



Dokumentasi Produk Tugas Akhir

Lembar Sampul Dokumen

Judul Dokumen	TUGAS AKHIR TEKNIK ELEKTRO: <i>Internet of Things: Personalisasi Informasi dan Promosi</i> di Lingkungan Kampus dengan iBeacon
Jenis Dokumen	IMPLEMENTASI <small>Catatan: Dokumen ini dikendalikan penyebarannya oleh Program Studi Teknik Elektro ITB</small>
Nomor Dokumen	B400-03-TA1516.01.003
Nomor Revisi	03
Nama File	B400-03-TA1516.01.003.docx
Tanggal Penerbitan	4 Mei 2016
Unit Penerbit	Program Studi Teknik Elektro Institut Teknologi Bandung
Jumlah Halaman	77 <small>(termasuk lembar sampul ini)</small>

Data Pengusul dan Pembimbing				
Pengusul	Nama	Astari Purnomo	Jabatan	Anggota
	Tanggal	4 Mei 2016	Tanda Tangan	
	Nama	Rizky Indra Syafrian	Jabatan	Anggota
	Tanggal	4 Mei 2016	Tanda Tangan	
	Nama	Adirga Ibrahim Khairiy	Jabatan	Anggota
	Tanggal	4 Mei 2016	Tanda Tangan	
Pembimbing	Nama	Ir. Emir Mauludi Husni, M.Sc., Ph.D.	Jabatan	Dosen Pembimbing
	Tanggal	4 Mei 2016	Tanda Tangan	

DAFTAR ISI

CATATAN SEJARAH PERBAIKAN	4
PENGANTAR	5
1.1 Ringkasan Isi Dokumen	5
1.2 Tujuan Penulisan, Aplikasi, dan Fungsi Dokumen	5
1.3 Referensi	5
1.4 Daftar Singkatan.....	6
IMPLEMENTASI.....	8
2.1 Implementasi Sub-sistem <i>Back-end Application</i>	8
2.1.1 Pendahuluan	8
2.1.2 Struktur Implementasi	9
2.1.3 <i>Environment</i>	22
2.1.4 Prosedur Implementasi	23
2.1.5 Progres Implementasi	24
2.1.6 Permasalahan dan Solusi	28
2.2 Implementasi Sub-sistem <i>Mobile Application</i>	29
2.2.1 Pendahuluan	29
2.2.2 Struktur Implementasi	29
2.2.3 <i>Environment</i>	31
2.2.4 Prosedur Implementasi	32
2.2.5 Progres Implementasi	32
2.2.5.1 Pemilihan perangkat android yang kompatibel dengan spesifikasi yang.....	32
dibutuhkan.	32
2.2.5.2 Membangun <i>mobile application</i> berbasis android menggunakan Android Studio. ...	32
2.2.5.3 Fitur <i>Bluetooth Low Energy</i> (BLE) untuk pemindaian(<i>scan</i>) beacon.	32
2.2.5.4 Interaksi <i>mobile application</i> dengan server IBM Bluemix melalui MQTT.....	35
2.2.5.5 Fitur penyimpanan dan pengelolaan data pada <i>local memory</i> smartphone.....	37
2.2.5.6 <i>Graphic User Interface</i> (GUI) sebagai interaksi antarmuka dengan pengguna.	41
2.2.5.7 Membangun fitur personalisasi penggunaan aplikasi dalam bentuk akun pengguna	
(<i>user account</i>).	48
2.2.5.8 <i>Push-Notification</i> sebagai fitur personalisasi pengguna.	59
2.2.5.9 Personalisasi dengan fitur presensi mahasiswa dalam kelas perkuliahan..	64
2.2.6 Permasalahan dan Solusi	66
2.3 Implementasi Sub-sistem <i>Beacon</i>	66
2.3.1 Pendahuluan	66
2.3.2 Struktur Implementasi	66

2.3.3	<i>Environment</i>	67
2.3.4	Prosedur Implementasi	67
2.3.5	Progres Implementasi	68
2.3.6	Permasalahan dan Solusi	69
2.4	Lampiran	70
2.4.1	Lampiran <i>Source Code Back-end Application Project</i>	70

CATATAN SEJARAH PERBAIKAN

Tabel I – Catatan Sejarah Perbaikan

Versi	Tanggal	Penyunting	Perbaikan
01	11 Maret 2016	Astari Purnomo Rizky Indra Syafrian Adirga Ibrahim Khairy	Rilis dokumen versi 01
02	8 April 2016	Rizky Indra Syafrian	Penambahan IBM Presence Insight pada bagian <i>back-end application</i>
03	4 Mei 2016	Astari Purnomo Rizky Indra Syafrian Adirga Ibrahim Khairy	Penambahan fitur personalisasi pengguna Penambahan fitur <i>attend class</i> Penambahan fitur <i>beacon monitoring</i> Pengubahan <i>layout homepage</i> aplikasi Penambahan <i>attendance monitoring system</i> dan <i>notification center</i>

PENGANTAR

1.1 Ringkasan Isi Dokumen

Dokumen ini berisi implementasi dari rancangan proyek tugas akhir yang berjudul Personalisasi Informasi dan Promosi di Lingkungan Kampus dengan iBeacon. Dokumen ini diajukan sebagai salah satu syarat untuk memenuhi proyek Tugas Akhir II Teknik Elektro ITB tahun 2015-2016.

Implementasi yang dijelaskan dalam dokumen ini meliputi implementasi sub-sistem aplikasi Android, sub-sistem *back-end application*, dan sub-sistem *beacon*. Dokumen ini terdiri dari beberapa bagian, yaitu struktur implementasi, *environment*, prosedur implementasi, progress implementasi, serta permasalahan dan solusi.

1.2 Tujuan Penulisan, Aplikasi, dan Fungsi Dokumen

Penulisan dokumen B400 ini memiliki tujuan sebagai berikut.

1. Dokumentasi tahap implementasi Personalisasi Informasi dan Promosi di Lingkungan Kampus dengan iBeacon berdasarkan rancangan dan spesifikasi yang telah dibuat.
2. Menjadi panduan dalam pelaksanaan kerja dan pengambilan keputusan dalam pengembangan sistem lebih lanjut.
3. Sebagai salah satu dokumentasi proyek tugas akhir Personalisasi Informasi dan Promosi di Lingkungan Kampus dengan iBeacon.

1.3 Referensi

1. International Business Machines (IBM), Corp. 2016. *IBM Bluemix Documentation*. IBM Bluemix. (Online). <https://console.ng.bluemix.net/docs/>. Diakses 8 Maret 2016 16:30.
2. Tutorials Points. 2016. *Node.js Tutorial*. (Online). <http://www.tutorialspoint.com/nodejs/>. Diakses 8 Maret 2016 16:30.
3. Rudd, Adam, Matteo Collina, and Maxime Agor. 2016. *A library for the MQTT protocol*. Node Package Manager (npm), Inc. (Online). <https://www.npmjs.com/package/mqtt>. Diakses 8 Maret 2016 16:30.
4. StrongLoop, IBM, and other expressjs.com contributors. *Express Framework Node.js Guide*. Node.js Foundation. (Online). <http://expressjs.com/en/guide/>. Diakses 8 Maret 2016 16:30.
5. International Business Machines (IBM), Corp. 2016. *Cloudant Node.js CRUD*. IBM Cloudant. (Online). <https://github.com/cloudant/nodejs-cloudant>. Diakses 8 Maret 2016 16:30.
6. Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, Sweta Rawat, Nagesh Subrahmanyam, Rong Xiang, Gerald Kallas, Neeraj Krishna, Stefan Fassmann, Martin Keen, Dave Locke. 2012. *IBM Redbooks Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. USA : International Business Machines Corporation (IBM Corp).
7. Google, Inc. 2015. *Processes and Application Life Cycle*. Android Developer (Online). <http://developer.android.com/guide/topics/processes/process-lifecycle.html>. Diakses 4 Januari 2015 14.00.

8. Google, Inc. 2015. *Processes and Thread*. Android Developer (Online). <http://developer.android.com/guide/components/processes-and-threads.html>. Diakses 4 Januari 2015 14.00.
9. Google, Inc. 2015. *Connectivity Manager* (Online). <http://developer.android.com/reference/android/net/ConnectivityManager.html>. Diakses 4 Januari 2015 14.00.
10. Google, Inc. 2015. *View* (Online). [http://developer.android.com/reference/android/view/View.html#isEditMode\(\)](http://developer.android.com/reference/android/view/View.html#isEditMode()). Diakses 4 Januari 2015 14.00.
11. IBM Corporation. 2015. *IA92: WBI Brokers - Java implementation of WebSphere MQ Telemetry transport*. IBM Corporation (Online). http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24006006&loc=en_US&cs=utf-8&lang=en. Diakses 4 November 2015 16:00.
12. IBM Corporation. 2015. *Using MQ Telemetry Transport protocol in IBM Worklight mobile applications*. IBM Corporation (Online). http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24033580&loc=en_US&cs=utf-8&lang=en. Diakses 4 November 2015 16:00.
13. GitHub, Inc. 2016. *IoT Starter for Android*. GitHub, Inc (Online). <https://github.com/ibm-messaging/iot-starter-for-android>. Diakses pada 12 Januari 2016 17.00
14. GitHub, Inc. 2016. *Android MQTT Demo*. GitHub, Inc (Online). <https://github.com/jeffprestres/AndroidMQTTDemo>. Diakses pada 12 Januari 2016 17.00
15. Dc-square GmbH. 2016. *MQTT Client Library Encyclopedia – Paho Android Service*. HiveMQ Enterprise MQTT Broker (Online). <http://www.hivemq.com/blog/mqtt-client-library-encyclopedia-paho-android-service>. Diakses pada 12 Januari 2016 17.00
16. Dc-square GmbH. 2016. *MQTT Security Fundamentals: Authorization*. HiveMQ Enterprise MQTT Broker (Online). <http://www.hivemq.com/blog/mqtt-security-fundamentals-authorization/>. Diakses pada 12 Januari 2016 17.00
17. Auth0, Inc. 2013-2015. *Authenticating & Authorizing Devices using MQTT with Auth0*. Auth0, Inc (Online). <https://auth0.com/docs/scenarios/mqtt>. Diakses pada 12 Januari 2016 17.00
- IBM Corporation. 2015. *Learning Center-Introducing IBM Cloudant*. IBM Corporation (Online). <https://cloudant.com/learning-center/>. Diakses 4 November 2015 16:00.

1.4 Daftar Singkatan

Berikut ini diberikan tabel referensi dari singkatan-singkatan yang digunakan dalam dokumen ini.

Tabel II – Daftar Singkatan

Singkatan	Arti
IoT	Internet of Things
BLE	Bluetooth Low Energy
UUID	Universally Unique Identifier
GUI	Graphical User Interface
SDK	Software Development Kit
IDE	Integrated Development Environment

IBM	International Business Machines
REST	Representative State Transfer
API	Application Program Interface
MAC	Media Access Control
RSSI	Radio Signal Strength Indicator
PI	Presence Insights
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
SQL	Structured Query Language
PCB	Printed Circuit Board
IDR	Indonesia Rupiah
ITB	Institut Teknologi Bandung
AJAX	Asynchronous Javascript and XML
XML	Extensible Markup Language
MQTT	Message Queuing Telemetry Transport
JSON	JavaScript Object Notation

IMPLEMENTASI

2.1 Implementasi Sub-sistem *Back-end Application*

Berikut ini tabel gambaran umum dari sub-sistem *back-end application*.

Tabel III – Gambaran umum sub-sistem *Back-end Application*

Blok	Fungsi
Fungsi	<ul style="list-style-type: none">• Mengolah data pengguna (<i>user</i>) aplikasi Android• Mengolah data <i>beacon</i> dan mengembalikannya menjadi informasi <i>beacon</i> yang sesuai• Mengolah data pengguna yang mengirimkan data <i>beacon</i> kelas menjadi daftar hadir sistem absensi• Mengirimkan notifikasi kepada pengguna sesuai dengan kegiatannya• Mengolah <i>traffic</i> pengguna aplikasi yang melewati daerah-daerah yang telah dipasang <i>beacon</i>.
Input	<ul style="list-style-type: none">• Data registrasi pengguna• Data <i>beacon</i> yang terdeteksi oleh <i>bluetooth</i> pada <i>smartphone</i> pengguna• Data <i>beacon</i> yang terdeteksi oleh <i>bluetooth</i> pengguna <i>smartphone</i> yang terdaftar pada kelas kuliah tertentu• Kiriman notifikasi dari <i>back-end application</i> melalui antarmuka <i>front-end</i> yang digunakan oleh <i>administrator</i>• Informasi <i>beacon</i> yang telah dimasukkan ke <i>database</i> oleh <i>administrator</i>
Output	<ul style="list-style-type: none">• Data registrasi pengguna yang telah disimpan pada <i>database</i> dan dapat diubah sewaktu-waktu oleh pengguna• Informasi tentang <i>beacon</i> yang terdeteksi dikirim ke <i>smartphone</i> pengguna• Daftar peserta kelas yang hadir di kelas kuliah karena <i>smartphone</i>-nya yang mendeteksi <i>beacon</i>• Informasi notifikasi pada <i>smartphone</i> pengguna• Data dan hasil analisis dari IBM Presence Insights
Kebutuhan Kuantitatif	<ul style="list-style-type: none">• IBM Bluemix sebagai <i>server</i> tempat berjalannya <i>back-end application</i> dan antarmuka <i>front-end application</i> yang digunakan oleh <i>administrator</i>.
Deskripsi kebutuhan performansi	<ul style="list-style-type: none">• Dapat melakukan pengolahan data registrasi pengguna• Dapat melakukan pengolahan data <i>beacon</i> dan mengirimkan kembali informasi yang sesuai• Dapat mengolah data <i>beacon</i> menjadi daftar hadir sistem absensi kelas• Dapat mengirimkan notifikasi kepada pengguna aplikasi Android• Dapat mengolah <i>traffic</i> pengguna aplikasi selama berada di sekitar daerah-daerah yang telah dipasang <i>beacon</i>.

2.1.1 Pendahuluan

Sub-sistem *back-end application* menangani semua *request* informasi yang masuk dari aplikasi Android mulai dari sistem *sign-in* sampai dengan permintaan informasi sesuai dengan *beacon* yang terdeteksi oleh *smartphone*. *Back-end application* juga tempat semua data pengguna dan data informasi di simpan. Penyimpanan data tersebut dilakukan pada sebuah *database*.

Front-end yang dibangun akan digunakan oleh *administrator* sebagai jembatan antara *administrator* dengan *back-end application*. Komunikasi antara *front-end* dengan *back-end* menggunakan AJAX (Asynchronous Javascript and XML). Dengan *front-end*, *administrator* dapat melakukan pengiriman notifikasi tanpa mengubah *source code* program *back-end application*. Daftar hadir sistem absensi yang diolah oleh *back-end application* juga dikirimkan dan ditampilkan ke *front-end* menggunakan AJAX.

Protokol *request* yang digunakan adalah MQTT (*Message Queuing Telemetry Transport*). Melalui protokol ini, sebuah pesan dikirim melalui MQTT *broker*. Di dalam *broker* tersebut terdapat berbagai nama *topic* yang dapat di-*subscribe* dan *publish*. *Topic* yang digunakan berbeda-beda untuk setiap jenis *request* informasi yang dilakukan.

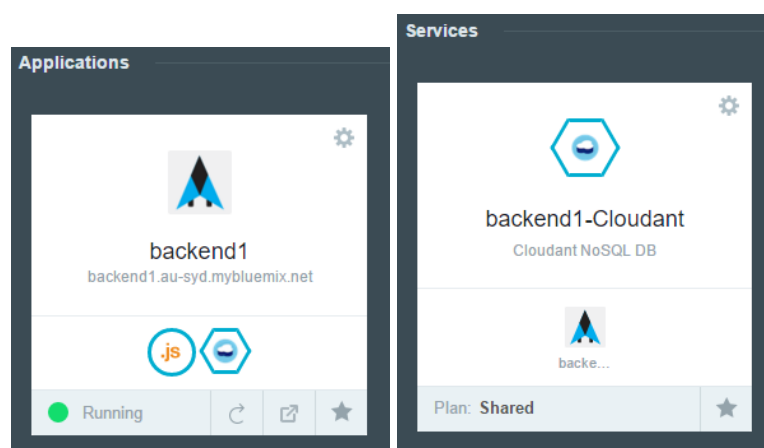
Back-end application menerima segala bentuk *request* yang masuk dalam bentuk paket data objek JSON (*JavaScript Object Notation*). JSON yang masuk kemudian di-*parsing* sesuai dengan kebutuhan untuk kemudian diproses lebih lanjut. *Response* balasan dari *back-end application* juga dikirimkan kembali ke aplikasi Android dalam bentuk JSON.

Data yang dikirimkan dari *back-end application* sebagai *response* dari *request* aplikasi Android diambil dari *database*. *Database* menyimpan data registrasi user dan data informasi *beacon* per UUID-major-minor. Ketika pengguna aplikasi Android melakukan *sign-in* ataupun *sign-up*, *back-end application* akan melakukan pencocokan dan penambahan pada *database*. Informasi *beacon* per UUID-major-minor diatur oleh *administrator*.

