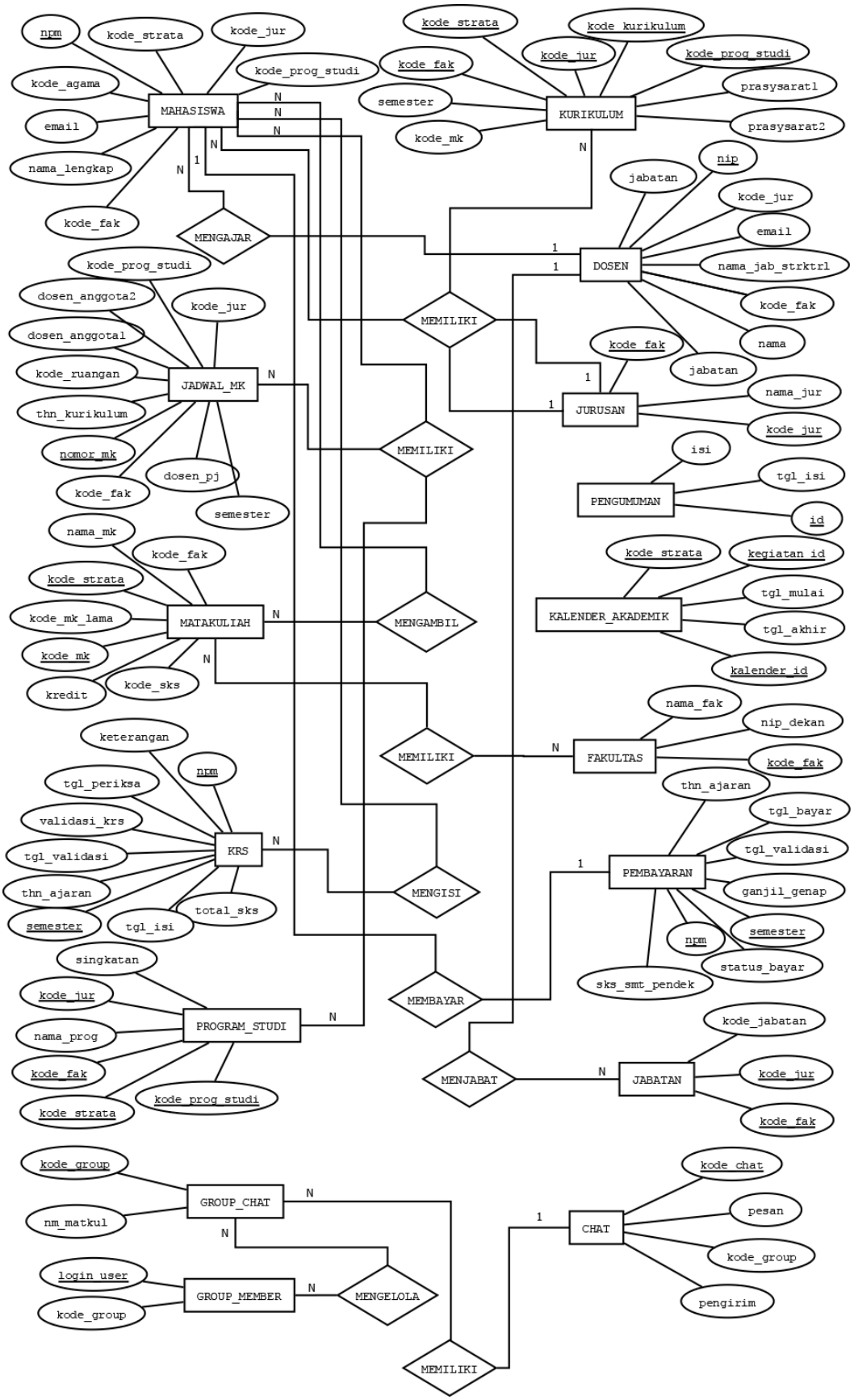
Untuk menampilkan *contact list user* harus mengklik menu *popup More* untuk mengklik menu *contact list* yang ada di Halaman *Home*, maka *client* akan meminta sistem untuk menampilkan *contact list*. Untuk lebih jelasnya *sequence* diagram menampilkan *contact list* disajikan pada Gambar 72.



Gambar 72 *Sequence* Diagram Menampilkan *Contact List*

e. *Entity Relationship Diagram* (ERD)

Entity Relationship Diagram (ERD) adalah suatu gambar atau diagram yang menjelaskan hubungan antar data dalam suatu *database*. Dalam penelitian ini ERD digunakan untuk menjelaskan hubungan antar data dalam suatu *database* dari *web server* aplikasi yang akan digunakan. Adapun Entity Relationship Diagram (ERD) untuk ini disajikan pada Gambar 73.



Gambar 73 *Entity Relationship Diagram* (ERD)

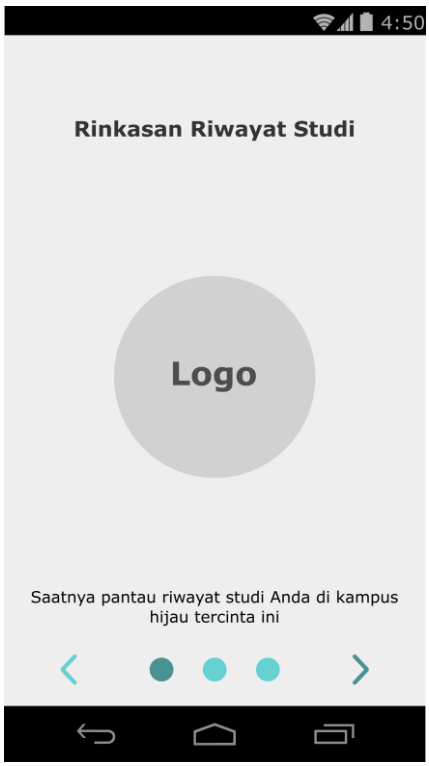
f. Perancangan *Interface* (Antarmuka) Aplikasi

1) Perancangan *Interface* (Antarmuka) Aplikasi Untuk Mahasiswa

Perancangan antarmuka ini dilakukan untuk merancang tata letak sistem sesuai dengan analisis kebutuhan sistem. Antarmuka yang dirancang dalam aplikasi ini adalah sebagai berikut.

a) *Layout Onboarding*

Layout Onboarding adalah halaman yang menampilkan perkenalan atau fungsi secara garis besar dari aplikasi "SIAKAD *Mobile*". Halaman ini akan muncul pertama kali saat aplikasi di *install* oleh pengguna. Perancangan *layout onboarding* disajikan pada Gambar 74, 75, dan 76.



Gambar 74 *Layout Onboarding 1*

Gambar 75 *Layout Onboarding 2*

Gambar 76 *Layout Onboarding 3*

b) *Layout Splash Screen*

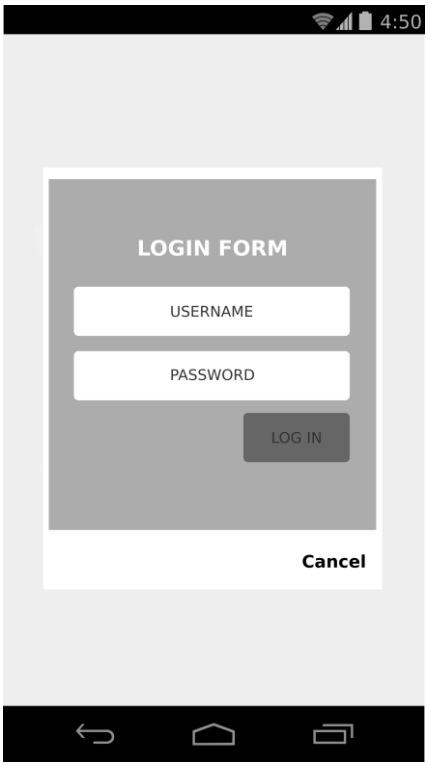
Splash Screen adalah halaman yang ditampilkan ketika aplikasi dijalankan. Aplikasi "SIAKAD *Mobile*" menggunakan animasi gambar saat pertama membuka aplikasi "SIAKAD *Mobile*" lalu dilanjutkan ke halaman login aplikasi. *Splash screen* di sini dimaksudkan sebagai estetika untuk menujukan identitas aplikasi saja, tanpa fungsi lainnya. Perancangan *layout splash screen* disajikan pada Gambar 77.