2.1.2 Struktur Implementasi

Back-end Application

Sub-sistem *back-end application* merupakan program yang bekerja pada *server* dan menerima *request* dari aplikasi Android. Tugas utama *back-end application* adalah menerima data *sign-in/sign-up* pengguna, menerima data *beacon*, mengembalikan status berhasil/gagalnya *sign-in/sign-up*, dan mengembalikan data informasi *beacon* per UUID-major-minor.



Gambar 1 *Back-end Application* dan *service* nya, terdiri dari Node.js dan IBM Cloudant NoSQL *database*. Tampilan ini ada pada konfigurasi IBM Bluemix via *web browser*. ©Dokumentasi Penulis

Back-end application dijalankan pada *server* IBM Bluemix dengan Node.js Runtime. Agar *back-end application* dapat menjalankan tugasnya, maka dalam pembuatannya perlu disertakan modul-modul Node.js untuk membantu memudahkan

penggunaan fungsi-fungsi tertentu. Berikut ini adalah modul Node.js yang digunakan.

Tabel IV – Modul Node.js

Node.js Module	Fungsi
Express framework	Express framework merupakan modul tambahan untuk implementasi Node.js sebagai <i>web</i> dan <i>mobile application</i> . Fungsi yang digunakan dari modul ini adalah agar <i>back-end application</i> yang dibuat dapat diakses via internet
Cloud foundry	Cloud foundry merupakan modul untuk <i>deploy</i> program yang telah dibuat ke <i>server</i> . Fungsinya adalah untuk menyesuaikan <i>environment</i> program dengan <i>server</i> tujuan <i>deploy</i> .
MQTT	MQTT merupakan modul untuk melakukan komunikasi dengan protokol MQTT via internet. Fungsi <i>publish</i> dan <i>subscribe</i> dilakukan dengan modul ini.
Nano	Nano merupakan modul untuk melakukan akses <i>database</i> . Pada modul ini terdapat berbagai <i>syntax</i> untuk akses <i>database</i> (read, write, delete, dll).
Cloudant	Cloudant merupakan modul untuk autentikasi akses <i>database</i> di IBM Cloudant NoSQL Database.
HTTPS	HTTPS digunakan untuk melakukan HTTPS POSTS dari <i>back-end application</i> ke IBM Presence Insights
Body-Parser	Body-Parser digunakan untuk melakukan <i>parsing</i> JSON

Komunikasi *back-end application* dengan aplikasi Android menggunakan protokol MQTT via internet. Untuk melakukan komunikasi, baik *back-end application* maupun aplikasi Android harus melakukan *publish* dan *subscribe* ke sebuah *topic* pada MQTT Broker. *Topic* yang digunakan harus berbeda-beda agar tidak terjadi miskomunikasi antara *back-end application* dengan aplikasi Android dengan pengguna yang berbeda-beda.

Untuk pengiriman data *sign-up* pengguna, dilakukan pengiriman paket data JSON berisi data-data pengguna oleh aplikasi Android. Paket data ini dikirimkan ke *back-end application* melalui *topic* “ubeacon/user/signup”. Aplikasi Android melakukan *publish* paket data JSON ke *topic* ini, kemudian *back-end application* melakukan *subscribe* ke *topic* yang sama agar dapat menerima paket data JSON.

Pengiriman kembali status berhasilnya *sign-up* dilakukan oleh *server* ke aplikasi Android. Status dikirimkan melalui *topic* “ubeacon/user/signup/<username>” dengan *username* diambil dari isian pengguna pada saat melakukan *sign-up*. *Topic* yang digunakan harus berbeda agar *back-end application* mengirimkan status kepada *requester*.

Skema yang sama juga digunakan untuk *sign-in*. Paket data JSON berisi *username* dan *password* pengguna dikirimkan dari aplikasi Android ke *back-end application* menggunakan *topic* “ubeacon/user/signin”, kemudian *back-end application* mengembalikan status melalui *topic* “ubeacon/user/signin/<username>”.

Untuk melakukan perubahan data pengguna melalui menu *edit profile* skema yang digunakan sama dengan *sign-up*, hanya saja untuk *edit profile* ada data tambahan berupa *password* lama dan *password* baru. *Topic* yang digunakan untuk mengirim data yang akan diubah adalah “ubeacon/user/editprofile/”, untuk menerima balasan dari *back-end application* adalah “ubeacon/user/editprofile/<username>”.

Pengiriman data *beacon* dari aplikasi Android ke *back-end application* dilakukan melalui *topic* “ubeacon/request”. Skema yang sama juga digunakan untuk pengiriman informasi *beacon* per UUID-major-minor, yaitu dilakukan melalui *topic* “ubeacon/user/response/<username>”.

Berikut ini adalah tabel penggunaan *topic* pada setiap komunikasi yang dilakukan oleh *back-end application*.

Tabel V – Penggunaan *topic* pada MQTT *broker* untuk setiap fungsi

Fungsi	Android app publish to Back-end subscribe to	Backend publish to Android app subscribe to
<i>Sign-up</i>	ubeacon/user/signup	ubeacon/user/signup/<username>
<i>Sign-in</i>	ubeacon/user/signin	ubeacon/user/signin/<username>
<i>Edit profile</i>	ubeacon/user/editprofile	ubeacon/user/editprofile/<username>
Data dan informasi <i>beacon</i>	ubeacon/request	ubeacon/user/response/<username>

Semua data yang dikirimkan antara *back-end application* dan aplikasi Android dipaketkan dalam JSON. Format JSON untuk setiap data yang dibawa juga berbeda-beda. Berikut ini adalah format JSON yang digunakan untuk membawa data.

Tabel VI – Penggunaan format JSON untuk setiap fungsi

Fungsi	JSON Format
<i>Sign-up</i>	<pre>{ "_id": "rizkyindra", "password": "asdfghjkl;", "fullname": "Rizky Indra Syafrian", "email": "rzkyndr@gmail.com", "birthdate": "1994-08-02", "occupation": "Mahasiswa", "interest": "Seminar" }</pre>
<i>Sign-in</i>	<pre>{ "_id": "rizkyindra", "password": "asdfghjkl;" }</pre>
<i>Edit profile</i>	<pre>{ "_id": "rizkyindra", "oldpassword": "asdfghjkl;", "password": "asdfghjkl", "fullname": "Rizky Indra Syafrian", "email": "rzkyndr@gmail.com", "birthdate": "1994-08-02", "occupation": "Mahasiswa", "interest": "Seminar" }</pre>
<i>Beacon data</i>	<pre>{ "bnm": [{ "descriptor": "aa:bb:cc:dd:ee:ff", "detectedTime": "2016-02-02T11:00:00.000Z", "data": { "proximityUUID": "cb10023f-a318-3394-4199-a8730c7c1aec", "major": "284", "minor": "1", "rssi": 69, "accuracy": 75, "proximity": "Immediate" } }] }</pre>
<i>Beacon information</i>	<pre>{ "_id": "cb10023f-a318-3394-4199-a8730c7c1aec-284-1", "_rev": "8-f0c5ec74cbc17c2e69aa74b40405f322", "proximityUUID": "cb10023f-a318-3394-4199-a8730c7c1aec", "major": "284", "minor": "1", "rssi": 69, "accuracy": 75, "proximity": "Immediate" }</pre>

	<pre> "minor": "1", "image_url": "http://backend1.au-syd.mybluemix.net/images/sampleimage1.jpg", "category": "Sample category 1", "brief": "This is sample brief description for event 1", "full": "This is sample long description of the event 1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer feugiat dapibus euismod. Aliquam ac dui non quam ornare tincidunt et consectetur nunc.", "time": "2016-05-01T08:00:00.000Z", "location": "Sample location 1", "organizer": "Sample organization 1", "contact": "Sample contact 1: 081234567890" } </pre>
Sign-up reply status	<pre> { "status": "Registration Success" } or { "status": "Username taken" } </pre>
Sign-in reply status	<pre> { "status": "Sign-in success" } or { "status": "Wrong password" } or { "status": "Username not exist" } </pre>
Edit profile reply status	<pre> { "status": "Data successfully changed" } or { "status": "Wrong password" } </pre>

Ketika menerima data registrasi pengguna, *back-end application* memeriksa *username* yang diregisterasikan pengguna sudah terpakai atau belum. Jika *username* yang diregisterasikan sudah terpakai maka akan ada balasan bahwa *username* sudah dipakai. Jika belum maka registrasi akan sukses.

Berikut ini adalah potongan *source code* untuk bagian *sign-up*.

```

case 'ubeacon/user/signup':
  // Parse incoming message
  var signup_1 = message.toString();
  var signup_2 = JSON.parse(signup_1);
  var signup_3 = signup_2._id;
  // Read 'user' database
  var db_user = cloudant.db.use('user');
  var user_topic_signup = "ubeacon/user/signup/" + signup_3;
  db_user.get(signup_3, function(err, req, res){
    // Conditional: If username not used, registration success, otherwise failed
    if (!req) {
      db_user.insert(signup_2, function(err, req, res) {
        console.log("Registration success");
      });
      var response = "{\"status\":\"Registration success\"}";
      client.publish(user_topic_signup, response, function(){
        console.log("Response sent");
      });
    } else {
      console.log("Username taken");
      var response = "{\"status\":\"Username taken\"}";
      client.publish(user_topic_signup, response, function(){
        console.log("Response sent");
      });
    }
  });
  ToPresenceInsight_RegisterDevices(signup_2);
  break;

```

Ketika menerima data *sign-in* pengguna, *back-end application* akan memeriksa *username* yang dimasukkan pengguna berada pada *database*. Jika *username* ada pada *database*, maka akan diperiksa kecocokan *password* yang dimasukkan. Jika keduanya benar maka pengguna akan berhasil *sign-in*. Jika pemeriksaan *username* dan/atau *password* gagal, maka akan ada balasan *username* tidak ada atau *password* salah.

Berikut ini adalah potongan *source code* untuk bagian *sign-in*.

```
case 'ubeacon/user/signin':
    // Parse incoming message
    var signin_1 = message.toString();
    var signin_2 = JSON.parse(signin_1);
    var signin_3 = signin_2._id;
    var signin_4 = signin_2.password;
    // Read 'user' database
    var db_user = cloudant.db.use('user');
    var user_topic_signin = "ubeacon/user/signin/" + signin_3;
    db_user.get(signin_3, function(err, req, res){
        // Conditional: If username and password match success, otherwise failed
        if (req) {
            if (req.password === signin_4) {
                console.log("Sign-in success");
                var response = "{\"status\":\"Sign-in success\"}";
                client.publish(user_topic_signin, response, function(){
                    console.log("Response sent");
                });
            } else {
                console.log("Wrong password");
                var response = "{\"status\":\"Wrong password\"}";
                client.publish(user_topic_signin, response, function(){
                    console.log("Response sent");
                });
            }
        } else {
            console.log("Username not exist");
            var response = "{\"status\":\"Username not exist\"}";
            client.publish(user_topic_signin, response, function(){
                console.log("Response sent");
            });
        }
    });
    break;
```

Ketika pengguna mengubah datanya melalui menu *edit profile* pada aplikasi Android, data baru akan dikirimkan ke *back-end application*. Data baru ini berisikan data tambahan berupa *password* lama dan *password* baru. *Password* lama akan dicocokkan, apabila sesuai maka *password* lama akan dihilangkan dan data-data lain yang mengikut akan diubah. Apabila *password* salah, perubahan data tidak bisa dilakukan.

Berikut ini adalah potongan *source code* untuk melakukan *edit profile*.

```
case 'ubeacon/user/editprofile':
    // Parse incoming message
    var edit_1 = message.toString();
    var edit_2 = JSON.parse(edit_1);
    var edit_3 = edit_2._id;
    var edit_4 = edit_2.oldpassword;
    // Read 'user' database
    var db_user = cloudant.db.use('user');
    var user_topic_edit = "ubeacon/user/editprofile/" + edit_3;
    db_user.get(edit_3, function(err, req, res){
        // Conditional: If username and password match success, otherwise failed
        if (req) {
            if (req.password === edit_4) {
                delete edit_2.oldpassword;
                edit_2._rev = req._rev;
                db_user.insert(edit_2, function(err, req, res) {
                    console.log("Data successfully changed");
                });
            }
        }
    });
```

```

        var response = "{\"status\":\"Data successfully changed\"}";
        client.publish(user_topic_edit, response, function(){
            console.log("Response sent");
        });
    } else {
        console.log("Authetctication Failed");
        var response = "{\"status\":\"Wrong password\"}";
        client.publish(user_topic_edit, response, function(){
            console.log("Response sent");
        });
    }
} else {
    console.log("Username not exist");
    var response = "{\"status\":\"Username not exist\"}";
    client.publish(user_topic_edit, response, function(){
        console.log("Response sent");
    });
}
});
break;

```

Untuk pengiriman komunikasi tentang data dan informasi *beacon*, aplikasi Android mengirimkan data *beacon* dengan format yang sudah dijelaskan. Kemudian *back-end application* menerima data *beacon* tersebut. Data yang sudah diterima diambil UUID-major-minor-nya. Kemudian dicari yang sesuai pada *database*. Jika ditemukan informasi yang sesuai pada *beacon*, maka informasi tersebut akan langsung dikirimkan kembali ke aplikasi Android.

Berikut ini adalah potongan *source code* untuk bagian penerimaan data *beacon* dan pengiriman informasi *beacon*.

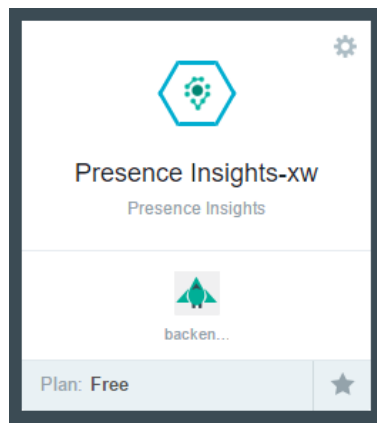
```

case 'ubeacon/user/request':
    // Parse incoming message
    var request_1 = message.toString();
    var request_2 = JSON.parse(request_1);
    console.log(request_1);
    // Extract UUID, major, minor
    var proximityUUID = request_2.bnm[0].data.proximityUUID;
    var majorUUID = request_2.bnm[0].data.major;
    var minorUUID = request_2.bnm[0].data.minor;
    var descriptor = request_2.bnm[0].descriptor;
    var UUID = (proximityUUID + '-' + majorUUID + '-' + minorUUID);
    // Read 'beacon' database
    var db_beacon = cloudant.db.use('beacon');
    var user_topic_response = "ubeacon/user/response/" + descriptor;
    db_beacon.get(UUID, function(err, req, res){
        var doc = JSON.stringify(req);
        // Send corresponding beacon database
        client.publish(user_topic_response, doc, function(){
            console.log("JSON sent");
        });
    });
    ToPresenceInsight_BeaconConnector(request_1);
    break;

```

IBM Presence Insights

IBM Presence Insight merupakan sebuah *service* dari IBM Bluemix yang digunakan untuk melihat data dan analitik dari perangkat yang dapat mendeteksi *beacon*. *Service* ini dapat ditambahkan melalui katalog IBM Bluemix dan dapat di-*binding* dengan *back-end application* yang sudah ada.



Gambar 2 IBM Presence Insights service. Tampilan ini ada pada konfigurasi IBM Bluemix via *web browser*. ©Dokumentasi Penulis

IBM Presence Insight (PI) digunakan untuk menerima kiriman paket data JSON dari aplikasi Android yang mendeteksi adanya *beacon* di sekitar *smartphone*. Paket data JSON yang dikirimkan berisi data mengenai *beacon* yang terdeteksi dan waktu terjadinya deteksi. Dari paket data JSON yang diterima ini, dapat dibuat data mengenai jumlah perangkat yang sedang berada di sekitar *beacon*, lamanya perangkat tersebut berada di sekitar *beacon*, para pengguna yang sedang berada di sekitar *beacon*, dan data lainnya yang dapat dikemas dalam sebuah statistik yang berguna untuk analisis pengunjung.

IBM Presence Insight sebenarnya digunakan sebagai penerima pertama data *beacon* yang dikirimkan oleh aplikasi Android. Setelah itu, data *beacon* yang masuk ke PI kemudian diteruskan ke *subscriber*. *Subscriber* merupakan *back-end application* yang menerima data terusan dari PI untuk kemudian diolah lebih lanjut.

Pada implementasi ini, protokol yang digunakan untuk pengiriman data dari aplikasi Android ke *server* adalah MQTT, sedangkan API (*Application Programming Interface*) PI sendiri hanya dapat menerima data melalui protokol HTTP. Untuk itu, skema pengiriman data ke PI perlu diubah. Pengiriman paket data JSON yang berisi data *beacon* akan dikirimkan oleh *back-end application*. Pada *back-end application*, paket data JSON yang masuk melalui protokol MQTT dikirimkan ulang ke PI dengan protokol HTTP.

Paket data JSON yang berisi data *beacon* dengan format standar dikirimkan ke PI melalui API-nya (*Application Programming Interface*). Pengiriman data *beacon* ke PI dilakukan dengan HTTP POSTS ke API *beacon connector* milik PI. Berikut ini adalah detail HTTP POSTS yang dikirimkan ke PI.

Tabel VII – Detail HTTP POSTS untuk pengiriman data ke PI

Key	Value
Hostname	presenceinsights.au-syd.bluemix.net
Port	443
Path	/conn-beacon/v1/tenants/1g4a05cq/orgs/my1u08tp
Method	POST
Header	Content-Type: application/json Authorization: Basic NTJnMDBrBtpiNXhQdzZKLW5USnE=
Body	<pre>{ "bnm": [{ "descriptor": "aa:bb:cc:dd:ee:ff", "detectedTime": "2016-02-02T11:00:00.000Z", "data": { "proximityUUID": "cb10023f-a318-3394-4199-a8730c7c1aec", "major": "284",</pre>


```

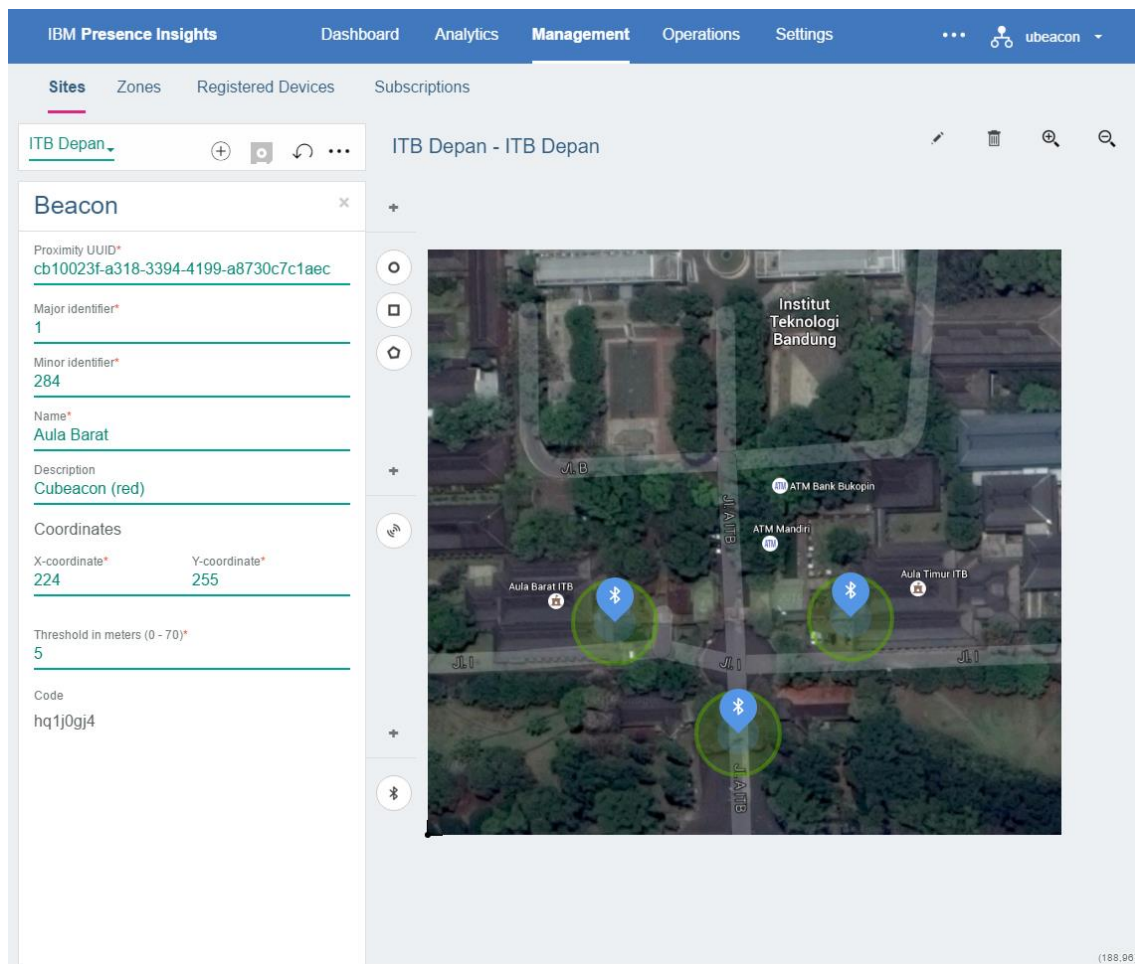
        "minor": "1",
        "rssi": 69,
        "accuracy": 75,
        "proximity": "Immediate"
    }
  ]
}

```

Respons yang diberikan oleh PI ke *back-end application* berupa *status code*. Bila respons yang diterima berupa *status code* 202, artinya data *beacon* telah berhasil diterima PI. Apabila respons yang diterima berupa *status code* 400, artinya ada kesalahan pada format JSON yang dikirimkan.

Sebelumnya pada PI, konfigurasi *beacon* yang digunakan telah dimasukkan. Konfigurasi pada PI meliputi peta tempat *beacon* akan ditempatkan, titik-titik tempat *beacon* diletakkan, dan data *beacon* untuk setiap titik. Data penempatan *beacon* terpisah berdasarkan *sites*, *floors*, dan *zones*.

Pada gambar berikut ini telah dibuat sebuah konfigurasi dari tiga *beacon* yang akan digunakan dalam implementasi. *Beacon* ini tersebar pada satu *site*, yang berisi satu *floor*, dan tiga *zones*.



Gambar 3 Back-end Application dan service nya, terdiri dari Node.js dan IBM Cloudant NoSQL database. Tampilan ini ada pada konfigurasi IBM Bluemix via web browser. ©Dokumentasi Penulis

Tabel berikut ini akan menjelaskan tentang konfigurasi *beacon* yang digunakan pada PI.

Tabel VIII – Sites dan Zones yang telah dikonfigurasi pada PI

Sites (code)	Floor (code)	Zones (code)	Beacon (code)
ITB Depan (j51s0wdc)	Ground Floor (uh1m05g0)	Gerbang Depan (uq1y08sd) Radius 10 m	Blue Beacon (lt1n0597) Threshold 5 m
		Aula Bara (vf1m076u) Radius 10 m	Red Beacon (hq1j0gj4) Threshold 5 m
		Aula Timur (vz1r018b) Radius 10 m	Green Beacon (ks1v08ia) Threshold 5 m

Konfigurasi ini adalah contoh yang digunakan pada saat implementasi. Konfigurasi tidak hanya terbatas pada tiga *beacon* ini. Sebagai contoh, pengembangan yang lebih luas pasti membutuhkan lebih banyak *sites* dengan setiap *sites* memuat beberapa *floor*, setiap *floor* terdapat beberapa *zones*, dan setiap *zones* memuat beberapa *beacon*.

Pada saat data *beacon* masuk, PI secara otomatis akan menghasilkan sebuah paket data JSON lain yang disebut sebagai *event*. JSON ini berisi data lengkap tentang keberadaan perangkat terhadap *beacon*. Paket data ini berisi *sites*, *floors*, dan *zones* tempat perangkat mendeteksi *beacon*. Selain itu, aplikasi Android dirancang mengirimkan data tentang *beacon* yang terdeteksi setiap 5 menit sehingga PI dapat mengetahui durasi perangkat tersebut berada di sekitar *beacon*. Berikut ini adalah contoh *event* yang dihasilkan oleh PI.

```
{
  "device": {
    // The following is for a registered device.
    // An unregistered device would only contain the following: "descriptor"
    and "registered"
    "data": {}, // Object
    "unencryptedData": {}, // Object
    "name": "VisitorsPhone", // String
    "descriptor": "425f34421e7000400f7f93937a694200f2d715d4", // String
    "registered": true, // Boolean
    "blacklist": false // Boolean
  },
  "tenant_code": "71f01v6", // String
  "tenantDoc": {
    "name": "71f01v6", // String
    "dwellTime": 20000, // Number
    "deviceTimeout": 10000 // Number
  },
  "org": {
    "name": "Org", // String
    "registrationTypes": [
      "Internal",
      "External"
    ], // Array of unique Strings
    "description": null // String or null
  },
  "site": {
    "name": "Site", // String
    "tags": [], // Array of unique Strings, may be empty
    "address": {
      "country": "",
      "state": "",
      "city": "",
      "street": "",
      "zip": ""
    }, // Object
    "description": null, // String or null
    "timeZone": "America/New_York" // String
  },
  "floor": { // Object, floor follows GeoJson format
    "type": "Feature", // String

```

```

    "geometry": {
      "type": "Point",
      "coordinates": [
        0,
        0
      ]
    }, // Object
    "properties": {
      "name": "Lobby",
      "z": 1
    }, // Object
    "name": "Lobby" // String
  },
  "zone": {}, // Object, zone follows GeoJson format , may be empty
  "zone_code": "p136w01mv", // String
  "zoneTags": [], // Array of unique Strings, may be empty
  "site_code": "b43bm0bmd", // String
  "siteTags": [], // Array of unique Strings, may be empty
  "org_code": "wi2qo0119", // String
  "floor_code": "qc35g01ch", // String
  "detected_timestamp": 1441717101111, // Number
  "device_descriptor": "425f34421e7000400f7f93937a694200f2d715d4", // String
  "device_created": 1445903471963, // Number
  "registered": true, // Boolean
  "registrationType": "anonymous", // String
  "x": 91, // Number
  "y": 104, // Number
  "request_id": "proximity_71f01v6_1b6b2f575af441b9b0519f768a05e752", // String
  "sensor_type": "proximity", // String
  "state": 1, // Number
  "dwellPeriod": 0, // Number
  "dwellPeriodDelta": 0, // Number
  "dwellId": "aa3dbc20-8f01-11e5-8c99-910b7d395e83", // String
  "@docType": "RealTimeEvent", // String
  "activity": "enter", // String: "enter","exit",or "dwell"
  "subscription_code": "p5me00vm" // String
}

```

Pada JSON *event* tersebut, terdapat atribut berupa *floor_code*, *site_code*, dan *floor_code* yang menginformasikan tempat perangkat mendeteksi *beacon* tersebut. Atribut *detected_timestamp* merupakan waktu ketika perangkat mendeteksi *beacon*. Atribut *state* merupakan status yang menandakan keadaan perangkat tersebut. *State* tersebut dapat berupa *enter*, *dwell*, *exit*, atau *timeout*. Atribut *dwellPeriod* merupakan durasi waktu perangkat tersebut berada di sekitar *beacon*.

Tabel berikut ini akan menjelaskan secara detail tentang atribut *state*.

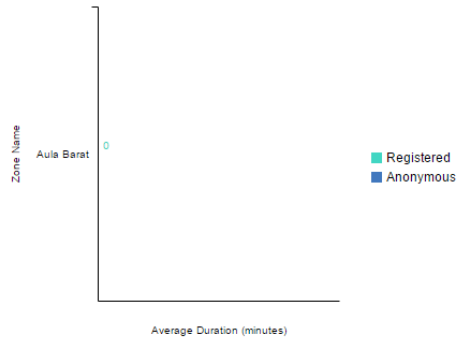
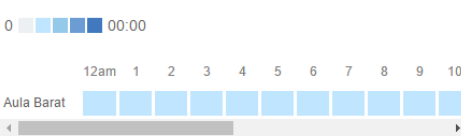

Tabel IX – State dan penjelasannya dari detail event yang dikirimkan PI

State	Number	Description
<i>Enter</i>	1	<i>State</i> akan bernilai 1 ketika perangkat berada di sekitar <i>beacon</i> untuk pertama kalinya
<i>Dwell</i>	0	<i>State</i> akan bernilai 0 ketika perangkat sudah berada di sekitar <i>beacon</i> selama 5 menit berikutnya
<i>Exit</i>	-1	<i>State</i> akan bernilai -1 ketika perangkat berada di sekitar yang <i>beacon</i> lain sehingga perangkat dianggap sudah tidak berada di sekitar <i>beacon</i> terakhir
<i>Timeout</i>	-2	<i>State</i> akan bernilai -2 ketika perangkat tidak lagi berada di sekitar <i>beacon</i> terakhir dan juga tidak berada di sekitar <i>beacon</i> lain.

Pada PI, telah disediakan sebuah antarmuka untuk melihat semua data beserta analisisnya tentang perangkat pengguna yang mendeteksi *beacon*. Data-data ini dapat digunakan untuk kebutuhan analisis kepadatan pengunjung pada titik-titik tertentu di kampus. Berikut ini adalah data yang disajikan pada *dashboard* PI.

Tabel X – Data dan analitik yang disediakan PI

Analytic	Details	Dashboard
Visits	<p>Site Visitors</p> <hr/> <p>Total Visits</p> <hr/> <p>Visit by Zone</p>	<p>Total Visits</p> <p>Site: ITB Depan Date Range: 04/04/2016 — 04/04/2016</p> <p>Site Visitors</p>  <p>Visits by Zone</p> 
	<p>Visit Density Across Time</p> <hr/> <p>Visit Density on Floor Layout</p>	<p>Visit Density Across Time</p> <p>Site: ITB Depan Floor: All Floors Date Range: 04/04/2016 — 04/04/2016</p>  <p>Visit Density on Floor Layout</p> <p>Site: ITB Depan Floor: ITB Depan Date Range: 04/04/2016 — 04/04/2016</p> 

Average Visit Duration	<div> <div>Average Visit Duration</div> <div> <div>Site</div> <div>ITB</div> <div>Depan</div> </div> <div> <div>Date Range</div> <div>04/04/2016 —</div> <div>04/04/2016</div> </div> </div> 
Duration Density Across Time	<div> <div>Duration Density Across Time</div> <div> <div>Site</div> <div>ITB</div> <div>Depan</div> </div> <div> <div>Floor</div> <div>All</div> <div>Floors</div> </div> <div> <div>Date Range</div> <div>Day</div> <div>04/04/2016 —</div> <div>04/04/2016</div> </div> </div> <div> <div>0 00:00</div> <div>12am 1 2 3 4 5 6 7 8 9 10</div> <div>Aula Barat</div> </div> 
Duration Density on Floor Layout	<div> <div>Duration Density on Floor Layout</div> <div> <div>Site</div> <div>ITB</div> <div>Depan</div> </div> <div> <div>Floor</div> <div>ITB</div> <div>Depan</div> </div> <div> <div>Date Range</div> <div>04/04/2016 —</div> <div>04/04/2016</div> </div> </div> <div> <div>+</div> <div>-</div> <div>Average Duration</div> <div>00m:00s Average Duration</div>  </div>

Sistem Notifikasi

Pengiriman notifikasi oleh administrator dilakukan melalui *front-end*. Pada *front-end* akan disediakan *form* detail notifikasi yang akan dikirimkan ke pengguna aplikasi Android. Notifikasi akan dikirimkan dari *front-end* ke *back-end* dengan format JSON melalui HTTP POST. Dari *back-end application*, notifikasi diteruskan ke *smartphone* pengguna melalui MQTT.

Notifikasi yang dikirimkan oleh *administrator* akan dibedakan menjadi tiga tipe. Tiga tipe tersebut adalah notifikasi berdasarkan jurusan/program studi, notifikasi berdasarkan unit mahasiswa, dan notifikasi untuk semua pengguna. Setiap notifikasi memiliki *route* HTTP POST dan *topic* sendiri agar notifikasi dapat sampai ke

pengguna spesifik. Meskipun notifikasi dibedakan menjadi tiga tipe, prosedur pengirimannya dari *front-end* ke *back-end* tetap sama.

Tabel XI – Penggunaan format JSON untuk pengiriman notifikasi

Notification	JSON Format
Major member	<pre>{ "major": "132", "batch": "12" }</pre>
Unit member	<pre>{ "unit": "UBG" }</pre>
Notification core body	<pre>{ "image_url": "Test", "title": "Test", "notifications": "Test", "time": "test", "location": "Test", "contact": "Test" }</pre>

Berikut ini adalah *source code* pengiriman data dari *front-end* ke *back-end* yang berupa Javascript berikut dengan *form* notifikasinya dalam bentuk HTML.

```
<html>
<head>
<script type="text/javascript">
  function submitForm() {
    var title = document.advertise.title.value;
    var notifications = document.advertise.notifications.value;
    var time = document.advertise.time.value;
    var location = document.advertise.location.value;
    var contact = document.advertise.contact.value;
    var json =
    '{"title":"' + title + '\", \"notifications\": \"' + notifications + '\", \"time\": \"' + time +
    '\", \"location\": \"' + location + '\", \"contact\": \"' + contact + '\"}';
    postData(json);
  }
  function postData(input) {
    var xhr = new XMLHttpRequest();
    var url = "/post/advertise";
    xhr.open("POST", url, true);
    xhr.setRequestHeader('Content-Type', 'application/json');
    xhr.send(input);
    alert("Advertisement has been sent");
  }
</script>
</head>
<body>
<div id="page-content-wrapper">
  <div class="container-fluid">
    <div class="row">
      <div class="col-lg-12">
        <h2>Notification Center</h2>
        <h3>Advertise</h3>
        <form name="advertise">
          Image URL:<br>
          <input type="text" name="image_url"><br>
          Title:<br>
          <input type="text" name="title"><br>
          Notifications:<br>
          <textarea name="notifications" cols="50"
rows="10"></textarea><br>
          Time:<br>
          <input type="text" name="time"><br>
          Location:<br>
          <input type="text" name="location"><br>
          Contact:<br>
          <input type="text" name="contact"><br><br>
          <input type="submit" value="Send"
onclick="javascript:submitForm()">
        </form>
      </div>
    </div>
  </div>
</div>
```

```
</div>
</div>
</div>
</body>
</html>
```

Fungsi *submitForm* merupakan pengambilan data dari form HTML yang diisi. Fungsi *postData* merupakan pengiriman data ke *back-end*.

```
app.post('/post/advertise', function (req, res){
  var notificationData = req.body;
  var doc = JSON.stringify(notificationData);
  client.publish("ubeacon/user/notification/advertise", doc, function(){
    console.log("JSON sent (advertise)");
  });
});
```

Source code di atas merupakan penerusan notifikasi yang diterima dari *front-end* ke pengguna aplikasi melalui MQTT. Notifikasi dari *front-end* dikirimkan melalui HTTP POSTS, pada *back-end* diterima melalui masing-masing *route*-nya dan dikirimkan ke masing-masing *topic* agar sampai ke pengguna spesifik.