Gambar 77 *Layout Splash Screen*

c) *Layout Login*

Layout login merupakan halaman yang digunakan untuk proses otentikasi *user*. Halaman ini berisikan 2 *button* yaitu: *button Login* dan *button Info*. *Button Login* digunakan untuk melakukan otentikasi. Sedangkan *button Info* digunakan untuk melihat versi dari aplikasi. Perancangan Halaman *Login* disajikan pada Gambar 78 dan 79.

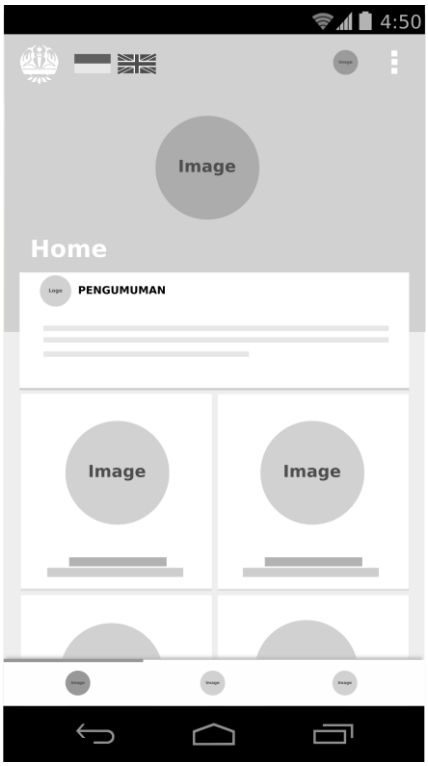
Gambar 78 *Layout Login*



Gambar 79 *Layout Form Login*

d) *Layout* Halaman Utama

Layout halaman utama berisikan menu-menu pilihan yang dapat digunakan oleh *user*. *Action* yang terdapat pada halaman utama aplikasi "SIAKAD *Mobile*" terbagi menjadi 3 bagian antara lain: (1) *action toolbar* (bagian atas) terdiri dari menu *notification, setting* bahasa aplikasi, *popup* menu (menu *Manual Book,* menu FAQ, menu *Contact* List). (2) *home* menu terdiri dari menu Jadwal Kuliah, menu Kalender, dan menu Wisuda, serta tampilan pengumuman terakhir. (3) *tab layout* (bagian bawah) terdiri dari *icon home, icon* akademik, dan *icon* pesan. Perancangan *layout* halaman *home* disajikan pada Gambar 80.

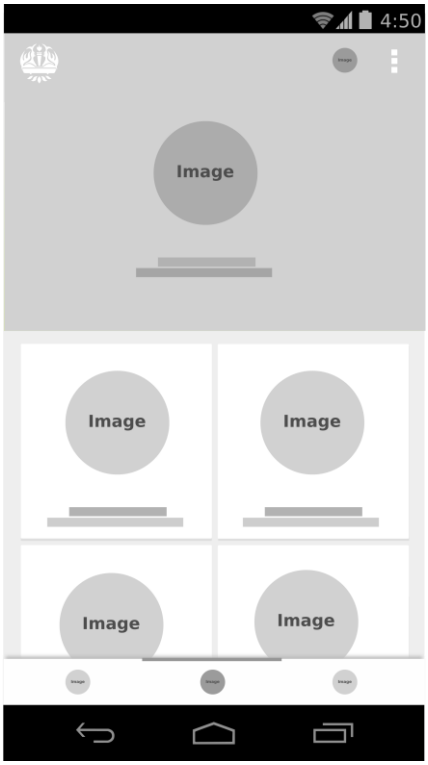


Gambar 80 *Layout* Halaman Utama

e) *Layout* Halaman Akademik

Layout halaman akademik berisikan menu yang berhubungan dengan data studi mahasiswa. Halaman ini menampilkan *preview* nama dan NPM mahasiswa login serta *action* yang terdapat pada halaman akademis aplikasi "SIAKAD *Mobile*" terbagi menjadi 3 bagian antara lain: (1) *action toolbar* (bagian atas) terdiri dari menu *notification*, dan *popup* menu (menu *Manual Book*, menu FAQ, menu *Contact* List). (2) *home* menu terdiri dari menu KRS, menu Nilai, dan menu Kurikulum, dan menu SPP. (3) *tab layout* (bagian bawah) terdiri dari *icon home*, *icon* akademik, dan

icon pesan. Perancangan *layout* halaman akademik disajikan pada Gambar 81.



Gambar 81 *Layout* Halaman Akademik

f) *Layout* Halaman Pesan

Layout halaman pesan berisikan tampilan untuk berkomunikasi dengan dosen pengampu dan pembimbing akademik. Halaman pesan memiliki 2 *sub* halaman yang dikendalikan menggunakan tampilan *tab layout* yang terletak di bagian atas diantaranya halaman pemberitahuan dan *sub* halaman forum.

Halaman pemberitahuan berisikan pemberitahuan dari dosen pengampu mata kuliah atau pembimbing akademik dalam

tampilan *list*, tampilan ini di dukung dengan menggunakan *scroll view* agar pengguna lebih mudah melakukan geser tampilan *list* pemberitahuan. Perancangan *layout* halaman pemberitahuan disajikan pada Gambar 82.

Gambar 82 *Layout* Halaman Pemberitahuan

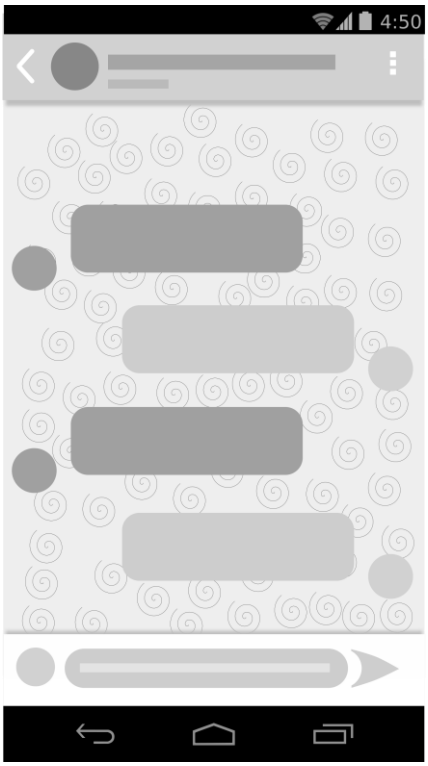
g) *Layout* Halaman Forum

Halaman forum berisikan *list* group *chat* mahasiswa dan dosen mata kuliah yang diambil dan mahasiswa dengan dosen pembimbing akademik. Perancangan *layout* halaman forum disajikan pada Gambar 83.

Gambar 83 *Layout* Halaman Forum

h) *Layout* Halaman Detail Forum

Layout halaman detail forum berisikan percakapan antara anggota *group* tersebut. Untuk *group* mata kuliah berisikan anggota mahasiswa yang mengambil mata kuliah tersebut dan dosen pengampu sedangkan untuk *group* bimbingan akademik berisikan anggota mahasiswa dan dosen pembimbing akademik. Perancangan *layout* halaman detail *group* chat disajikan pada Gambar 84.