2.1.3 Environment

Dalam implementasi sub-sistem *back-end application* digunakan beberapa *software* pendukung sebagai berikut.

1. Eclipse IDE (version Mars 4.5.0)

Eclipse Integrated Development Environment (IDE) digunakan sebagai *project manager* serta alat untuk menulis *source code* dalam Node.js. Dengan menggunakan *plug-in* IBM Bluemix, Eclipse juga digunakan untuk *deploying* Node.js yang telah ditulis.

2. IBM Bluemix Node.js Runtime

IBM Bluemix diakses menggunakan *web browser* kemudian dikonfigurasi secara langsung pada halamannya. Dengan menggunakan IBM Bluemix Node.js Runtime, *back-end application* yang telah ditulis dapat dijalankan pada *server*.

3. IBM Presence Insights

IBM Presence Insights diakses menggunakan *web browser*. Pada lamannya, terdapat *dashboard* yang menampilkan data-data dari perangkat yang mendeteksi *beacon* serta sajian analitik dari data-data tersebut.

4. IBM Cloudant NoSQL Database

IBM Cloudant NoSQL Database digunakan sebagai *database* untuk menyimpan data registrasi pengguna serta informasi *beacon* per UUID-major-minor. Semua data disimpan dalam bentuk JSON yang terpisah pada setiap dokumen.

5. Mosquitto MQTT Broker

Sebuah *server* MQTT *open source* yang digunakan sebagai MQTT Broker selama masa implementasi. Digunakan sebagai jembatan penghubung antara aplikasi Android dengan *back-end application*.

6. HiveMQ Websocket Client

Simulator MQTT *publish/subscribe* yang digunakan selama masa implementasi untuk membantu memudahkan pembuatan *back-end application* sebagai *requester* dan penerima *response*.

2.1.4 Prosedur Implementasi

Dalam melakukan implementasi sub-sistem *back-end application* dilakukan beberapa tahapan prosedur sebagai berikut.

Back-end Application

1. Melakukan pemilihan *server* tempat *back-end application* akan dijalankan. Pada implementasi ini dipilih IBM Bluemix sebagai *server*.
2. Melakukan pemilihan *runtime*. Pada implementasi ini dipilih Node.js Runtime.
3. Melakukan penulisan program dan *deploying* ke *server* menggunakan Eclipse IDE.
4. Melakukan pengetesan *back-end application* dan Mosquitto MQTT Broker dengan mengirimkan data menggunakan HiveMQ Websocket Client.
5. Melakukan pengetesan pengiriman data *sign-up*, *sign-in*, dan data *beacon*. Melihat balasan dari *back-end application* setelah mendapat kiriman data tersebut.
6. Melihat status balasan *back-end application* dengan data *sign-up* dan *sign-in* yang telah disimpan di IBM Cloudant NoSQL Database.
7. Melihat adanya balasan *back-end application* dengan informasi *beacon* yang telah disimpan di IBM Cloudant NoSQL Database.
8. Melakukan pengiriman data ke *back-end application* untuk semua fungsi dari aplikasi Android.
9. Melihat balasan *back-end application* pada aplikasi Android.

IBM Presence Insights

10. Menyiapkan peta tempat yang akan dipasang *beacon* dan penentuan tempat *beacon* pada peta.
11. Penambahan IBM Presence Insights *services* dan *binding* dengan *back-end application* yang sudah ada.
12. Pembuatan *organization* dan *tenant* baru untuk mendapatkan *credentials* agar dapat mengirim data ke dalam PI, serta membuat *public key* untuk enkripsi data pengguna yang masuk ke PI.
13. Melakukan konfigurasi pada IBM Presence Insights. Konfigurasi meliputi pembuatan *sites*, *floor*, dan *zones* pada peta yang telah diunggah, penentuan radius *zones*, penentuan *threshold* jarak *beacon*, dan mengisi sebuah *webhook* untuk menerima respons dari PI.
14. Mengirimkan data *beacon* dalam format JSON ke *back-end application* yang kemudian akan diteruskan ke PI
15. Melihat data dan analisis yang masuk pada *dashboard* PI.

Sistem Notifikasi

16. Menyiapkan dua atau lebih *smartphone* dengan aplikasi untuk menerima notifikasi

17. Mengisi *form* notifikasi pada *front-end* administrator
18. Mengirimkan notifikasi
19. Mencocokkan notifikasi yang sampai pada kedua *smartphone* dengan yang dikirimkan dari *front-end*

Sistem Absensi

20. Menyiapkan dua atau lebih *smartphone* untuk melakukan sistem absensi
21. Menyiapkan *beacon* yang sudah didaftarkan sebagai *beacon* kelas pada *database*
22. Menyalakan mode kelas pada *smartphone* untuk memulai sistem absensi
23. Melihat nama pengguna *smartphone* pada daftar hadir yang ditampilkan pada *front-end*.

2.1.5 Progres Implementasi

Sebelum *deploying* ke *server*, diperlukan beberapa *setting* manual pada IBM Bluemix melalui halaman *web* di <http://console.au-syd.bluemix.net/>. Sebuah aplikasi baru perlu dibuat dengan *runtime* Node.js dilengkapi dengan IBM Cloudant NoSQL Database *service*.

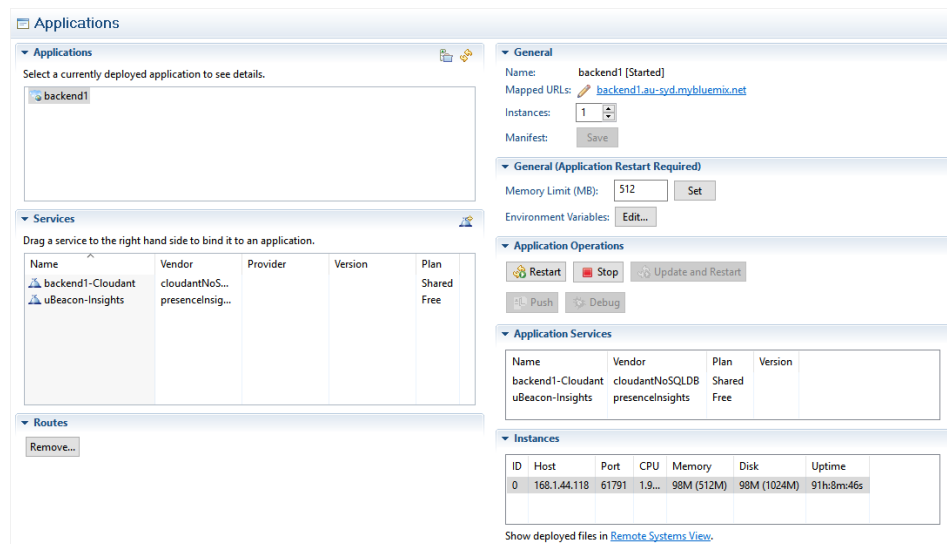
Untuk menggunakan modul-modul Node.js pada *server* IBM Bluemix, diperlukan deklarasi *dependencies* modul di file *package.json* pada *back-end application project files*. Hal ini perlu dilakukan sebelum *deploying* ke *server*. Deklarasi ini dibutuhkan agar *server* mengetahui modul apa saja yang dibutuhkan.

Deploying ke *server* membutuhkan spesifikasi kuota *disk*, *memory*, *domain*, *hostname*, dan *instance*. Spesifikasi kebutuhan alokasi *server* ini dideklarasikan pada file *manifest.yml*

Setelah *source code* ditulis dan disimpan sebagai file bagian dari *back-end application project* pada Eclipse IDE, tahap selanjutnya adalah *deploying* ke *server* menggunakan IBM Bluemix *plug-in* Eclipse IDE yang dapat diunduh dari Eclipse Marketplace.

The screenshot shows the 'Overview' tab of the IBM Bluemix console. It contains several sections for configuring the environment:

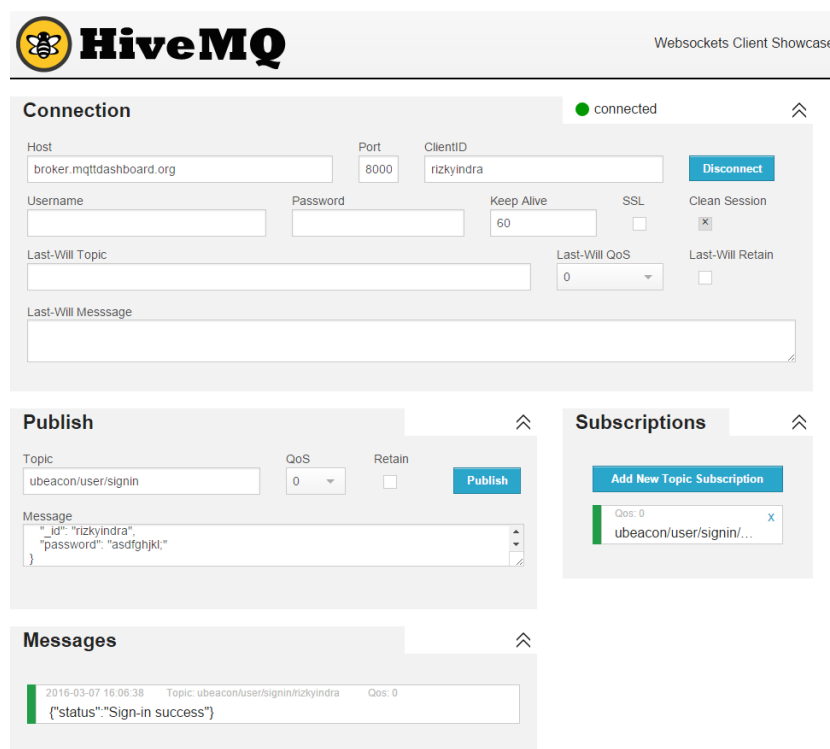
- General Information:** Fields for 'Server name' (IBM Bluemix), 'Host name' (Cloud), and 'Runtime Environment' (IBM Bluemix Runtime).
- Account Information:** Fields for 'Email' (rizkyindra@students.itb.ac.id), 'Password' (masked), 'URL' (https://api.au-syd.bluemix.net), 'Organization' (rizkyindra@students.itb.ac.id), and 'Space' (Rizky Indra Syafrian). Buttons for 'Clone Server...', 'Update Password...', and 'Validate Account' are present.
- Publishing:** Options for 'Never publish automatically' (selected), 'Automatically publish when resources change', and 'Automatically publish after a build event'. A 'Publishing interval (in seconds)' of 15 is set.
- Timeouts:** Fields for 'Start (in seconds)' (600) and 'Stop (in seconds)' (60).
- Server Status:** Shows 'IBM Bluemix: Connected' with 'Connect' and 'Disconnect' buttons.



Gambar 4 Eclipse IDE dengan *plug-in* IBM Bluemix. Konfigurasi *server* dapat dilakukan melalui IDE ini atau secara langsung via *web browser*. ©Dokumentasi Penulis

Setelah berhasil *deploying* ke *server*, tahap selanjutnya mengecek *back-end application* telah berjalan pada *server*. Untuk melihat kerja *back-end application* perlu dilakukan pengiriman pesan ke *server* via internet dengan protokol MQTT. Untuk dapat mengirimkan pesan dengan protokol MQTT, digunakan sebuah *online MQTT client* HiveMQ.

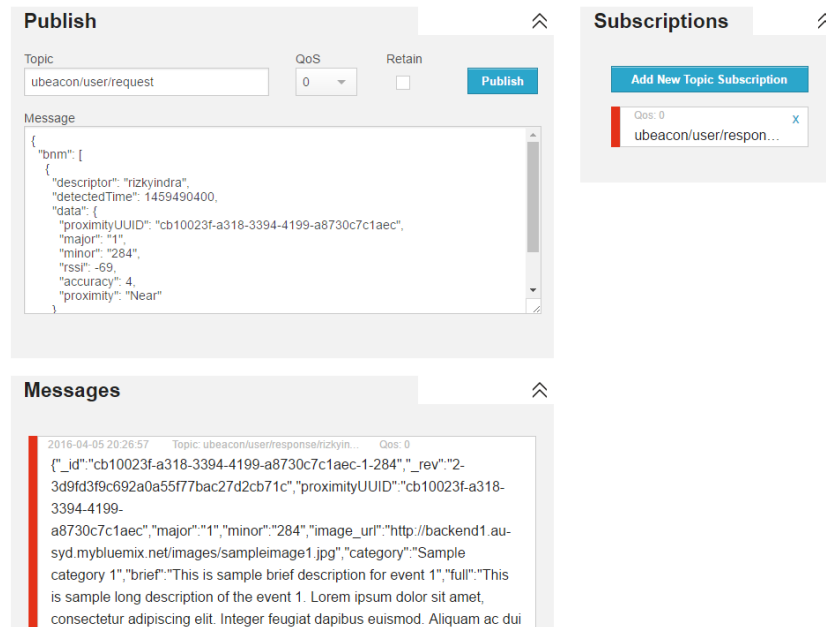
Di bawah ini merupakan tampilan HiveMQ. *Host* dan *port* diisikan dengan alamat *broker* MQTT yang digunakan. *Publish* dan *subscribe* diisikan dengan alamat yang sudah ditentukan pada *back-end application*. Gambar berikut ini adalah percobaan pengiriman data *sign-in* ke *back-end application* beserta balasannya. Pada bagian *publish* diisikan *topic* dan pesan berupa data *sign-in*. Pada bagian *subscribe*, *topic* diatur agar mendengarkan pesan pada *topic* tempat balasan akan dikirim.



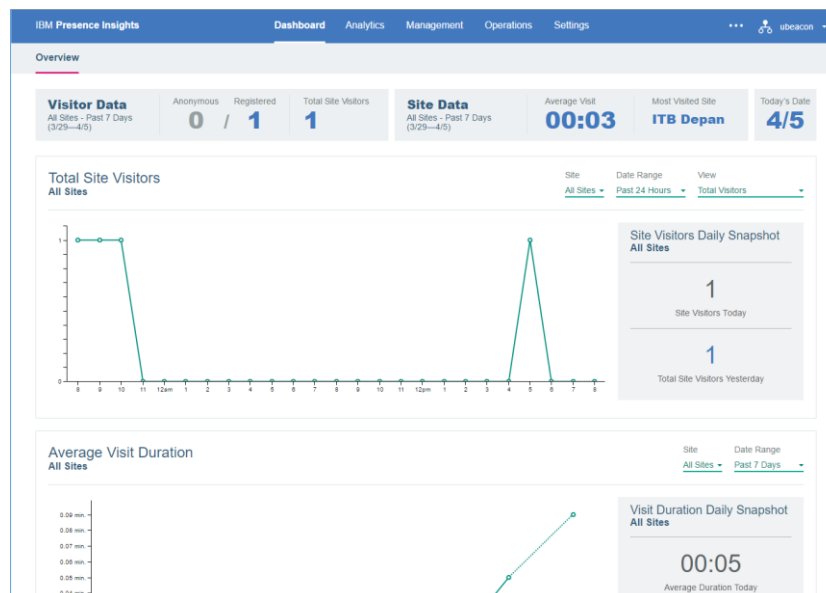
Gambar 5 Tampilan HiveMQ MQTT Client yang dibuka pada *web browser* untuk simulasi koneksi MQTT. Pada gambar terlihat HiveMQ menerima pesan *sign-in success*. ©Dokumentasi Penulis

Pada gambar tersebut terlihat bahwa *back-end application* telah berfungsi dengan baik. *Back-end application* juga dapat menangani apabila pengguna memasukkan *username* yang tidak valid atau *password* yang salah.

Untuk PI, implementasi dapat dilakukan dengan cara mengirimkan data *beacon* ke *back-end application*. *Back-end application* kemudian akan mengirimkan balasan berupa informasi *beacon* sekaligus meneruskan data *beacon* ke PI. Data *beacon* yang diteruskan ke PI ditampilkan pada *dashboard* lengkap dengan data dan analisisnya.



Gambar 6 Pada gambar terlihat HiveMQ mengirimkan pesan berupa data *beacon* serta menerima pesan balasan berupa informasi *beacon*. ©Dokumentasi Penulis

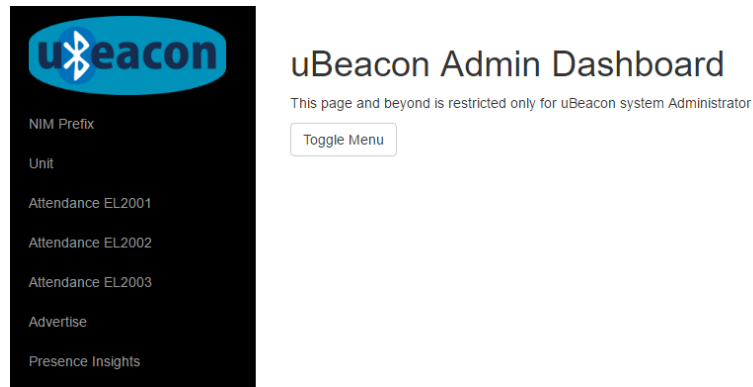


Gambar 7 Gambar *overview dashboard* pada IBM Presence Insight menunjukkan aktivitas perangkat yang mendeteksi *beacon*. ©Dokumentasi Penulis

Dapat terlihat pada gambar di atas, *overview dashboard* PI menampilkan jumlah perangkat yang mengunjungi *sites* ITB Depan yang telah dibuat serta rata-rata durasi kunjungan. Di bawahnya ditampilkan sebuah grafik waktu kunjungan pada *sites* ITB Depan. Di bawahnya lagi ditampilkan sebuah grafik rata-rata durasi kunjungan. Data yang ditampilkan juga mengandung data-data percobaan implementasi lainnya.