Gambar 84 *Layout* Halaman Detail Forum

i) *Layout* Menu *Popup* Notifikasi

Layout menu *popup* notifikasi berisikan *list* notifikasi 5 terakhir. Tampilan ini juga berisikan *button* "*See More*" yang digunakan untuk melihat detail seluruh notifikasi yang pernah masuk ke aplikasi. Perancangan *layout* menu *popup* notifikasi disajikan pada Gambar 85.

Gambar 85 *Layout* Menu *Popup* Notifikasi

j) *Layout* Menu *Popup Support*

Layout menu *popup support* berisikan menu-menu untuk mengakses halaman dari menu *Manual Book* yaitu halaman yang berisikan tentang cara pengunaan aplikasi, halaman menu FAQ yaitu halaman yang berisikan tentang seputar pertanyaan yang sering ditanyakan di dalam aplikasi, dan halaman menu *Contact List* yaitu halaman yang berisikan *list contact* dari seluruh prodi, jurusan dan fakultas yang ada di Universitas Lampung. Perancangan *layout* menu *popup support* disajikan pada Gambar 86.

Gambar 86 *Layout* Menu *Popup Support*

k) *Layout* Menu Jadwal Mata Kuliah

Layout menu jadwal mata kuliah berisikan tampilan jadwal mata kuliah yang diambil oleh mahasiswa berdasarkan hari aktif kuliah. Pengguna ketika mengklik menu Jadwal Kuliah maka halaman akan meminta jadwal kuliah berdasarkan hari tersebut. Perancangan *layout* jadwal kuliah disajikan pada Gambar 87.

Gambar 87 *Layout* Menu Jadwal Kuliah

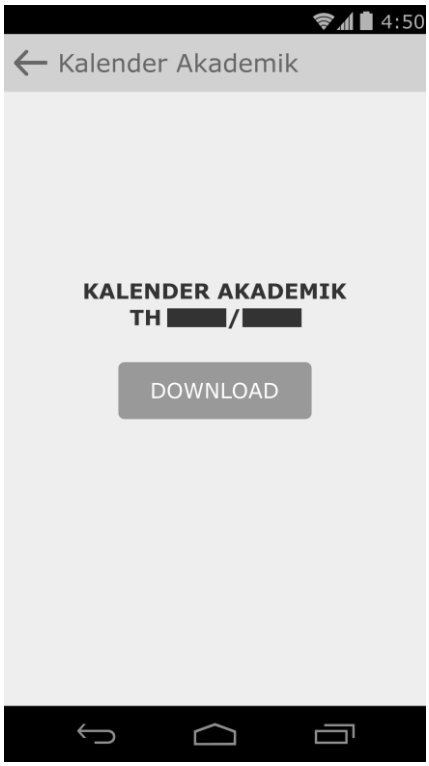
l) *Layout* Menu Wisuda

Layout menu wisuda berisikan *button* otorisasi untuk login ke halaman *web* sistem informasi wisuda *online*. Perancangan *layout* menu wisuda disajikan pada Gambar 88.

Gambar 88 *Layout* Menu Wisuda

m) *Layout* Menu Kalender Akademik

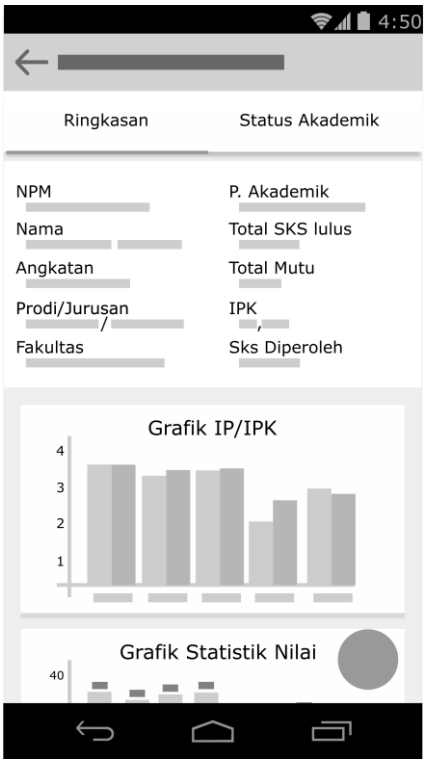
Layout menu kalender akademik berisikan *button* untuk men*download* dokumen kalender akademik sesuai tahun ajaran yang sedang berlaku. Perancangan *layout* kalender akademik disajikan pada Gambar 89.



Gambar 89 *Layout* Menu Kalender Akademik

n) *Layout* Halaman Ringkasan Studi

Layout halaman ringkasan studi berisikan tampilan ringkasan data diri mahasiswa, grafik IP dan IPK, dan grafik statistik nilai. Selain itu terdapat menu Edit dan Cetak IDM (Isi Data Mahasiswa). Perancangan *layout* halaman ringkasan studi disajikan pada Gambar 90 dan 91.



Gambar 90 *Layout* Halaman Ringkasan Studi 1

Gambar 91 *Layout* Halaman Ringkasan Studi 2

o) *Layout* Menu Status Akademik

Layout menu status akademik berisikan *list* status akademik dari *user* memulai perkuliahan sampai dengan saat ini. Perancangan *layout* menu status akademik disajikan pada Gambar 92.

Gambar 92 *Layout* Menu Status Akademik

p) *Layout* Menu KRS

Layout menu KRS berisikan halaman untuk menampilkan KRS (Kartu Rencana Studi) yang diambil oleh *user*. Selain itu terdapat *button* "Edit" yang digunakan untuk memperbarui KRS dari kesalahan selama masih batas pengisian dan tombol "Isi Baru" digunakan untuk membuat

KRS baru dengan waktu isi masih berlaku. Perancangan *layout* menu KRS disajikan pada Gambar 93 dan 94.

Gambar 93 *Layout* Menu KRS 1

Gambar 94 *Layout* KRS 2

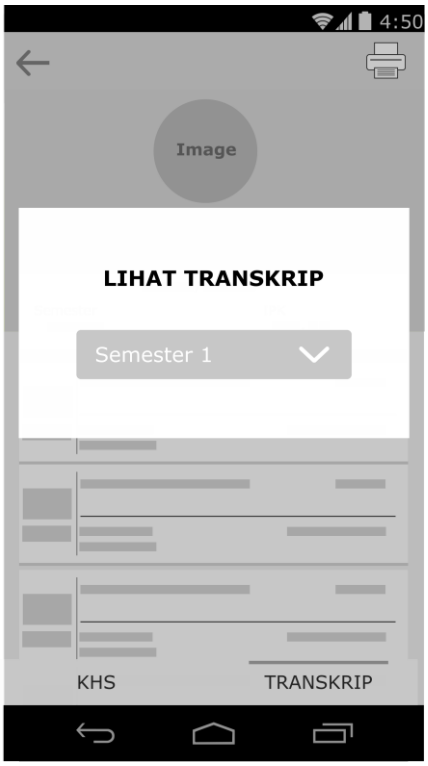
q) *Layout* Halaman Nilai

Layout halaman nilai berisikan tampilan nilai atau KHS (Kartu Hasil Studi) selama mengikuti perkuliahan. Tampilan nilai berisikan *list* nilai mata kuliah persemester, untuk dapat melihat *list* nilai semester selanjutnya dapat digunakan *action swipe* ke kanan atau ke kiri. *User* dapat mencetak KHS dengan mengklik *icon* "Cetak" yang terdapat pada *action bar*. Perancangan *layout* halaman nilai disajikan pada Gambar 95.

Gambar 95 *Layout* Halaman KHS

r) *Layout* Halaman Transkrip

Layout halaman transkrip berisikan *list* nilai berdasarkan jangkauan semester yang dipilih oleh *user*. Sebelum melihat *list* nilai *user* terlebih dahulu memilih jangkauan semester yang akan dilihat. Perancangan *layout* transkrip disajikan pada Gambar 96 dan 97.