Selain itu, *back-end application* dapat menerima pesan dan membalas pesan sesuai dengan fungsinya sesuai dengan tabel format JSON yang telah dijelaskan pada bagian struktur implementasi

Front-end yang dibangun dapat digunakan sebagai pengirim notifikasi dan sebagai pemantau daftar hadir kelas kuliah. Berikut ini adalah tampilan *front-end* yang akan digunakan oleh *administrator*.



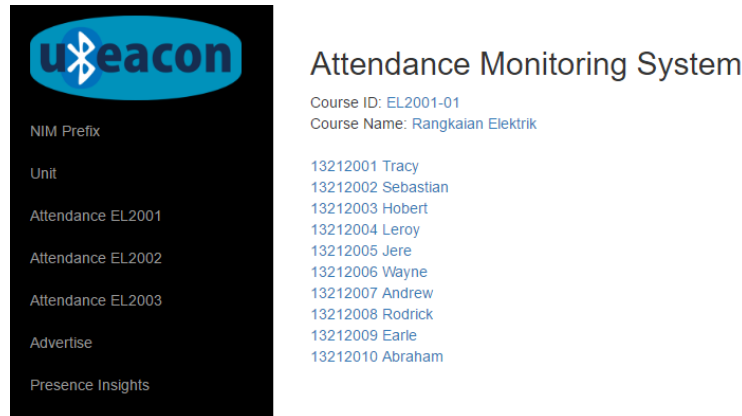
Gambar 8 Tampilan uBeacon Admin Dashboard yang merupakan *front-end* untuk *administrator*. ©Dokumentasi Penulis

Pada *administrator dashboard*, terdapat beberapa menu dengan fungsi yang berbeda-beda. NIM Prefix merupakan menu yang digunakan untuk mengirimkan notifikasi ke mahasiswa jurusan/program studi dan angkatan tertentu. Unit juga menu untuk mengirimkan notifikasi ke anggota unit mahasiswa. Sedangkan Advertise merupakan menu untuk mengirimkan notifikasi ke semua pengguna aplikasi.

Form yang digunakan *administrator* untuk mengirimkan notifikasi tidak berbeda jauh untuk setiap tipe notifikasi. *Form* mengikuti format JSON yang digunakan untuk mengirimkan notifikasi. Berikut ini adalah gambar *administrator dashboard* bagian notifikasi Advertise.

Gambar 9 Tampilan uBeacon Admin Dashboard bagian notifikasi Advertise. ©Dokumentasi Penulis

Pada menu Attendance, kode kelas dan nama kuliah ditampilkan. Kemudian apabila ada peserta kelas yang hadir dan mengaktifkan mode kelas, maka namanya akan muncul pada daftar hadir. Berikut ini adalah gambar *administrator dashboard* bagian Attendance dengan beberapa nama peserta kelas yang hadir.



Gambar 10 Tampilan uBeacon Admin Dashboard bagian notifikasi Advertise. ©Dokumentasi Penulis

2.1.6 Permasalahan dan Solusi

Dalam implementasi sub-sistem *back-end application* ini ditemukan beberapa masalah sebagai berikut.

1. Ketika mengirimkan pesan berupa data *sign-in* ataupun *sign-up*, apabila pengguna mencoba menggunakan *username* yang sudah terpakai maka pesan akan terkirimkan melalui *topic* dengan nama *username* tersebut. Artinya pesan tersebut akan diterima oleh pengguna dengan *username* aktif tersebut. Aplikasi Android dengan *username* aktif tersebut harus dapat mengabaikan pesan ini.
2. Tidak ada enkripsi yang digunakan ketika pengiriman pesan yang berisikan data pengguna. Semua data pengguna yang lewat melalui internet dapat ditangkap oleh *man in the middle*. Harus ada mekanisme enkripsi sebelum pesan dikirim untuk agar data pengguna aman.

2.2 Implementasi Sub-sistem *Mobile Application*

2.2.1 Pendahuluan

Sub-sistem *Mobile Application* memiliki peran untuk melakukan proses yang menghubungkan *Beacon*, *Backend Application*, dan pengguna dalam sebuah sistem. Platform *mobile application* yang digunakan adalah Android dengan versi minimum 4.2.2 yang sudah mengandung fitur *Bluetooth Low Energy*. *Mobile Application* akan menampilkan hasil pemrosesan berupa informasi yang dibutuhkan oleh pengguna aplikasi tersebut. Adapun, dalam implementasinya *mobile application* akan terbagi dalam *foreground process* dan *background process*.

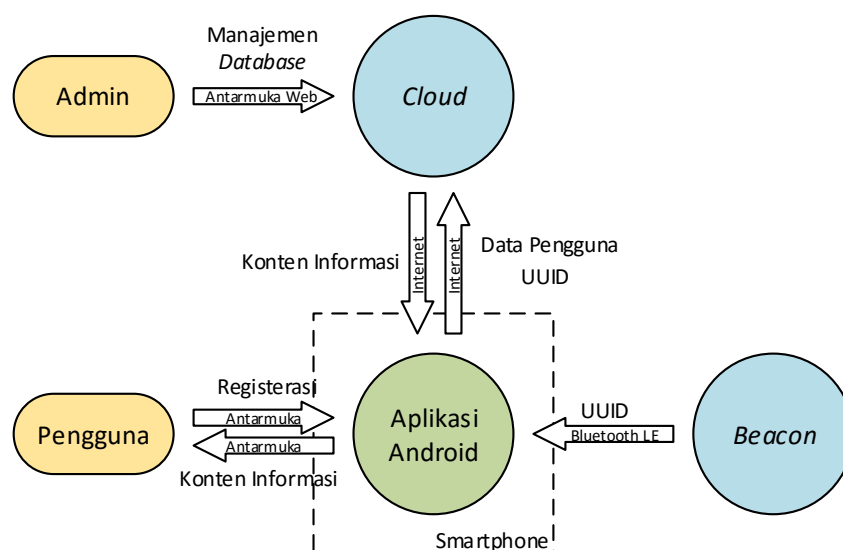
Mobile Application bertugas untuk menerima sinyal *bluetooth* yang mengandung informasi *identifier* yang dipancarkan oleh perangkat *Beacon*. *Mobile Application* akan memproses dan mengirimkan informasi yang di bawa oleh *Beacon* kepada *Backend Application* via MQTT. Selanjutnya, *Mobile Application* akan menerima kembali informasi yang diberikan oleh *Backend Application* via MQTT sehingga *Mobile Application* dapat mengakses gambar dan teks yang sesuai dengan *Beacon* yang terdeteksi.

Identifier yang akan dipancarkan oleh perangkat Beacon antara lain adalah Universally Unique Identifier(UUID), Major, Minor, Proximity, dan RSSI. Komponen *identifier* yang akan membedakan antar perangkat beacon yang terdeteksi adalah pada komponen *major*-nya, sehingga *Mobile Application* akan memproses setiap komponen *major* yang terdeteksi.

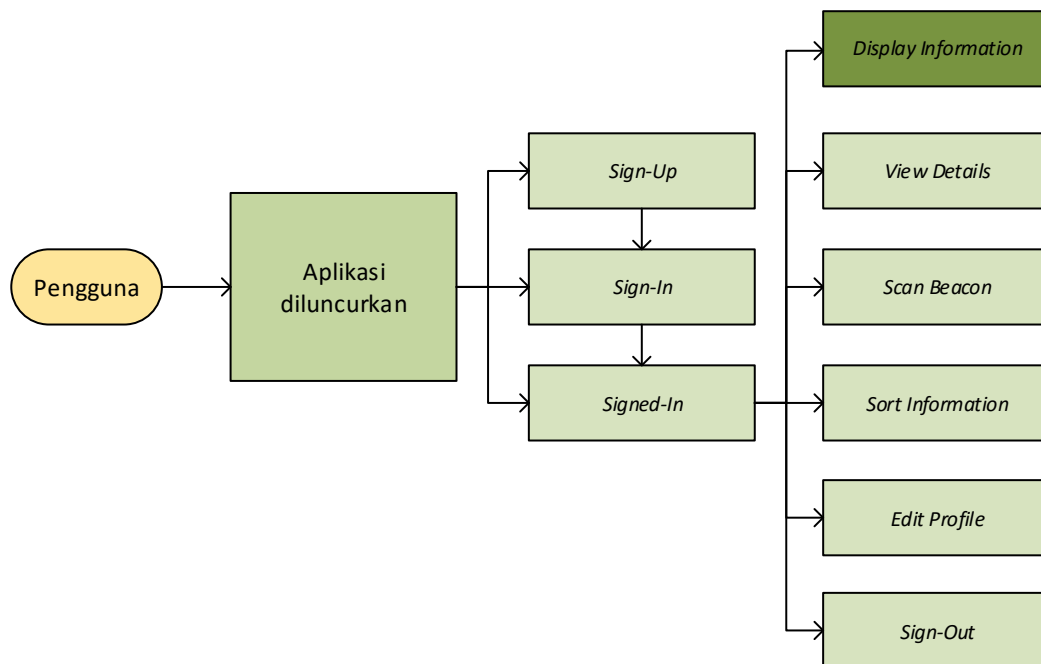
Setelah mendapatkan informasi yang lengkap dari *Backend Application*, maka *mobile application* akan menampilkan hasil informasi yang sesuai dengan cara mengakses file gambar dan teks yang telah diarahkan oleh *Backend Application*, sehingga pengguna dapat mengakses informasi tentang acara yang sedang berlangsung di suatu gedung yang dipasangkan perangkat beacon dengan *identifier* tertentu.

2.2.2 Struktur Implementasi

Secara garis besar, alur interaksi yang terjadi antara pengguna dan *mobile application* terjadi seperti bagan di bawah ini.



Gambar 11 Skema hubungan antar sistem ©Dokumentasi Penulis



Gambar 12 Diagram blok antarmuka pengguna aplikasi Android. ©Dokumentasi Penulis

Implementasi mobile application dibagi ke dalam dua proses yaitu foreground process dan background process. Foreground process terdiri dari tampilan GUI yang bertujuan untuk memudahkan interaksi pengguna dalam memberikan perintah kepada sistem. Proses ini terdiri dari sekumpulan proses yang dilihat oleh pengguna pada waktu tertentu meliputi tampilan aplikasi yang meminta data pengguna untuk keperluan Sign In, Sign Up maupun Edit Profile berupa nama, username, password, e-mail, jenis kelamin, NIM, unit, interest, dan tanggal dan tahun lahir pengguna. Sementara background process merupakan proses yang tidak dilihat oleh pengguna pada waktu tertentu. Proses ini meliputi peregistrasi aplikasi sebagai subscriber ke server untuk menerima push notification, scanning Beacon dan proses komunikasi data menggunakan MQTT.

Berikut adalah gambaran umum dari sub modul antarmuka dan koordinasi yang disajikan dalam tabel.

Tabel XII – Gambaran umum sub-sistem *mobile application*

Blok	Fungsi
Fungsi	<ul style="list-style-type: none"> • Mengkoordinasi sub sistem perangkat beacon dengan sub sistem backend apps.
Input	<ul style="list-style-type: none"> • <i>Username</i>, sebagai public profile berupa variasi dari nama pengguna dan digunakan untuk pengaturan informasi secara personal. • <i>Password</i>, sebagai kumpulan karakter personal yang digunakan sebagai akses kedalam Aplikasi Antarmuka. • <i>Name</i>, sebagai data pengguna. • <i>E-mail</i>, sebagai data pengguna. • <i>NIM</i>, sebagai data pengguna untuk personalisasi informasi. • <i>Date of Birth</i>, sebagai data pengguna. • <i>Unit</i>, sebagai data pengguna untuk personalisasi informasi. • <i>Interest</i>, sebagai data pengguna untuk personalisasi informasi. • <i>Occupation</i>, sebagai data pengguna untuk personalisasi informasi.
Output	<ul style="list-style-type: none"> • Data pengguna masuk kedalam <i>database</i> • Informasi yang berada pada server terunduh dan ditampilkan pada layar

<i>smartphone</i> pengguna secara lengkap dan personal.	
Kebutuhan Kuantitatif	<ul style="list-style-type: none"> • Dapat melakukan cek state terhadap status sign-in pengguna saat aplikasi diluncurkan (bila sign-in telah dilakukan sebelumnya maka bila aplikasi diluncurkan pengguna akan langsung masuk ke state utama tanpa perlu melakukan proses sign-in kembali). • Dapat mengubah masukan pengguna berupa Username, Password, Name, E-mail, NIM, Unit, Interest, Occupation dan Date of Birth sebagai JSON file. • Dapat mengirim JSON file ke server melalui MQTT. • Dapat menerima balasan dari server setelah Sign-In, Sign-Up atau Edit Profile dilakukan oleh user. • Dapat menyimpan balasan dari server ke dalam Shared Preference untuk login state dan menyimpan data profile pengguna dari server dalam memory local <i>smartphone</i>.
Deskripsi Kebutuhan Performansi	<ul style="list-style-type: none"> • Dapat menampilkan main activity atau state utama setelah proses sign-in berhasil dilakukan. • Dapat menampilkan Sign-In state setelah pengguna berhasil melakukan registrasi akun melalui Sign-Up. • Dapat menerima push notification dari server dan backend kapanpun selama aplikasi dijalankan dan tidak di-destroy. • Dapat mengirimkan payload beacon ke backend untuk fitur Attend Class yang bisa setiap 5 menit sekali dikirim secara otomatis. • Dapat menerima informasi berupa JSON file dari server. • Dapat mengubah informasi berupa JSON file yang diterima dari server ke dalam layout pada aplikasi (parsing JSON). • Dapat menampilkan konten informasi yang diterima dari server ke layar <i>smartphone</i> pengguna secara lengkap dan personal. • Dapat melakukan proses log out.

2.2.3 *Environment*

Dalam mengimplementasikan sub-sistem *mobile application* digunakan *software* dan *hardware* sebagai berikut.

1. Android Studio

Software Android Studio digunakan untuk membangun program berbasis Android yang akan digunakan untuk scan Bluetooth, pemrosesan dan pengiriman data, serta sebagai *Graphic User Interface(GUI)* bagi pengguna nya. Dalam pembuatan aplikasi nya, Android Studio memiliki berbagai tools serta fitur *drag and drop* maupun manual dalam pembuatan *user interface* nya.

2. Micro-USB Cable Data/ Kabel Galaxy Note 10.1

Kabel data Micro-USB digunakan untuk melakukan download aplikasi ke dalam perangkat Android. Ketika perangkat android telah tersambung dengan computer/laptop dan tombol *Run* pada Android Studio di klik, maka *mobile application* akan otomatis ter-install pada perangkat android yang dituju.

3. Samsung Galaxy Note 10.1

Perangkat Samsung Galaxy Note 10.1 digunakan karena devais ini sudah memenuhi syarat minimum suatu devais yang mendukung fitur Bluetooth Low

Energy yang dimiliki oleh Android versi 4.2.2 ke atas. Perangkat ini juga digunakan sesuai dengan keberadaan sumber daya yang telah ada.

2.2.4 Prosedur Implementasi

Dalam melaksanakan implementasi dari sub-sistem *mobile application* ini dilakukan beberapa langkah prosedur sebagai berikut.

1. Melakukan pemilihan perangkat android yang kompatibel dengan spesifikasi yang dibutuhkan.
2. Membangun *mobile application* berbasis android menggunakan Android Studio.
3. Menambahkan fitur *Bluetooth Low Energy*(BLE) untuk pemindaian(*scan*) beacon.
4. Membangun interaksi *mobile application* dengan server IBM Bluemix melalui MQTT.
5. Membangun fitur penyimpanan dan pengelolaan data pada *local memory* smartphone.
6. Membangun *Graphic User Interface*(GUI) sebagai interaksi antarmuka dengan pengguna.
7. Membangun fitur personalisasi penggunaan aplikasi dalam bentuk akun pengguna (*user account*).
8. *Push-Notification* sebagai fitur personalisasi pengguna.
9. Personalisasi dengan menambahkan fitur presensi kehadiran mahasiswa dalam kelas perkuliahan.

2.2.5 Progres Implementasi

2.2.5.1 Pemilihan perangkat android yang kompatibel dengan spesifikasi yang dibutuhkan.

Pada implementasi yang dilakukan, digunakan perangkat android yang kompatibel yang memiliki fitur Bluetooth Low Energy(BLE), yaitu perangkat android yang memiliki versi 4.1.2 ke atas. Dalam hal ini, pengguna menggunakan device android OPPO Find 7 untuk implementasi pengembangan *mobile application* yang di bangun.

2.2.5.2 Membangun *mobile application* berbasis android menggunakan Android Studio.

Pembuatan project android application dilakukan dengan menggunakan software Android Studio. Project terdiri dari beberapa kelas (Java Class) yang saling berkoordinasi untuk membangun sistem yang tepat sesuai kegunaannya. Kelas Navigation.java merupakan kelas utama yang menampung seluruh informasi tampilan yang ditempatkan pada bagian *fragment container* untuk menampilkan tampilan pada pengguna aplikasi.

2.2.5.3 Fitur *Bluetooth Low Energy*(BLE) untuk pemindaian(*scan*) beacon.

Fungsi utama dari *mobile application* uBeacon adalah untuk dapat melakukan pemindaian dan menangkap informasi pada suatu beacon yang terdeteksi. Beacon secara kontinu akan terus memancarkan signal Bluetooth dengan teknologi *Bluetooth Low Energy* selama beacon dapat menerima daya dari baterai. Oleh karena itu, perlu untuk menambahkan fitur Bluetooth Low Energy pada *mobile application* yang dibangun.

Dalam melakukan implementasi hal ini, digunakan library yang telah tersedia oleh pengembang teknologi Beacon, yaitu Estimote Beacon Library. Untuk dapat mengimplementasikan library estimate perlu dilakukan pengaturan pada skrip build.gradle(Module:app) untuk mendapatkan Estimote SDK seperti berikut.

```
dependencies {  
    ...  
    compile 'com.estimote:sdk:0.9.4@aar'  
    ...  
}
```

Dengan menggunakan library Estimote Beacon dengan Estimote SDK, maka dapat dilakukan berbagai operasi terhadap pemindaian beacon untuk mendapatkan berbagai atribut.

Selanjutnya, untuk dapat mengetahui kehadiran dan keberadaan beacon digunakan fitur *Monitor Beacon*. Dengan menggunakan fitur *Monitor Beacon*, smartphone akan melakukan pemindaian beacon dalam periode tertentu selama modul Bluetooth pada smartphone aktif meskipun aplikasi belum dijalankan. Implementasi *Monitor Beacon* dilakukan dengan membuat kelas (class) *MyApplication.java* sebagai berikut.

```
public class MyApplication extends BeaconScannerApp implements MqttCallback  
{  
  
    private BeaconManager beaconManager1, beaconManager2, beaconManager3,  
        beaconManager4, beaconManager5, beaconManager6;  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
  
        beaconManager1 = new BeaconManager(getApplicationContext());  
        beaconManager2 = new BeaconManager(getApplicationContext());  
        beaconManager3 = new BeaconManager(getApplicationContext());  
        beaconManager4 = new BeaconManager(getApplicationContext());  
        beaconManager5 = new BeaconManager(getApplicationContext());  
        beaconManager6 = new BeaconManager(getApplicationContext());  
  
        //Major-1  
        beaconManager1.connect(new BeaconManager.ServiceReadyCallback() {  
            @Override  
            public void onServiceReady() {  
                beaconManager1.startMonitoring(new Region(  
                    "monitored region",  
                    UUID.fromString(  
                        "CB10023F-A318-3394-4199-A8730C7C1AEC"),  
                        1,  
                        284));  
            }  
        });  
  
        beaconManager1.setMonitoringListener(new  
            BeaconManager.MonitoringListener() {  
            @Override  
            public void onEnteredRegion(Region region, List<Beacon> list) {  
                Data.Major_onEntered = "1";  
                System.out.println("Variabel Major =  
"+Data.Major_onEntered);  
  
                showNotification(  
                    "Beacon-1 detected!",  
                    "Tap to see what's happening.",  
                    Navigation.class);  
                publishPayload("1");  
            }  
        });  
    }  
}
```

```

        public void onExitedRegion(Region region) {

            ...
        }
    }
}

```

Kelas (class) *MyApplication.java* merupakan sebuah kelas yang berperan sebagai *Application* pada aplikasi yang dibangun, sehingga perintah yang dijalankan pada kelas ini akan berjalan pada background activity smartphone. Hal tersebut memungkinkan aplikasi untuk berjalan tanpa dijalankan(*launch*), bahkan sejak dilakukan *bootup* pada smartphone.

Fitur Monitoring Beacon memungkinkan aplikasi untuk dapat membuat daerah(*region*) tertentu pada setiap beacon dengan atribut tertentu(ditentukan oleh *identifier* UUID, Major, dan Minor). Dalam pengujian sistem yang dibuat, digunakan 6 buah perangkat beacon untuk dilakukan pengujian. Oleh karena itu, dibentuk object *beaconManager1*, *beaconManager2*, hingga *beaconManager6* seperti yang ditunjukkan pada potongan kode di atas.

Berdasarkan rancangan sistem yang dibangun, setiap kali smartphone mendeteksi keberadaan beacon dengan Major tertentu, maka user akan diinformasikan dengan cara memberikan notifikasi pada beacon. Hal itu di atur dalam kode berikut :

```

public void showNotification(String title, String message, Class
Move) {

    Random random = new Random();
    int m = random.nextInt(9999 - 1000) + 1000;

    Intent notifyIntent = new Intent(this, Move);
    notifyIntent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
    PendingIntent pendingIntent = PendingIntent.getActivities(this,
0,
    new Intent[] { notifyIntent },
    PendingIntent.FLAG_UPDATE_CURRENT);
    Notification notification = new Notification.Builder(this)
        .setSmallIcon(R.drawable.icon)
        .setContentTitle(title)
        .setContentText(message)
        .setAutoCancel(true)
        .setContentIntent(pendingIntent)
        .build();
    notification.defaults |= Notification.DEFAULT_SOUND;
    NotificationManager notificationManager =
        (NotificationManager)
        getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(m, notification);
}

```

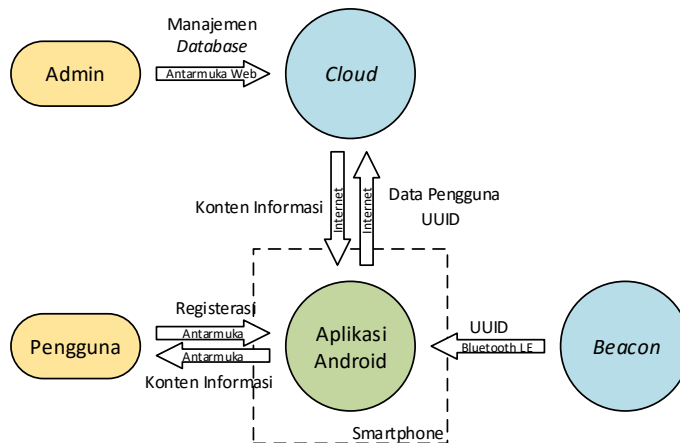
Pengguna smartphone di atur agar hanya akan menerima satu kali notifikasi pada beacon yang sama selama user berada pada daerah tertentu. Hal tersebut diatur pada method *onEnteredRegion(...)*. Jika pengguna meninggalkan lokasi dan tidak terikat dengan beacon sebelumnya, maka aplikasi akan memasuki keadaan *onExitedRegion(...)* sehingga ketika pengguna kembali ke tempat yang sama, maka akan muncul notifikasi kembali pada saat mendekati beacon sebelumnya. Hal tersebut dapat menjaga kenyamanan pengguna dalam berinteraksi dengan sistem.

Pemanggilan method *publishPayload()* pada kode merupakan proses pengiriman payload data melalui MQTT yang akan dijelaskan pada bagian selanjutnya. Secara umum, aplikasi akan menyimpan identifier beacon untuk dikirimkan melalui MQTT

kepada server untuk mendapatkan balasan informasi yang sesuai dengan beacon yang ditemukan.

2.2.5.4 Interaksi *mobile application* dengan server IBM Bluemix melalui MQTT.

Mobile Application menerima identifier beacon yang telah dilakukan pada bagian sebelumnya. Dengan mendeteksi daerah(*region*) yang telah terdefinisi pada *MyApplication.java*, maka aplikasi selanjutnya akan mengirimkan informasi tertentu untuk mendapatkan balasan dari server IBM Bluemix yang sesuai dengan beacon yang terdeteksi. Interaksi antara kedua bagian terjadi melalui *Message Queue Telemetry Transport* (MQTT). Berikut skema yang terjadi antara aplikasi android dengan server.



Gambar 13 Skema hubungan antar sistem ©Dokumentasi Penulis

Untuk dapat mengimplementasikan fitur MQTT, maka pada kelas *MyApplication.java* perlu dikonfigurasi untuk mengimplementasikan *MqttCallback* seperti berikut.

```

public class MyApplication extends BeaconScannerApp implements
MqttCallback {
    ...
    ...
    ...
}

```

Selain itu, perlu dilakukan konfigurasi penggunaan library MQTT Paho berupa file *org.eclipse.paho.client.mqttv3-1.0.2.jar* dan *wmqtt.jar*. Kedua file tersebut dilampirkan pada bagian *libs* pada project android. Selanjutnya kedua file tersebut perlu untuk disisipkan dalam skrip *build.gradle* untuk di *compile*.

Pada awalnya, untuk menggunakan fitur MQTT perlu dilakukan inisialisasi koneksi seperti sebagai berikut.

```

private MqttClient client;
...
...
try {
    String cacheDir = getCacheDir().getAbsolutePath();

    client = new MqttClient("tcp://test.mosquitto.org:1883",
        "ubeacon/user/request",
        new MqttDefaultFilePersistence(cacheDir));

    client.connect();
    client.setCallback(MyApplication.this);
    client.subscribe("ubeacon/user/response/" + nameHeader);
} catch (MqttException e) {
    throw new RuntimeException(e);
}

```

```
}  
...  
...
```

Implementasi penggunaan MQTT pada aplikasi terbagi menjadi dua bagian, yaitu publish dan subscribe. Tahap publish dilakukan dengan menerapkan method `publishPayload()` sebagai berikut.

```
private void publishPayload(String majorDetected){  
    //Paket JSON Payload pada MQTT  
    JSONObject jsonObject = new JSONObject();  
    try {  
        //paketin jadi json  
        JSONArray array1 = new JSONArray();  
        JSONObject obj = new JSONObject();  
        JSONObject obj11 = new JSONObject();  
        obj.put("descriptor", nameHeader);  
        obj.put("detectedTime", System.currentTimeMillis());  
        obj11.put("proximityUUID", "cb10023f-a318-3394-4199-a8730c7c1aec");  
        obj11.put("major", majorDetected);  
        obj11.put("minor", "284");  
        obj11.put("rssi", -81);  
        obj11.put("accuracy", 1.85412312);  
        obj11.put("proximity", "Near");  
        obj.put("data", obj11);  
        array1.put(obj);  
        jsonObject.put("bnm", array1);  
        //-----  
  
        Log.d("Sending Data-" + index11, " ");  
  
        MqttTopic mqttTopic = client4.getTopic("ubeacon/user/request");  
        MqttMessage mqttMessage = new  
MqttMessage(jsonObject.toString().getBytes());  
        mqttMessage.setQos(1);  
        //publish  
        mqttTopic.publish(mqttMessage);  
  
        //Cobain  
        System.out.println("Payload berhasil dikirimkan");  
        Toast.makeText(getApplicationContext(), "Payload is successfully  
sent",  
Toast.LENGTH_LONG).show();  
        index11++;  
  
    } catch (MqttException e) {  
        System.out.println("Masuk MQTT exception");  
        throw new RuntimeException(e);  
  
    } catch (JSONException e) {  
        System.out.println("Masuk JSON Exception");  
    }  
}
```

Keberhasilan pengiriman payload beacon akan dapat ditunjukkan pada bagian *logcat* pada Android Studio. Dalam hal ini, dikirimkan kepada server sebuah payload yang sesuai dengan kebutuhan backend server yang sesuai dengan payload *JSON File* yang diminta pada Presence Insights. Payload JSON tersebut akan menentukan informasi balikan yang akan didapatkan dari server IBM Bluemix. Setelah dilakukan proses publish, maka aplikasi akan berperan sebagai pendengar (*listener*) yang sedang melakukan *subscribe* untuk mendapatkan informasi tersebut.

Selanjutnya, untuk dapat menerima balasan dari server maka perlu dibentuk 3 buah method yang wajib, yaitu `connectionLost()`, `messageArrived()`, dan `deliveryComplete()` sebagai berikut.

```

@Override
public void connectionLost(Throwable throwable) {
}

@Override
public void messageArrived(String s, MqttMessage mqttMessage) throws
Exception {
    JSONObject object = new JSONObject (String.valueOf(mqttMessage));
    String temp_major = object.getString("major").toString();

    try {
        int i;

        String e_uuid, e_major, e_minor, e_url, e_category, e_title,
e_brief,
            e_full, e_time, e_location, e_organizer, e_contact;

        e_uuid = object.getString("proximityUUID").toString();
        e_major = object.getString("major").toString();
        e_minor = object.getString("minor").toString();

        String tempArray = object.getString("data").toString();

        JSONArray dataArray = new JSONArray(tempArray);
        for (i = 0; i < dataArray.length(); i++) {

            JSONObject object2 = dataArray.getJSONObject(i);

            e_url      = object2.getString("image_url").toString();
            e_category = object2.getString("category").toString();
            e_title    = object2.getString("title").toString();
            e_brief    = object2.getString("brief").toString();
            e_full     = object2.getString("full").toString();
            e_time     = object2.getString("time").toString();
            e_location = object2.getString("location").toString();
            e_organizer = object2.getString("organizer").toString();
            e_contact  = object2.getString("contact").toString();

            Events events = new Events(e_url,e_title,e_full,e_brief,
                e_category,e_location, e_organizer, e_contact,
                e_major, e_time, e_uuid, e_minor);
        }
        index11++;
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

@Override
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
    System.out.println("Sending Data-"+j+" Completed");
    j++;
}

```

Dalam hal ini, aplikasi akan menerima balasan berupa JSON file yang terlampir pada bagian `messageArrived()`. Aplikasi akan menerima 12 buah nilai String yang ditampung pada `e_uuid`, `e_major`, `e_minor`, `e_url`, `e_category`, `e_title`, `e_brief`, `e_full`, `e_time`, `e_location`, `e_organizer`, dan `e_contacts`. Seluruh variabel tersebut merupakan informasi yang akan ditampilkan pada tampilan utama dari aplikasi uBeacon ini. Selanjutnya, seluruh variabel akan di *assign* ke dalam sebuah *constructor* `Events` untuk menjadi sebuah Java Object yang akan selanjutnya ditampilkan pada `ListView` pada tampilan utama aplikasi.

2.2.5.5 Fitur penyimpanan dan pengelolaan data pada *local memory* smartphone.

Mobile application uBeacon menggunakan local memory pada smartphone pengguna sebagai tempat penyimpanan sementara bagi data-data berupa data diri pengguna, Uri dari gambar yang telah dipilih pengguna sebagai profile picture, juga sebagai tempat untuk menyimpan state Sign In pengguna. Dalam hal ini, digunakan

SharedPreferences sebagai metode penyimpanan serta pengolahan data-data tersebut diatas.

Shared Preference merupakan metode yang digunakan untuk menyimpan sebuah set key-value dalam jumlah yang tidak terlalu banyak dan disimpan dalam memori local smartphone pengguna. SharedPreferences diakses melalui sebuah file yang terdiri dari beberapa pasang key-value dengan metode sederhana untuk membaca dan menuliskannya. SharedPreferences hanya dapat digunakan untuk membaca dan menulis pasangan key-value.

Pada aplikasi ini, digunakan beberapa file SharedPreferences, sehingga dalam implementasinya digunakan metode `getSharedPreferences()`. Setiap file SharedPreferences dibedakan oleh nama filenya, dan file tersebut dapat diakses (dipanggil) melalui `Context` dalam aplikasi.

Untuk menyimpan status Sign-In pengguna, maka dibuatlah sebuah file SharedPreferences yang dituliskan dalam baris deskripsi kode berikut:

```
//save the login state and store it to local memory
SharedPreferences pref = getSharedPreferences(PREF_NAME, MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.putString(KEY_STATUS, "ok");
editor.commit();
```

Pada baris kode diatas, dibuat sebuah file SharedPreferences yang dapat diakses dengan variable `pref`. File SharedPreferences ini diatur agar bersifat private sehingga aplikasi lain dalam smartphone pengguna tidak dapat mengakses file ini. Untuk melakukan hal tersebut, digunakan `MODE_PRIVATE`. File SharedPreferences harus dinamai dengan penanda yang unik sehingga mudah untuk dikenali oleh aplikasi yang telah dibuat. Nama yang digunakan pada file SharedPreferences yang dibuat adalah `"UBEACON_PREF"` yang diinisialisasi sebagai:

```
private static String PREF_NAME = "UBEACON_PREF";
```

Selanjutnya setelah file SharedPreferences berhasil disiapkan, maka untuk menulis pada file SharedPreferences harus dibuat sebuah `SharedPreferences.Editor` dengan memanggil `edit()` di SharedPreferences. Untuk menyimpan status Sign In pengguna, pada file SharedPreferences disiapkan sebuah key:

```
private static String KEY_STATUS = "login_status";
```

Selanjutnya key tersebut diisi dengan value `"ok"`. Untuk mengakhiri penulisan pada file Shared preference maka diberikan deskripsi kode `editor.commit();`. Baris deskripsi kode untuk penulisan status Sign In pengguna dilakukan tepat saat Sign In berhasil dilakukan.