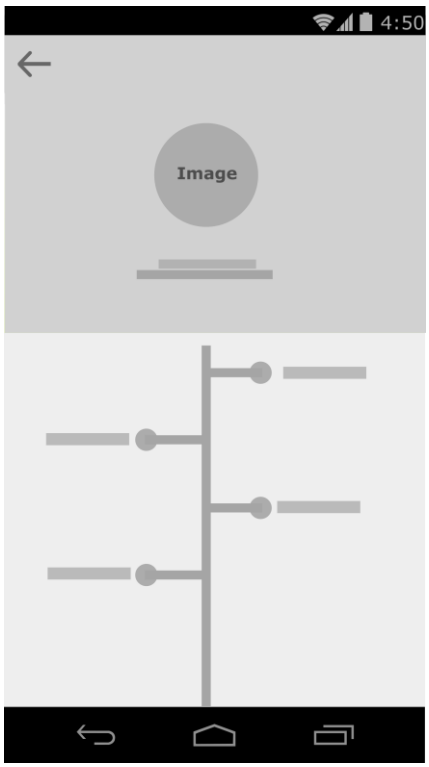


Gambar 96 *Layout* Halaman Transkrip 1

Gambar 97 *Layout* Halaman Transkrip 2

s) *Layout* Menu Kurikulum

Layout menu kurikulum berisikan *list* kurikulum yang pernah ada di program studi *user*. *User* dapat memilih *list* kurikulum untuk melihat detail kurikulum. Perancangan *layout* menu kurikulum disajikan pada Gambar 98.



Gambar 98 *Layout* Manu Kurikulum

t) *Layout* Menu SPP

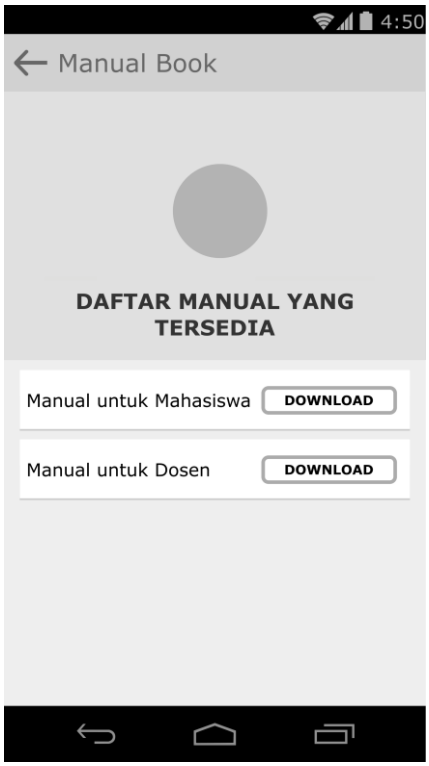
Layout menu SPP berisikan *list* pembayaran SPP oleh *user*. *User* dapat mengetahui pembayaran SPP yang mereka lakukan melalui menu SPP atau dari menu notifikasi. Pada halaman *list* pembayaran SPP *user* dapat dengan mudah mengetahui indikator pembayaran SPP melalui tampilan

yang berbeda warna. Perancangan *layout* pembayaran SPP disajikan pada Gambar 99.

Gambar 99 *Layout* Menu SPP

u) *Layout* Menu *Manual Book*

Manual Book adalah suatu dokumen komunikasi teknis yang bertujuan memberikan bantuan untuk penggunaan suatu sistem. *Layout* menu *manual book* berisikan daftar cara penggunaan aplikasi "SIAKAD *Mobile*". Perancangan *layout* menu *manual book* disajikan pada Gambar 100.



Gambar 100 *Layout* Menu *Manual Book*

v) *Layout* Menu FAQ

FAQ adalah daftar kumpulan pertanyaan dan jawaban yang sering di pertanyakan tentang berbagai hal. *Layout* menu FAQ berisikan *list* pertanyaan yang sering ditanyakan dalam aplikasi. Perancangan *layout* menu FAQ disajikan pada Gambar 101.

Gambar 101 *Layout* Menu FAQ

w) *Layout* Menu *Contact List*

Layout menu *contact list* berisikan daftar kontak informasi penting yang berhubungan program studi *user* mahasiswa. Perancangan *layout* menu *contact list* disajikan pada Gambar 102.

Gambar 102 *Layout* Menu *Contact List*

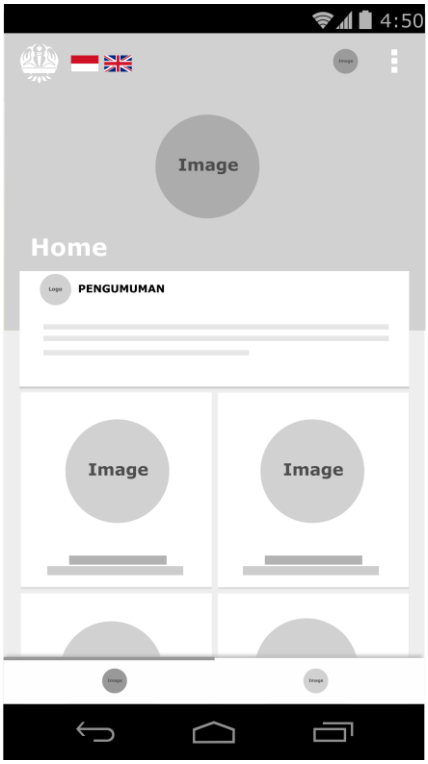
2) Perancangan *Interface* (Antarmuka) Aplikasi Untuk Dosen Pengampu Mata Kuliah dan Pembimbing Akademik

Perancangan antar muka aplikasi untuk *user* dosen pengampu mata kuliah dan pembimbing akademik memiliki beberapa perbedaan dari fungsionalitas menu. Secara umum perancangan antarmuka sama dengan *user* mahasiswa. Antarmuka yang dirancang untuk dosen pengampu mata kuliah dan pembimbing akademik adalah sebagai berikut.

a) *Layout* Halaman Utama

Layout halaman utama *user* dosen berisikan menu diantarnya: (1) bagian atas terdapat menu Ubah Bahasa Aplikasi, menu Notifikasi, dan menu *Support*. (2) bagian utama terdapat menu jadwal dan menu kalender akademik.

(3) bagian bawah terdapat menu untuk halaman utama dan menu untuk halaman pesan. Perancangan *layout* halaman utama disajikan pada Gambar 103.



Gambar 103 *Layout* Halaman Dosen

b) *Layout* Halaman Pesan

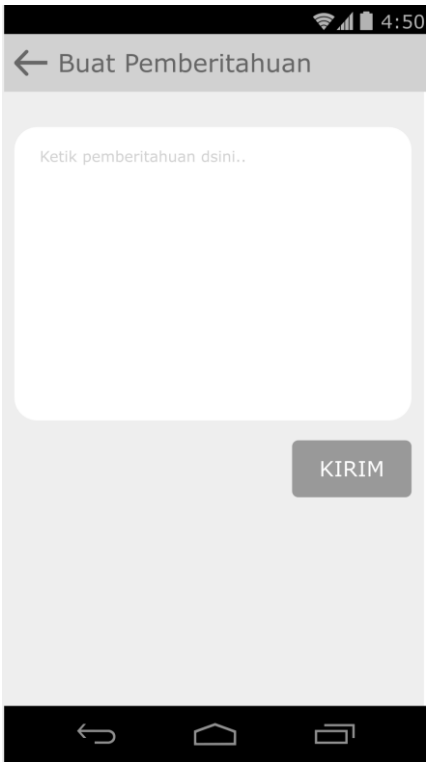
Layout halaman pesan berisikan menu pemberitahuan yang pernah di *post* oleh *user* dosen dan halaman forum dengan mahasiswa yang mengambil mata kuliah yang diampu oleh *user* dosen dan mahasiswa bimbingan akademik. *User* dapat membuat atau menambahkan *post* pemberitahuan baru dengan mengklik *button* "Tambah" yang terdapat pada bagian pojok kanan bawah halaman pemberitahuan aplikasi.