Sesaat setelah aplikasi di destroy dan diluncurkan lagi, maka secara otomatis akan diluncurkan `SignIn.class` sebagai default. Pada saat class ini diluncurkan, file SharedPreferences ini dipanggil untuk dilakukan cek pada status `SignIn` dalam baris kode dibawah:

```
SharedPreferences pref = getSharedPreferences(PREF_NAME, MODE_PRIVATE);
String status = pref.getString(KEY_STATUS, "null");
System.out.println("status pref: " + status);
if (status.equals("ok")) {
    Intent intent = new Intent(this, Navigation.class);
    startActivity(intent);
    finish();
}
```

Bila value dari status adalah "ok" maka aplikasi akan diteruskan pada menu utama yaitu Navigation.class. sedangkan bila didapati value dari state ini adalah "null" maka proses akan tetap dilanjutkan pada class SignIn.

Saat Sign In berhasil dilakukan server mengirimkan status kepada aplikasi beserta dengan data diri pengguna yang sebelumnya telah diisi pada saat proses Sign Up. Data ini juga disimpan pada local memory dengan file SharedPreferences yang berbeda dari file yang digunakan untuk menyimpan status Sign-In. deskripsi kode pada penyimpanan data diri pengguna dituliskan sebagai berikut:

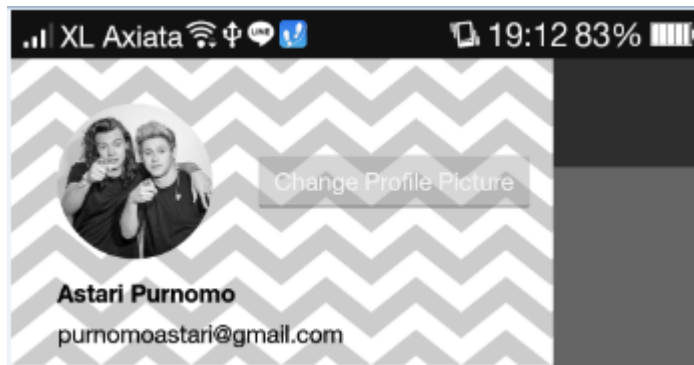
```
//keep the data from server in local memory using sharedPreferences
SharedPreferences pref = getSharedPreferences(Data.SPREF_NAME,
MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.putString("Name", Name);
editor.putString("Email", Email);
editor.putString("Username", Username);
editor.putString("Password", Password);
editor.putString("Birthdate", Birthdate);
editor.putString("NIM", NIM);
editor.putString("Occupation", Occupation);
editor.putString("Interest", Interest);
editor.putString("Unit", Unit);
editor.commit();
```

Data diri pengguna disimpan dalam pasangan key-value seperti diatas. Sebelumnya data-data ini diperoleh dari proses parsing paket JSON yang dikirim sebagai balasan dari server. Begitu value dari masing-masing key pada JSON file berhasil diperoleh dalam bentuk String, nilai tersebut di-passing dan disimpan dalam file SharedPreferences.

Penyimpanan data diri pengguna dalam local memory menggunakan SharedPreferences ini bertujuan untuk validasi dan personalisasi pengguna. Validasi digunakan saat pengguna hendak melakukan edit profile pada data dirinya. Untuk memastikan bahwa hanya pengguna yang melakukan edit profile maka validasi dilakukan dengan meminta pengguna untuk melakukan input password. Password yang dimasukan oleh pengguna selanjutnya dibandingkan dengan password yang telah disimpan aplikasi pada SharedPreferences. Password pada SharedPreferences dipanggil dengan deskripsi sebagai berikut:

```
SharedPreferences pref = getSharedPreferences(Data.SPREF_NAME,
MODE_PRIVATE);
password = pref.getString("Password", "");
```

Untuk memanggil dan menerima data yang telah disimpan dalam file SharedPreferences, file SharedPreferences diinisialisasi kemudian dipanggil menggunakan deskripsi `getString("Password", "")`. Value yang diperoleh kemudian disimpan dalam variabel `password` yang bertipe String. Variable ini yang selanjutnya dibandingkan dengan input yang dimasukan oleh pengguna. Selanjutnya file ini juga dipanggil untuk mendapatkan value dari email pengguna dan nama pengguna untuk ditampilkan pada header Navigation.class. Berikut adalah hasil parsing dari file SharedPreferences:



Gambar 14 Tampilan pada navigation bar ©Dokumentasi Penulis

Selanjutnya SharedPreference juga digunakan dalam melakukan komunikasi antara service-activity. Pada hal ini, dilakukan komunikasi berupa passing data dari MyServiceNotification.class menuju Notificaton.class. Saat aplikasi menerima paket data dari server berupa informasi dari unit dan jurusan pada Service selanjutnya data ini disimpan dalam SharedPreference. Data pada SharedPreference kemudian dipanggil pada Notification.class dan ditampilkan sesuai dengan layout yang telah diatur dalam Notification.xml. Berikut adalah kode deskripsi saat dilakukan penyimpanan data pada SharedPreference di MyServiceNotification.class

```
SharedPreferences pref =
getApplicationContext().getSharedPreferences(Data.SPREF_INFO,
MODE_APPEND);
SharedPreferences.Editor editor = pref.edit();
editor.putString("Image_url", infourl);
editor.putString("Title", title);
editor.putString("Notifications", notifications);
editor.putString("Time", time);
editor.putString("Location", location);
editor.putString("Contact", contact);
editor.commit();
```

Selanjutnya berikut adalah baris deskripsi saat data dalam file SharedPreference diambil dan di parsing dalam layout:

```
haredPreferences pref = getSharedPreferences(Data.SPREF_INFO,
MODE_PRIVATE);
Image = pref.getString("Image_url", "");
Title = pref.getString("Title", "");
Time = pref.getString("Time", "");
Notif = pref.getString("Notifications", "");
Location = pref.getString("Location", "");
Contact = pref.getString("Contact", "");
```

SharedPreference juga digunakan untuk menyimpan Uri dari gambar yang dipakai pengguna sebagai profile picture. Setelah pengguna berhasil memilih gambar, Uri dari gambar akan disimpan dalam SharedPreference seperti baris kode dibawah:

```
SharedPreferences pref = getSharedPreferences(PREF_PROFPIC,
MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.putString(KEY_PROFPIC, imageUri.toString());
editor.commit();
```

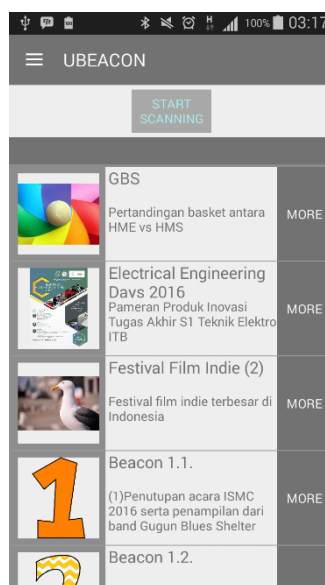
File ini akan dipanggil setiap saat pengguna berhasil masuk ke Navigaation.class, sehingga gamabr yang dipilih pengguna sebagai profile picture akan tetap tersimpan.

Selanjutnya untuk proses Log out, file SharedPreference yang sebelumnya telah dibuat pada saat dilakukan penyimpanan status Sign In dipanggil dan ditimpa dengan value “null” sehingga saat aplikasi kembali diluncurkan dan baris kode cek status berjalan maka value "null" akan membuat aplikasi meminta pengguna untuk melakukan SignIn. Berikut adalah kode yang dituliskan saat Log out dilakukan:

```
SharedPreferences pref = getSharedPreferences(PREF_NAME, MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.putString(KEY_STATUS, "null");
editor.commit();
```

2.2.5.6 Graphic User Interface(GUI) sebagai interaksi antarmuka dengan pengguna.

Aplikasi uBeacon memiliki fitur utama untuk melakukan scan terhadap Bluetooth Beacon untuk mengambil informasi yang sesuai dengan keberadaannya. Tampilan utama dari aplikasi ini akan ditampilkan dalam sebuah List View sehingga dapat menampung data yang banyak dan dapat dilihat dengan melakukan scroll. Berikut tampilan yang di desain pada aplikasi ini.

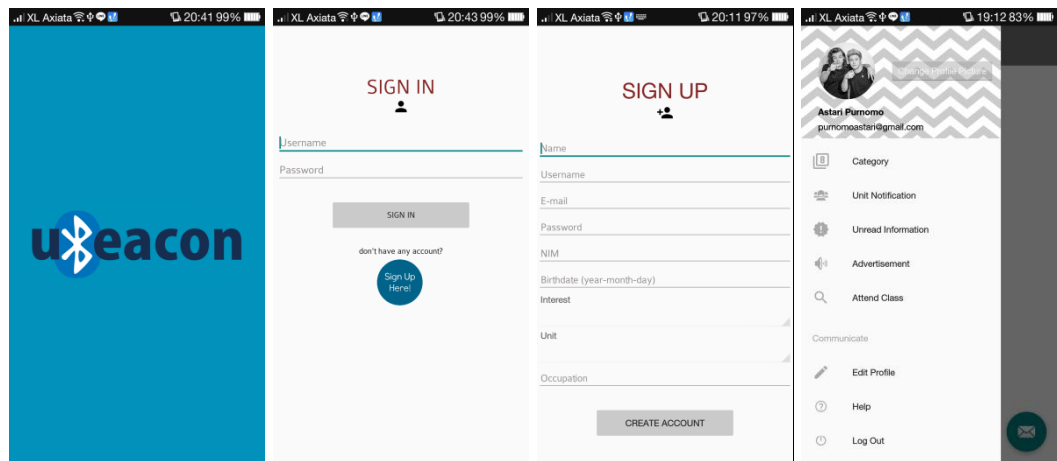


Gambar 15 Tampilan utama – homepage ©Dokumentasi Penulis

Secara keseluruhan ada tiga keadaan(*state*) yang dialami pengguna setelah mengunduh aplikasi uBeacon, yaitu:

1. Pengguna yang belum memiliki akun dalam hal ini harus melakukan proses registrasi atau Sign-Up terlebih dahulu.
2. Pengguna yang sudah memiliki akun namun belum melakukan Sign-In.
3. Pengguna yang telah memiliki akun dan telah melakukan Sign-In.

Semua proses yang dijalankan pada aplikasi secara umum dibagi kedalam dua buah proses yaitu Foreground Process dan Background Process. Foreground process dimulai dari splash screen yang muncul saat pengguna meluncurkan uBeacon mobile application seperti yang terlihat pada gambar dibawah.



Gambar 16 Tampilan antarmuka aplikasi Android “uBeacon” yang dibuat. ©Dokumentasi Penulis

Pada tampilan pertama dari gambar diatas, merupakan splash screen dari Android. Splash screen adalah tampilan awal dari aplikasi sebelum masuk ke state atau menu utama dari aplikasi. Untuk implementasi dari splash screen, dibentuk sebuah aktivitas baru yang dideklarasikan dalam android manifest. Aktivitas ini diatur sebagai **LAUNCHER** sedangkan aktivitas utama pada mobile application diatur sebagai **DEFAULT**. Selanjutnya dilakukan pengaturan waktu untuk menentukan rentang waktu splash screen muncul dalam layar smartphone menggunakan sebuah class Animation. Pada kasus kali ini digunakan splash screen dengan animasi *fade_out*. Setelah animasi ini berakhir, tampilan akan masuk ke state selanjutnya tergantung dengan state yang sedang dialami pengguna (Sign-In atau Signed-In).




```
@Override
public void onAnimationEnd(Animation animation) {
    //memulai splash
    iv.startAnimation(an2);
    //mengakhiri splash
    finish();
    Intent i = new Intent(SplashWelcome.this, SignIn.class);
    startActivity(i);
}
```


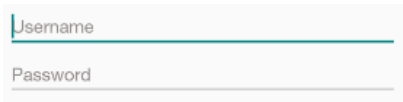


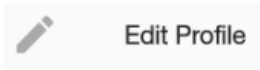
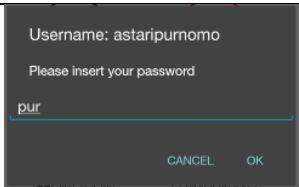
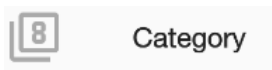
Selanjutnya setelah durasi animasi *abc_fade_out* pada splash screen berakhir, program akan memanggil default activity. Pada kasus ini, default activity yang dimaksud tergantung dari state pengguna apakah sebelumnya pengguna telah melakukan proses Sign-In. Pada sepenggal baris kode diatas, dapat dijabarkan bahwa setelah animasi berakhir akan dilakukan pergantian tampilan atau state yang dilakukan menggunakan Intent. Intent merupakan barisan deskripsi yang menyatakan operasi yang harus ditampilkan, pada hal ini Intent dilakukan dari splash screen menuju layout Sign-In. Pada transisi ini akan dilakukan pengecekan pada SharedPreferences dimana sebelumnya telah di-store state yang sedang dijalani pengguna. Jika sebelumnya pengguna telah melakukan Sign-In maka pengguna langsung masuk ke tampilan menu utama, tapi jika Sign-in belum dilakukan akan dilakukan proses Sign-In terlebih dahulu untuk masuk ke dalam menu utama.






Pada table dibawah akan dijabarkan setiap fungsi pada layout yang digunakan pada keseluruhan aplikasi sebelum dibahas lebih lanjut:






Tabel XIII – Penjelasan antarmuka “uBeacon”

Nama Fitur	Tampilan	Deskripsi
------------	----------	-----------

Sign In		<ul style="list-style-type: none"> • Mengubah input dari pengguna berupa Username dan Password ke dalam bentuk JSON file. • Mengirimkan data pengguna berupa username dan password ke database untuk dilakukan pengecekan. • Apabila data tidak cocok dengan database maka akan ditampilkan pesan “Wrong password” atau ”Username not exist” sedangkan bila input yang dimasukan cocok dengan database akan diterima pesan status “Sign-in success”. • Aplikasi akan menampilkan halaman utama dari program berupa Navigation.class yang menampilkan semua informasi dari beacon ketika Sign-In berhasil dilakukan. • Balasan dari server akan disimpan dalam memory local smartphone pengguna menggunakan Shared Preference untuk menyimpan status login serta data diri pengguna. • Saat Sign In berhasil dilakukan, aplikasi menjalankan Service pada background dan berubah menjadi subscriber ke alamat topic tertentu dan harus selalu siap menerima data dari server kapanpun itu.
Sign-Up Here		<ul style="list-style-type: none"> • Bila pengguna belum pernah melakukan registrasi sebelumnya maka pengguna diminta untuk melakukan registrasi terlebih dahulu untuk mendapatkan akun pribadinya. Dengan meng’klik’ tombol ini maka akan ditampilkan layout Sign Up menggunakan baris kode Intent. • Apabila layout telah berubah ke layout Sign Up maka akan ditampilkan pesan “Please make a new account”.
Create Account		<ul style="list-style-type: none"> • Mengubah input dari pengguna berupa Nama, Username, Password, Email, NIM, Birthdate, Interest, Unit, dan Occupation ke dalam bentuk JSON file. • Mengirimkan data pengguna ke database untuk dilakukan pengecekan apakah username yang dipilih sudah pernah digunakan. • Apabila data tidak cocok dengan database maka akan ditampilkan pesan “Username taken” dan pengguna harus melakukan input username ulang. Sedangkan bila input yang dimasukan cocok dengan database akan diterima pesan status “Registration success”. • Aplikasi akan menampilkan halaman Sign-In setelah proses registrasi berhasil dilakukan.

Menu		<ul style="list-style-type: none"> Dengan meng'klik'tombol menu akan ditampilkan navigation drawer atau pilihan menu yang ada pada aplikasi berupa sorting informasi sesuai dengan jenis informasi yang ada. Terdapat 8 pilihan dalam menu yaitu Category, Unit Notification, Unread Information, Advertisement, Attend Class, Edit Profile, Help, dan Log Out.
Parameter input pengguna		<ul style="list-style-type: none"> Tampilan untuk meminta input pengguna berupa username dan password.
Profile Picture		<ul style="list-style-type: none"> Dengan meng'klik'tombol ini maka aplikasi akan melakukan aksi dengan melakukan akses ke gallery smartphone pengguna sehingga pengguna dapat memilih file .jpg atau .png untuk diubah kedalam bitmap dan ditampilkan dalam ImageView. Uri dari file tersebut disimpan dalam Shared Preference agar dapat diakses sewaktu aplikasi dibuka kembali.
Nama dan Email pengguna		<ul style="list-style-type: none"> Pada navigation header juga terdapat informasi berupa nama dan email pengguna. Informasi ini ditarik dari Shared Preference yang sebelumnya diperoleh dari proses Sign In.
Edit Profile		<ul style="list-style-type: none"> Dengan meng'klik; tombol ini maka aplikasi akan meminta input berupa password pengguna untuk validasi. Password pengguna diperoleh aplikasi saat dilakukan proses Sign-In. Jika input password salah dan tidak berhasil dilakukan maka akan muncul dialog yang menyatakan bahwa password yang diinput pengguna salah. Sedangkan jika input yang dilakukan pengguna benar maka layout akan berubah ke xml dari Edit Profile.
Validasi password untuk Edit Profile		<ul style="list-style-type: none"> Tampilan dialog saat aplikasi meminta input dari pengguna berupa password pengguna untuk validasi dilakukannya edit pada profile pengguna.
Category		<ul style="list-style-type: none"> Dengan meng'klik' tombol ini maka aplikasi akan menampilkan xml dari fragment category. Ditampilkan informasi berupa list category yang dapat dipilih oleh pengguna. Terdapat 8 kategori, yaitu Seminar, Campus Info, Promotion, Unit and Himpunan Event, Workshop, Concert, Sprots, Event.