Perancangan *layout* halaman pesan disajikan pada Gambar 104.

Gambar 104 *Layout* Halaman Pesan

c) *Layout* Halaman Membuat Pemberitahuan

Layout halaman membuat pemberitahuan berisikan *form* berupa *textbox* untuk menginputkan pemberitahuan ke halaman *post* pemberitahuan. Kemudian *user* dosen mengklik *button* "Kirim" untuk mengirimkan *text* pemberitahuan. Perancangan *layout* halaman membuat pemberitahuan disajikan pada Gambar 105.



Gambar 105 *Layout* Halaman Membuat Pemberitahuan

5. *Implementation Workflow*

Pada tahap ini akan dilakukan pembuatan program (*coding*), tahap *coding* dilakukan dengan mengimplementasikan permasalahan ke dalam sistem. Proses pembuatan program (*coding*) yang dilakukan pada tahap ini menggunakan bahasa pemrograman Java dan Kotlin dengan bantuan aplikasi Android Studio 3.0. Sedangkan dalam membuat API (*Application Programming Interface*) untuk menangani akses dan transaksi data dari *database* menggunakan Django REST *Framework*.

Pada aplikasi ini dibuat beberapa *class* java yang menjelaskan jalannya aplikasi khususnya fungsi otentikasi dan otorisasi *user* ke *server* untuk melakukan proses transaksi data menggunakan *web service* metode

REST. Aplikasi "SIAKAD *Mobile*" ini menggunakan *database server* untuk menyimpan data-data universitas.

6. *Test Workflow*

Setelah pembuatan program selesai, maka akan dilakukan pengujian. Pendekatan kasus uji dalam penelitian ini adalah pengujian *black box* dengan metode *Equivalence Partitioning* (EP), dimana metode ini akan membagi domain masukan dari program kedalam kelas-kelas sehingga *test case* dapat diperoleh. EP berusaha untuk mendefinisikan kasus uji yang menemukan sejumlah jenis kesalahan, dan mengurangi jumlah kasus uji yang harus dibuat.

EP berdasarkan pada premis masukan dan keluaran dari suatu komponen yang dipartisi ke dalam kelas-kelas, menurut spesifikasi dari komponen tersebut, yang akan diperlakukan sama (*ekuivalen*) oleh komponen tersebut. Pada pengujian ini harus diyakinkan bahwa masukan yang sama akan menghasilkan respon yang sama pula. Alasan menggunakan metode EP pada pengujian aplikasi "SIAKAD *Mobile*" ini adalah karena metode ini dapat digunakan untuk mencari kesalahan pada fungsi, dapat mengetahui kesalahan pada *interface* dan kesalahan pada struktur data sehingga dapat mengurangi masalah terhadap nilai masukan. Rancangan daftar pengujian disajikan pada Tabel 10.

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
1	Versi Android	Pengujian kompatibilitas versi *operating system* android	Pengujian pada android versi 4.0.3 (*Ice Cream Sandwich*)	Kompatibel dengan android versi 4.0.3 (*Ice Cream Sandwich*)
			Pengujian pada android versi 4.1-4.3.1 (*Jelly Bean*)	Kompatibel dengan android versi 4.1-4.3.1 (*Jelly Bean*)
			Pengujian pada android versi 4.4–4.4.4 (KitKat)	Kompatibel dengan android versi 4.4–4.4.4 (*KitKat*)
			Pengujian pada android versi 5.0 (*Lolipop*)	Pengujian pada android versi 4.1 (*Lolipop*)
			Pengujian pada android versi 6.0–6.0.1 (*Marshmallow*)	Pengujian pada android versi 6.0–6.0.1 (*Marshmallow*)
			Pengujian pada android versi 7.0–7.1.2 (*Nougat*)	Pengujian pada android versi 7.0–7.1.2 (*Nougat*)
2	Resolusi Layar dan Densitas Layar	Pengujian Resolusi Layar dan Densitas Layar pada android	Pengujian pada android dengan resolusi 4 inch	Kompatibel Pada android dengan resolusi 4 inch
			Pengujian terhadap resolusi 4.5 inch	Kompatibel terhadap resolusi 4.5 inch

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android (Lanjutan)

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
			Pengujian pada android dengan resolusi 5 inch	Kompatibel pada android dengan resolusi 5 inch
			Pengujian pada android dengan resolusi 5.7 inch	Kompatibel pada android dengan resolusi 5.7 inch
3	*Interface*	Pengujian pada *icon* SIAKAD *Mobile*	Klik *icon* pada perangkat android pengguna	Menampilkan *layout splash screen* (untuk pertama kali muncul *layout onboarding*)
		Pengujian pada halaman otentikasi	Klik tombol menu *Login*	Menampilkan *layout* otentikasi
			Klik tombol menu *Info*	Menampilkan *box dialog* versi aplikasi
		Pengujian pada halaman *home*	Klik *icon tab Home*	Menampilkan halaman *Home*
			Klik tombol *action bar* Notifikasi	Menampilkan *popup dialog* Notifikasi
			Klik tombol *action bar Support*	Menampilkan *popup dialog* menu *Support*

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android (Lanjutan)

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
			Klik tombol menu Pengumuman	Menampilkan *layout* Pengumuman
			Klik tombol menu Jadwal Kuliah	Menampilkan *layout* Jadwal Kuliah
			Klik tombol menu Kalender Akademik	Menampilkan *layout* Kalender Akademik
			Klik tombol menu Wisuda	Menampilkan *layout* Wisuda
		Pengujian pada halaman akademik	Klik *icon tab* Akademik	Menampilkan halaman akademik
			Klik *link* nama atau NPM	Menampilkan *layout* ringkasan studi dan status akademik
			Klik tombol menu KRS	Menampilkan *layout* KRS
			Klik tombol Nilai	Menampilkan *layout* Nilai
			Klik tombol Kurikulum	Menampilkan *layout* Kurikulum
			Klik tombol SPP	Menampilkan *layout* Pembayaran SPP
		Pengujian pada halaman pesan	Klik icon tab Pesan	Menampilkan halaman Pemberitahuan
			Klik icon tab Forum	Menampilkan layout Forum
			Swipe atau geser halaman pesan	Menampilkan secara berganti layout Pemberitahuan dan layout Forum

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android (Lanjutan)

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
		Pengujian halaman home dosen	Klik icon tab home	Menampilkan layout Home Menu
		Pengujian halaman tambah pemberitahuan	Klik tombol tambah pemberitahuan	Menampilkan layout form input pemberitahuan
4	Fungsi pada popup Notifikasi	Pengujian layout popup Notifikasi	Klik action bar Notifikasi	Menampilkan list notifikasi 5 terakhir dan link "See More"
			Klik *link* "*See More*"	Menampilkan *list* pengumuman secara keseluruhan
5	Fungsi pada *popup Support*	Pengujian *layout popup Support*	Klik *action bar Support*	Menampilkan *Manual Book,* FAQ, *Contact List,*
6	Fungsi pada menu *Manual Book*	Pengujian pada *layout Manual Book*	Klik menu *Manual Book*	Menampilkan *layout* untuk men*download manual* yang tersedia pada aplikasi
7	Fungsi pada menu FAQ	Pengujian pada *layout* FAQ	Klik menu FAQ	Menampilkan *list* FAQ
8	Fungsi pada menu *Contact List*	Pengujian pada *layout Contact List*	Klik menu *Contact List*	Menampilkan *list* kontak informasi program studi
9	Fungsi pada menu Jadwal Kuliah	Pengujian pada *layout* Jadwal Kuliah	Klik menu Jadwal Kuliah	Menampilkan *list* jadwal pada hari tersebut

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android (Lanjutan)

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
			Geser atau *swipe* ke kanan atau ke kiri pada halaman jadwal kuliah	Menampilkan *list* jadwal kuliah pada hari besok dan kemarin
10	Fungsi pada menu Kalender Akademik	Pengujian pada *layout* Kalender Akademik	Klik menu Kalender Akademik	Menampilkan tombol *download* kalender akademik terbaru
			Klik *download*	Men*download* kalender akademik
11	Fungsi pada menu Wisuda	Pengujian pada *layout* Web Wisuda	Klik menu Wisuda	Menampilkan otorisasi masuk *web* sistem informasi wisuda *online*
			Klik login	Menampilkan *webview* halaman wisuda *online*
12	Fungsi pada *link* NPM Mahasiswa	Pengujian pada *layout* Ringkasan Studi	Klik *link* Nama atau NPM	Menampilkan halaman Ringkasan Studi
			Klik *icon tab* Status Akademik	Menampilkan *list* status akademik
			Geser atau *swipe* ke kanan atau ke kiri pada halaman ringkasan studi	Mengganti halaman secara bergantian antara halaman Ringkasan Studi dan Status Akademik
13	Fungsi pada menu KRS	Pengujian pada *layout* KRS	Klik menu KRS	Menampilkan halaman *list* KRS saat semester sekarang

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android (Lanjutan)

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
			Klik *button* "*Edit*"	Menampilkan *form* input KRS
			Geser atau *swipe* halaman KRS	Mengganti tampilan KRS dari semester lainnya
14	Fungsi pada menu Nilai	Pengujian pada *layout* Nilai	Klik menu Nilai	Menampilkan halaman nilai semeter sekarang
			Geser atau *swipe* pada halaman nilai	Mengganti halaman nilai terhadap semester
			Klik menu Transkrip	Menampilkan halaman memilih jangkauan semeter
			Klik Lihat pada halaman Transkrip	Menampilkan halaman *list* nilai berdasarkan pada jangkauan semester
			Klik tombol menu "Cetak"	Mencetak nilai berdasarkan semester (jika di halaman nilai) dan transkrip (jika di halaman transkrip)
15	Fungsi menu kurikulum	Pengujian pada *layout* kurikulum	Klik menu kurikulum	Menampilkan *list* kurikulum berdasarkan program studi
			Klik salah satu *list* kurikulum	Menampilkan halaman detail *list* mata kuliah berdasarkan kurikulum
16	Fungsi menu SPP	Pengujian pada *layout* SPP	Klik menu SPP	Menampilkan *list* pembayaran SPP
17	Fungsi tombol menu pesan (halaman dosen)	Pengujian *layout* halaman dosen	Klik *icon tab* pesan	Menampilkan *form input* pemberitahuan
18	Fungsi tombol ubah bahasa	Pengujian pada mengubah bahasa aplikasi	Klik tombol "Kirim"	Menampilkan respon kirim pemberitahuan
			Klik tombol ubah bahasa	Mengubah bahasa aplikasi dari sebelumnya

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android (Lanjutan)

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
19	Fungsi pada *background* aplikasi	Pengujian pada *register device*	Klik *icon* aplikasi	Aplikasi mendaftarkan ID *Device* secara *realtime*
			ID *device* berhasil terdaftar	ID *Device* terdaftar ke *database server* melalui *background service*
			ID *device* gagal terdaftar	Menampilkan peringatan gagal terdaftar
20	Koneksi Internet dan *Server*	Pengujian pada koneksi internet dan *server* saat mengakses data	Koneksi internet stabil dan *server* stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
			Koneksi internet tidak stabil dan *server* stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
			Koneksi internet stabil dan *server* tidak stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
			Koneksi internet tidak stabil dan *server* tidak stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
21	Simpan data offline *user*	Pengujian penyimpanan *local* data	*Input* data identitas *user*	Menyimpan data *user* dalam *database* aplikasi
			Penyimpanan *database local* berhasil	*User* menggunakan aplikasi tanpa *login* kembali
22	Fungsi tombol *Login*	Pengujian pada otentikasi	*Username* = 1417051129, *Password* = 123456, Klik tombol l*ogin*	Masuk ke halaman utama aplikasi

Tabel 10 Daftar Pengujian *Equivalence Partitioning* Aplikasi Android (Lanjutan)

No	Kelas Uji	Daftar Pengujian	Skenario Uji	Hasil yang Diharapkan
			Username = 1417051129, Password = 123456, Klik tombol login ID *device* gagal terdaftar	Menampilkan kotak dialog password salah Menampilkan peringatan gagal terdaftar
20	Koneksi Internet dan *Server*	Pengujian pada koneksi internet dan *server* saat mengakses data	Koneksi internet stabil dan *server* stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
			Koneksi internet tidak stabil dan *server* stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
			Koneksi internet stabil dan *server* tidak stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
			Koneksi internet tidak stabil dan *server* tidak stabil saat mengakses data	Aplikasi berjalan dengan lancar saat sedang mengakses data
21	Simpan data offline *user*	Pengujian penyimpanan *local* data	*Input* data identitas *user*	Menyimpan data *user* dalam *database* aplikasi
			Penyimpanan *database local* berhasil	*User* menggunakan aplikasi tanpa *login* kembali
22	Fungsi tombol *Login*	Pengujian otentikasi	*Username* = 1417051129, *Password* = 123456, Klik tombol l*ogin*	Masuk ke halaman utama aplikasi
			Username = 1417051129, *Password* = 123456, Klik l*ogin*	Menampilkan kotak dialog password salah

7. *Deployment Workflow*

Tahap *deployment workflow* ini merupakan tahap pengaplikasian atau pemasangan sistem. Kegiatan yang dilakukan pada tahap ini yaitu membuat rancangan *deployment diagram*, kemudian penginstalasian sistem pada komputer server. Tahap terakhir yaitu membuat panduan atau User manual yang digunakan oleh pengguna yang memanfaatkan sistem informasi akademik Universitas Lampung .

Pada tahap ini akan dilakukan penyerahan sistem aplikasi ke-*user* (*roll-out*) melalui *Play Store*. *Play Store* adalah layanan konten digital milik Google yang melingkupi toko untuk produk-produk seperti musik/lagu, buku, aplikasi, permainan, ataupun pemutar media berbasis *cloud*. *Roll-out* ini dilakukan agar pengguna khususnya *civitas akademika* Unila dapat mengunduh aplikasi "SIAKAD *Mobile*" secara gratis melalui *Play Store*.

8. *Project Management Workflow*

Project manajement workflow ini merupakan pendefinisian dari berbagai strategi untuk bekerja dengan proses iterasi. Kegiatan yang dilakukan yaitu mengidentifikasi terhadap masalah yang terjadi di SIAKAD Universitas Lampung. Kemudian dilakukan juga mengidentifikasi sistem informasi akademik di Universitas Lampung. Setelah itu membuat rancangan pengembangan dan melaksanakan pengembangan serta melakukan pengujian sistem informasi akademik Universitas Lampung.

9. ***Configuration and Change Management Workflow***

   Configuration and Change Management Workflow mengendalikan perubahan pengembangan dan memelihara integrasi hasil pengembangan dan aktifitas manajemen. Kegiatan yang dilakukan yaitu membuat rancangan daftar kelengkapan dan persyaratan minimum untuk menjalankan sistem informasi akademik dan rancangan dokumentasi sistem.

10. ***Environment Workflow***

    Environment workflow mencangkup seluruh kebutuhan infrastruktur yang dibutuhkan untuk mengembangkan suatu sistem. Kegiatan yang dilakukan yaitu membuat daftar yang digunakan dalam pengembangan sistem informasi akademik Universitas Lampung versi Android.

12. Jadwal Kegiatan Penelitian

Tabel 11 Jadwal Kegiatan Penelitian

Kegiatan	Tahun 2017														
	April			Mei				September				Oktober			
	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Bussiness Modeling Worklflow															
a. Identifikasi Masalah b. *Business Use Case*															
Analysis and Design Workflow															
a. Desain Arsitektur															
b. Desin UML c. Perancangan Antarmuka															
Implementation Workflow															
a. Pembuatan program (*Coding*)															
Test Workflow															
a. Pengujian (*Testing*)															
Deployment Workflow															
a) Penyerahan aplikasi ke *user* (*roll-out*)															
Project Management Workflow															
Configuration and Change Management Workflow															
Environment Workflow															

V. SIMPULAN DAN SARAN

A. SIMPULAN

Dari hasil penelitian yang dilakukan, penulis dapat mengambil simpulan sebagai berikut.

1. Telah berhasil dibangun aplikasi Siakad Unila versi Android yang dibuat untuk menghasilkan teknologi *web service* dan otentikasi dengan menerapkan teknologi Django REST *Framework* dan OAuth 2.0.
2. Aplikasi ini telah berhasil menampilkan jadwal perkuliahan, menampilkan kalender akademik, menampilkan halaman wisuda online, menampilkan nilai dan rencana studi, menampilkan kurikulum berdasarkan tahun angkatan, menampilkan rekam pembayaran SPP, ringkasan profil, dan menampilkan berbagai jenis forum serta postingan pada pengguna mahasiswa.
3. Aplikasi ini telah berhasil menampilkan dan menyimpan forum serta postingan pada pengguna dosen.
4. Dari hasil data pengujian *Equivalence Partitioning*, aplikasi Siakad *Mobile* kompatibel terhadap semua versi OS Android dengan minimum requirement yang telah ditetapkan dalam pembuatan aplikasi, kompatibel terhadap *device* Android dengan resolusi 4 inch, 4.5 inch, 5

inch, 5.7 inch dan dari semua kelas yang diuji aplikasi dapat berfungsi sesuai analisis.

5. Berdasarkan dari hasil data pengujian penilaian variabel *information on application, interactive, trust, response time, ease of understanding, visual, innovativeness and emotional appeal*, dan *consistent image and color* aplikasi Siakad *Mobile* termasuk dalam kategori (Sangat Baik).

B. SARAN

Berdasarkan perancangan dan hasil implementasi program sistem yang dilakukan, maka beberapa saran yang perlu diperhatikan dalam mengembangkan sistem ini adalah sebagai berikut.

1. Data yang disajikan hanya menggunakan data *dummy* dari beberapa tahun terakhir, untuk pengembangan selanjutnya diperlukan data *real* yang lebih lengkap untuk semua fitur yang ditawarkan, sehingga akan diimplementasikan ke seluruh pengguna dan menciptakan siakad generasi baru.
2. Aplikasi ini nantinya dapat dikembangkan sehingga kompatibel pada *platform* selain Android, seperti IOS.
3. Aplikasi ini nantinya mampu menerapkan *single platform* ketika melakukan otentikasi, sehingga rekam akses aplikasi lebih aman dari penyelewengan dari pihak yang tidak bertanggung jawab.

DAFTAR PUSTAKA

Albahara. 2005. *Analisis dan Desain Sistem Informasi*.Yogyakarta: Graha Ilmu.

Allamaraju, S. 2010. *RESTful Web Services Cookbook*. California: O'Reilly.

Allokendek, Frans N., Marwan Ghalib, dan Rachmat. 2014. *Pemanfaatan Web Services Dalam Sistem Informasi Daftar Pemilih Tetap*. [Skripsi]. Institut Teknologi Bandung. Bandung.

Andry. 2011. *Android A sampai Z*. Jakarta: Pcplus.

Azwar, S. 2011. *Sikap dan Perilaku. Dalam: Sikap Manusia Teori dan Pengukurannya. 2nd ed*. Yogyakarta: Pustaka Pelajar.

Berners LT, Fielding R dan Mesinter L. 2005. *Uniform Resource Identifier (URI): Generic Syntax*. The Internet Engineering Task Force (IETF), RFC3986.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Michael, C., Ferris, C., dan Orchard D. 2004. *Web Services Architecture - W3C Working Group Note 11 February 2004*. [Online]. Tersedia: https://www.w3.org/. Diakses pada tanggal 21 April 2017.

Brail, Graig dan Ramji, Sam. 2012. *OAuth - The Big Picture*. [Online]. Tersedia: http://apigee.com. Diakses pada tanggal 21 April 2017.

Breitman K, Casanova MA dan Truszkowski W. 2007. *Semantic Web: Concepts, Technologies and Applications*. London, UK: Springer.

Byod, Ryan. 2012. *Getting Started With OAuth 2*. California: O'Reilly Media.

Cerami, E. 2002. *Web Services Essential*. California: O'Reilly Media.

Chiragsh. 2012. *OAuth 2*. [Online]. Tersedia: https://code.google.com/p/google-api-php-client/wiki/OAuth2. Diakses pada tanggal 21 April 2017.

Christian, Charles. 2009. *Pengembangan Web Service Pengurai Morfolgi Bahasa Indonesia Pada Language Grid*. [Skripsi]. Universitas Indonesia. Jawa Barat.

Christie, Tom. 2017. *Django Rest Framework*. [Online]. Tersedia: http://www.django-rest-framework.org. Diakses pada tanggal 18 April 2017.

Clune, T.L. dan Rood, Richard.B. 2011. *Software Testing and Verification In Climate Model Development*. IEEE Journal, Focus: Climate Change Software. September-October, pp. 49-55.

Deviana, Harati. 2007. *Penerapan XML Web Service Untuk Sistem Distribusi Barang Studi Kasus: PT. Apotek Plus*. Palembang.

Developers, Android. 2014. *Android Developers*. [Online]. Tersedia :http://developer.android.com/index.html. Diakses pada tanggal 21 April 2017.

Developers, Google. 2017. *Using OAuth 2.0 to Access Google APIs*. [Online]. Tersedia: https://developers.google.com/accounts/docs/OAuth2. Diakses pada tanggal 21 April 2017.

Documentation, Django. 2017. *Django documentation*. [Online]. Tersedia: https://docs.djangoproject.com/en/1.11/. Diakses pada tanggal 20 April 2017.

Douglas, K. dan Douglas, S. 2005. *PostgreSQL*, Second Edition. Sams.

Elman, Julia dan Lavin, Mark. 2015. *Lightweight Django:USING REST, WEBSOCKETS & BACKBONE*. California: O'Reilly Media.

Evonove. 2013. Create *an OAuth2 Client Application*. [Online]. Tersedia: http://django-oauth-toolkit.readthedocs.io/. Diakses pada tanggal 21 April 2017.

Fadjar, M. dan Effendy, M. 1989. *Dunia Perguruan Tinggi dan Kemahasiswaan. Edisi Jurnal Dinamika Dotcom Vol 2. No. 1*) Dosen STMIK PPKIA Pradnya Paramita Malang 31 Pertama*. Malang: Pusat Publikasi dan Penerbitan Universitas Muhammadiyah.

Fajar, A. 2015. *Implementasi Restful Web Services Pada Sistem Informasi Wisata Alam dan Wisata Kuliner DIY.* [Skripsi]. Universitas Gajah Mada. Yogyakarta.

Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach P., dan Barners-Lee, T. 1999. *Hypertext Transfer Protocol -- HTTP/1.1*. [Online]. Tersedia: http://www.ietf.org/rfc/rfc2616.txt. Diakses pada tanggal 21 April 2017.

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

Filho, O.F.F. dan Ferreira, M.A.G.V. 2009. *Semantic Web Service: A RESTful Approach*. University of São Paulo, Polytechnic School São Paulo, Brazil.

Fowler, Martin. 2004. *UML Distilled Panduan Singkat Bahasa pemodelan Objek Standar, Edisi 3*. Yogyakarta: Andi Publishing.

Gottschalk, K., Graham, S., Kreger, H. dan Snell J. 2002. *Introduction to Web Services Architecture*. IBM System Journal: New Developments in Web Services and E-Commerce.

Gregorio J., Fielding R., Hadley M., Nottingham, M., dan Orchard, D. 2012. *Uri Template*. [Online]. Tersedia: http://tools.ietf.org/html/rfc6570. Diakses pada tanggal 21 April 2017.

Group, PostgreSQL Global Development. 1996. *PostgreSQL History*. [Online] . Tersedia: https://www.postgresql.org/. Diakses pada tanggal 21 April 2017.

Hamad H., Saad M. dan Abed R. 2010. *Performance Evaluation of RESTful Web Services for Mobile Devices*. Computer Engineering Department. Islamic University of Gaza, Palestine.

Hammer, Eran. 2009. OAuth. [Online]. Tersedia: http://hueniverse.com/oauth/. Diakses pada tanggal 21 April 2017.

Hardt, D. 2012. *The* OAuth *2.0 Authorization Framework*. [Online]. Tersedia: http://tools.ietf.org/html/draft-ietf-oauth-v2-31. Diakses pada tanggal 21 April 2017.

Hartono, Jogiyanto. 2007. *Metode Penelitian Bisnis: Salah Kaprah dan Pengalaman-Pengalaman*. Yogyakarta: BPFE.

Higashino WA, Toledo BF dan Capretz MAM. 2009. *REST and Resource Oriented Architecture*. Proceedings First International Symposium on Services Science ISSS'09, Logos, Berlin.

Hourieh, Ayman. 2008. *Learning Website Development with Django*. California: Birmingham: Packet Publishing.

Jiang, F. dan Lu,Y. 2012. *Software testing model selection research based on yinyang testing theory*. In: IEEE Proceeding of International Conference on Computer Science and Information Processing (CISP), pp. 590-594.

Johnson, Ralph E. 1997. *Components, Frameworks, Patterns*. [Online]. Tersedia: http://www.inf.ufsc.br/~ricardo.silva/download/johnson97components.pdf. Diakses pada tanggal 21 April 2017.

Jubilee Enterprise. 2017. *MASTERING PYTHON*. Yogyakarta: JUD (JUBILEE DIGITAL).

Kaur, G. dan Aggarwal, D. 2013. *A Survey Paper On Social Sign On Protocol OAuth 2.0. Journal*. Blue Ocean Research Journals, Punjab.

Kementerian Komunikasi dan Informatika. 2015. *Indonesia Raksasa Teknologi Digital Asia*. [Online]. Tersedia: https://www.kominfo.go.id/. Diakses pada tanggal 12 April 2017.

Kroll, Per. dan MacIsaac, Bruce. 2006. *Agility and Discipline Made Easy: Practices from OpenUP and RUP*. Massachussets: Pearson Education, Inc.

Lampung, Universitas. 2014. *Siakad Unila Beralih ke Generasi Baru*. [Online]. Tersedia: https://www.unila.ac.id/. Diakses pada tanggal 13 April 2017.

Larman, Craig. 2002. *Applying UML and Patterns: An Introduction to ObjectOriented Analysis and Design and the Unified Process, 2nd Edition*. New Jersey: Prentice-Hall, Inc.

Lee, W. M. 2011. *Beginning Android Application Development*. New York City: Wiley Publishing, Inc.

Matthew, N. dan Stones, R. 2005. *Beginning Databases with PostgreSQL*. New York: Kinetic Publishing Services, LLC.

McLeod. 2004. *Sistem Informasi Manajemen*. Jakarta: PT. Indeks.

Meildy, Bayu. 2014. *Daftar Simbol*. [Online]. Tersedia: http://elib.unikom.ac.id/download.php?id=83238. Diakses pada tanggal 17 April 2017.

Parecki, Aaron. 2012. *OAuth 2*. [Online]. Tersedia: *http://oauth.net/2/*. Diakses pada tanggal 21 April 2017.

Parecki, Aaron. 2012. *OAuth 2 Simplified*. [Online]. Tersedia: http://aaronparecki.com/articles/2012/07/29/1/oauth2-simplified. Diakses pada tanggal 21 April 2017.

Pautasso C., Zimmermann O. dan Leymann F. 2008. *RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision*. WWW 2008 /Refereed Track: Web Engineering - Web Services Deployment. Beijing, China.

PostgreSQL, Wiki. 2017. *Documentation PostgreSQL*. [Online]. Tersedia: https://wiki.postgresql.org/. Diakses pada tanggal 21 April 2017.

Pressman, Roger S. 2001. *Software Engineering A Practitioner's Approach Fifth Edition*. , New York: McGraw-Hill Companies, Inc.

Pressman, Roger S. 2005. *Software Engineering, sixth edition*.. McGraw-Hill Series in Computer Science.

Psycopg. 2001. *Psycopg – PostgreSQL database adapter for Python*. [Online]. Tersedia: http://initd.org/psycopg/. Diakses pada tanggal 21 April 2017.

Richardson L. dan Ruby S. 2007. *Restful web service*.California: O'Reilly Media.

Roth, George. 2009. *RESTful HTTP in practice*. [Online]. Tersedia: https://www.infoq.com/. Diakses pada tanggal 21 April 2017.

Safaat, Nazruddin H. 2012. Pemograman Aplikasi *Mobile*. Smartphone dan Tablet PC Berbasis Android (Edisi Revisi). Bandung: Informatika.

Singh, Navdeep. 2016. Study of Google Firebase API for Android. *International Journal of Innovative Research in Computer and Communication Engineering: Vol. 4, Issue 9.*

Sommerville I. 2010. *Software engineering / Ian Sommerville. — 9th ed*. Boston, Massachusetts: Addison-Wesley.

Suyanto, Asep H. 2007. Web Design Theory And Practices, Yogyakarta: Andi Publisher.

Swaroop, H. 2005. *A Byte of Python.Creative Commons Attribution-NonCommercial-ShareAlike License 2.0* . [Online]. Tersedia: www.byteofpython.info. Diakses pada tanggal 17 April 2017.

Tapiador, A, Victor, S, Joaquin, S. 2012. *An Analysis Of Social Network Connect Services*. [Tesis]. Universidad Politecnica de Madrid, Spain.

Tech in Asia. 2017. *Perkembangan Pengguna Internet di Indonesia Tahun 2016 Terbesar di Dunia*. [Online]. Tersedia: https://id.techinasia.com/. Diakses pada tanggal 12 April 2017.

Tidwell, D. 2001. *Web Services: The Web's next Revolution*. [Online]. Tersedia: http://www6.software.ibm.com/developerworks/education/wsbasics/wsbasics-a4.pdf. Diakses pada tanggal 17 April 2017.

TIOBE. 2017. *TIOBE Index for April 2017*. [Online]. Tersedia: www.tiobe.com. Diakses pada tanggal 12 April 2017.

Yitnosumarto. 2006. *Metode Penelitian Kuantitatif dan Kualitatif*. Yogyakarta: Graha Ilmu.

Yosrinal. 2014. *Perancangan dan Implementasi Resource Server dan Authorization Server Menggunakan Teknologi Otentikasi OAuth 2*. [Skripsi]. Universitas Sumatra Utara. Sumatra Utara.