Seminar		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list berita di kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text dari informasi. • Mengubah JSON file kedalam layout aplikasi (parsing JSON).
Campus Info		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list berita di kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text dari informasi. • Mengubah JSON file kedalam layout aplikasi.
Promotion		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list promosi di kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text dari informasi. • Mengubah JSON file kedalam layout aplikasi.
Unit and Himpunan Event		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list acara yang diselenggarakan oleh unit atau himpunan di kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text dari informasi. • Mengubah JSON file kedalam layout aplikasi.
Workshop		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list workshop di kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text

		<p>dari informasi.</p> <ul style="list-style-type: none"> • Mengubah JSON file kedalam layout aplikasi.
Concert		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list konser yang diselenggarakan di kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text dari informasi. • Mengubah JSON file kedalam layout aplikasi.
Sport		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list acara olahraga di kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text dari informasi. • Mengubah JSON file kedalam layout aplikasi.
Event		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka aplikasi akan melakukan scan beacon. • Ditampilkan informasi berupa list acara yang berasal dari luar kampus pada layout image dan textview yang telah ditentukan. • Aplikasi berubah menjadi subscriber untuk menerima informasi dari backend berupa JSON file yang berisi informasi url image dan text dari informasi. • Mengubah JSON file kedalam layout aplikasi.
Help		<ul style="list-style-type: none"> • Dengan meng'klik' tombol ini maka akan ditampilkan tampilan berupa textview mengenai aplikasi
Logout		<ul style="list-style-type: none"> • Dengan menekan tombol ini, akan ditampilkan dialog yang meminta konfirmasi pengguna bila pengguna akan keluar dari kondisi <i>signed-in</i>. Perlu dilakukan <i>sign-in</i> untuk dapat masuk kembali ke aplikasi.

The image shows a vertical registration form. It contains the following input fields from top to bottom: Name, Username, E-mail, Password, NIM, Birthdate (year-month-day), Interest, Unit, and Occupation. Below these fields is a button labeled 'CREATE ACCOUNT'.

- Tampilan untuk meminta input pengguna berupa Nama, Username, Password, Email, NIM, Birthdate, Interest, Unit, dan Occupation.

Pengaturan layout dilakukan pada file bertipe .xml. Pengaturan layout secara sederhana dan efisien dimaksudkan agar pengguna mampu berinteraksi dengan server secara mudah.

Pada tampilan Sign-In, Sign-Up dan Edit Profile, layout disusun secara linear dengan orientasi vertical untuk memudahkan dilakukannya pengaturan pada tampilan. Lebar dan panjang diatur dalam tipe `match_parent` sehingga cocok untuk diaplikasikan pada berbagai tipe layar smartphone. Tipe `match_parent` membuat tampilan dioperasikan sesuai dengan layar yang dipakai. Berikut adalah beberapa baris kode deskripsi yang menunjukkan pengaturan tampilan:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.astaripurnomo.ubeacon.SignIn"
    android:id="@+id/SignIn"
    android:tag="@string/app_name">
```

Setiap komponen pada tampilan diatur pada file .xml ini. Pada tampilan Sign In, Sign Up, Edit Profile digunakan beberapa komponen, diantaranya EditText, Button, TextView, ImageView, dan ImageButton. EditText digunakan sebagai komponen untuk meminta masukan dari pengguna dengan input berupa text baik itu huruf maupun angka. Pengaturan pada EditText dilakukan sebagai berikut:

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPassword"
    android:ems="10"
    android:id="@+id/Password"
    android:layout_gravity="center_horizontal"
    android:hint="Password"
    android:textColorHint="#908989"
    android:maxLines="1"
    android:singleLine="true"
    android:paddingLeft="5dp"
    android:paddingRight="5dp"/>
```

Panjang dan lebar dari EditText disesuaikan dengan kebutuhan yang dibutuhkan. Dalam hal ini lebar disesuaikan dengan lebar layar yang akan menampilkan tampilan ini, sedangkan tinggi dari EditText disesuaikan dengan input pengguna. Tipe input

yang dapat dimasukkan oleh pengguna juga diatur dalam file .xml. Pada aplikasi ini tipe input diatur sesuai dengan kebutuhan yang diperlukan oleh system. Untuk memudahkan pengguna dalam memasukan input, maka dalam EditText diberikan hint.

Button digunakan sebagai komponen yang dapat digunakan untuk memulai aktivitas baru. Aktivitas yang dapat diatur dalam metode `setOnClickListener(new View.OnClickListener())`. Dalam metode tersebut dapat dituliskan berbagai aktivitas yang diinginkan. Berikut adalah baris pengaturan pada salah satu Button yang dilakukan:

```
<Button
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:text="Sign In"
    android:id="@+id/SignInBtn"
    android:enabled="false"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="20dp"
    android:clickable="true" />
```

Agar pada saat button di click dapat dilakukan pengaturan pada file .java, maka harus dipastikan bahwa clickable harus bernilai true.

TextView merupakan metode digunakan untuk menampilkan text pada tampilan tanpa dilakukan tindakan apapun (meskipun sebenarnya bisa dilakukan). Berikut adalah baris kode TextView:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="SIGN IN"
    android:textSize="30dp"
    android:layout_marginTop="70dp"
    android:textColor="#842424"
    android:layout_gravity="center_horizontal"
    android:id="@+id/SignInView" />
```

ImageButton pada aplikasi ini digunakan untuk transisi antara tampilan Sign In dengan tampilan Sign Up. Berikut adalah baris kode deskripsi ImageButton yang digunakan:

```
<ImageButton
    android:layout_width="75dp"
    android:layout_height="75dp"
    android:id="@+id/imageButton"
    android:layout_gravity="center_horizontal"
    android:src="@drawable/signup"
    android:layout_centerHorizontal="true"
    android:adjustViewBounds="true"
    android:cropToPadding="false"
    android:layout_below="@+id/hlTopBar"
    android:background="#00000000"
    android:scaleType="fitXY" />
```

2.2.5.7 Membangun fitur personalisasi penggunaan aplikasi dalam bentuk akun pengguna (*user account*).

Fitur personalisasi pada aplikasi ini diimplementasikan dengan dibuatnya akun pribadi bagi pengguna. Pembuatan fitur akun bagi pengguna akan mempermudah pengguna dalam menerima informasi sesuai dengan minat dan bidang favoritnya. Pengaturan personalisasi bagi pengguna ini dilakukan kedalam tiga tahap, yaitu:

Sign Up

Sign In

Edit Profile

Masing-masing class ini memiliki fungsinya masing-masing. Berikut akan dijelaskan lebih lanjut mengenai masing-masing dari class ini.

Sign-Up

Sign Up digunakan sebagai state untuk meminta data diri pengguna yang akan berguna untuk penyebaran informasi bagi setiap individu pengguna. Terdapat 9 komponen yang meminta masukan dari pengguna, diantaranya: Nama, username, password, email, NIM, tanggal dan tahun lahir, unit, minat (interest), dan pekerjaan (occupation).

Sign-Up dinyatakan sebagai Activity yang terdiri dalam dua format yaitu .xml dan .java. Pada bagian .xml dilakukan proses pengaturan layout atau tampilan sesuai yang dikehendaki sehingga diharapkan mudah untuk dipahami pengguna. Pada bagian .java segala pengaturan terhadap input pengguna dilakukan. Sign-Up mengimplementasikan protocol MqttCallback yang memiliki tiga fungsi bawaan yaitu:

```
@Override
public void connectionLost(Throwable throwable) {}

@Override
public void messageArrived(String s, MqttMessage mqttMessage) throws
Exception {}

@Override
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {}
```

```
CreateBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        //publish
        JSONObject jsonObject = new JSONObject();
        try {

            jsonObject.put("_id", UsernameTxt.getText().toString());
            jsonObject.put("password", PasswordTxt.getText().toString());
            jsonObject.put("fullname", NameTxt.getText().toString());
            jsonObject.put("nim", NIMTxt.getText().toString());
            jsonObject.put("email", EmailTxt.getText().toString());
            jsonObject.put("birthdate", BirthdateTxt.getText().toString());
            jsonObject.put("occupation", OccupationTxt.getText().toString());
            jsonObject.put("interest",
multiSelectionSpinner.getSelectedItemsAsString());
            jsonObject.put("unit",
unitSelectionSpinner.getSelectedItemsAsString());

            String cacheDir = getCacheDir().getAbsolutePath();

            MqttClient client = new
MqttClient("tcp://test.mosquitto.org:1883", "ubeacon/user/signup",
```

```

        new MqttDefaultFilePersistence(cacheDir));
        client.connect();
        client.setCallback(SignUp.this);
        client.subscribe("ubeacon/user/signup/" +
UsernameTxt.getText().toString());

        MqttTopic mqttTopic = client.getTopic("ubeacon/user/signup");
        MqttMessage mqttMessage = new
MqttMessage(jsonObject.toString().getBytes());
        mqttMessage.setQos(1);

        mqttTopic.publish(mqttMessage);

    } catch (MqttException e) {
        throw new RuntimeException(e);
    } catch (JSONException e) {}
    }
}
);

```

Pada baris penggalan kode diatas, saat tombol “Create Account” ditekan, maka input pengguna akan dipaketkan menjadi JSON file melalui baris kode:

```

jsonObject.put("key",value);

```

Selanjutnya akan dibuat koneksi ke broker dan topic melalui MQTT yang dinyatakan dalam:

```

MqttClient client = new
MqttClient("tcp://test.mosquitto.org:1883","ubeacon/user/signup",
        new MqttDefaultFilePersistence(cacheDir));
client.setCallback(SignUp.this);
client.connect();
client.subscribe("ubeacon/user/signup/" + UsernameTxt.getText().toString());

```

Broker yang dituju adalah dengan topic. Selanjutnya setelah broker `tcp://test.mosquitto.org:1883` dengan port 1883 dan topic `ubeacon/user/signup` diinisialisasi, dilakukan koneksi ke broker dan dilakukan subscribe terlebih dahulu untuk memastikan bahwa broker siap digunakan.

Selanjutnya, paket JSON yang sudah dibuat diubah menjadi `MqttMessage` dan dikirim atau di-publish ke broker melalui baris kode:

```

mqttTopic.publish(mqttMessage);

```

Selanjutnya aplikasi berubah kembali menjadi subscriber dan menunggu balasan dari backend mengenai JSON file yang telah dikirim. Aplikasi akan menerima balasan dari backend berupa JSON file dalam `public void messageArrived(String s, MqttMessage mqttMessage) throws Exception {}` sebagai berikut:

```

@Override
public void messageArrived(String s, MqttMessage mqttMessage) throws
Exception {
    System.out.println("Message arrived: " + mqttMessage.toString());
    try {
        JSONObject object = new JSONObject (String.valueOf(mqttMessage));
        status = object.getString("status");
        if (status.equals("Registration success")) {
            Intent intent = new Intent(this, SignIn.class);
            startActivity(intent);
            finish();
        } else if (status.equals("Username taken")) {
            UsernameTxt.setError(getString(R.string.UsernameTaken));
        }
    }
}

```

```

    }

    } catch (JSONException e) {
        e.printStackTrace();
    }
}

```

Balasan dari backend akan diubah kedalam JSON file untuk selanjutnya dilakukan parsing dan diambil value dari paket JSON dengan key “status”. Jika status yang diterima berupa "Registration success" maka akan dilakukan pergantian layout ke Sign-In. Sedangkan jika value yang diterima berupa "Username taken" maka akan ditampilkan pesan yang memberitahu pengguna bahwa username yang dipilih telah digunakan sehingga harus diganti dengan username yang lain.

Sign-In

Sign In digunakan sebagai jembatan bagi pengguna untuk memasuki menu utama. Secara garis besar pada proses Sign-In akan diminta masukan dari pengguna berupa username dan password yang sebelumnya telah ditentukan dan dibuat pada proses Sign Up. Input dari pengguna selanjutnya akan dikirim ke server menggunakan MQTT. Jika proses berhasil dilakukan maka tampilan akan berubah ke menu utama, tetapi bila proses gagal dilakukan maka akan ditampilkan pesan untuk melakukan input ulang karena ada kesalahan dalam username atau password yang dimasukan.

Sign-In dinyatakan sebagai Activity yang terdiri dalam dua format yaitu .xml dan .java. Pada bagian .xml dilakukan proses pengaturan layout atau tampilan sesuai yang dikehendaki sehingga diharapkan mudah untuk dipahami pengguna. Pada bagian .java segala pengaturan terhadap input pengguna dilakukan. Sign-In mengimplementasikan protocol MqttCallback yang memiliki tiga fungsi bawaan seperti yang ditunjukkan pada bagian Sign-Up diatas.

Sesuai dengan tampilan Sign-In pada gambar diatas, pengguna diminta untuk memberikan input berupa Username dan Password. Selanjutnya input dari pengguna akan diubah kedalam JSON file pada saat button “SIGN IN” ditekan sesuai dengan potongan baris kode berikut:

```

SignInBtn.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {

        //making a json object
        JSONObject jsonObject = new JSONObject();
        try {
            //input from user being packaged in json form
            jsonObject.put("_id", UsernameTxt.getText().toString());
            jsonObject.put("password", PasswordTxt.getText().toString());

            String cacheDir = getCacheDir().getAbsolutePath();

            //broker and topic inisialisation
            client = new MqttClient("tcp://test.mosquitto.org:1883",
            "ubeacon/user/signin", new MqttDefaultFilePersistence(cacheDir));

            //inisialisation callback method in this fragment
            client.setCallback(SignIn.this);
            //connect using Mqtt
            client.connect();
            //subscribe
            client.subscribe("ubeacon/user/signin/" + UsernameTxt.getText().toString());

```

```

MqttTopic mqttTopic = client.getTopic("ubeacon/user/signin");
MqttMessage mqttMessage = new MqttMessage(jsonObject.toString().getBytes());
//inisialisasi Qos
mqttMessage.setQos(1);
mqttTopic.publish(mqttMessage);
} catch (MqttException e) {
    throw new RuntimeException(e);
} catch (JSONException e) {
    }
}
}
);
}

```

Dari baris kode diatas, input dari pengguna diubah menjadi JSON file sesuai dengan deskripsi berikut:

```

JSONObject jsonObject = new JSONObject();
try {
    //input from user being packaged in json form
    jsonObject.put("_id", UsernameTxt.getText().toString());
    jsonObject.put("password", PasswordTxt.getText().toString());
}

```

Selanjutnya baris deskripsi ini dikirimkan melalui protocol MQTT via Internet setelah sebelumnya dilakukan koneksi dengan broker melalui topic `ubeacon/user/signin/` yang telah disiapkan dan dipastikan bahwa broker siap untuk menerima passing data.

Pada ketiga fungsi yang dibawa oleh MqttCallback dilakukan pengaturan pada saat fungsi `messageArrived` dijalankan. Deskripsi fungsi yang dilakukan adalah sebagai berikut:

```

@Override
public void messageArrived(String s, MqttMessage mqttMessage) throws
Exception {
    System.out.println("Message arrived: " + mqttMessage.toString());
    try {
        JSONObject object = new JSONObject(String.valueOf(mqttMessage));

        loginStatus = object.getString("status");

        switch (loginStatus) {
            case "Sign-in success":

                Name = object.optString("fullname");
                Email = object.optString("email");
                Username = object.optString("_id");
                Password = object.optString("password");
                Birthdate = object.optString("birthdate");
                NIM = object.optString("nim");
                Occupation = object.optString("occupation");
                Interest = object.optString("interest");
                Unit = object.optString("unit");

                SharedPreferences pref = getSharedPreferences(PREF_NAME,
MODE_PRIVATE);

                SharedPreferences.Editor editor = pref.edit();
                editor.putString(KEY_STATUS, "ok");
                editor.commit();

                Intent intent = new Intent(this, Navigation.class);
                startActivity(intent);
            }
        }
    }
}

```

```

        finish();
        break;
    case "Username not exist": {
        System.out.println("Login Gagal");
        UsernameTxt.setError(getString(R.string.UsernameWrong));
        break;
    }
    default: {
        System.out.println("Login Gagal");
        PasswordTxt.setError(getString(R.string.PasswordWrong));
        break;
    }
}
}

```

Pada baris kode diatas, aplikasi yang sebelumnya dijalankan sebagai publisher ke topic berubah menjadi subscriber menuju topic. Melalui baris tersebut maka aplikasi siap untuk menerima balasan dari backend tepat setelah aplikasi berhasil mengirimkan JSON file ke backend. Data balasan yang berasal dari backend akan diubah menjadi JSON file untuk selanjutnya diambil value dari setiap key yang dibawa.

Pada implementasinya, dilakukan perbandingan string dari value balasan server dengan key "status". Setiap value dari key "status" diberikan perlakuan yang berbeda-beda. Sign-In yang berhasil ditandai dengan diterimanya balasan dari server berupa value "Sign-in success" pada key "status". Setiap value dari file JSON yang dibawa akan disimpan pada SharedPreferences untuk digunakan bila pengguna ingin melakukan Edit Profile. Pada bagian ini, state Sign-In pengguna juga disimpan pada SharedPreferences. Jika proses Sign-In berhasil dilakukan, maka value "ok" akan disimpan untuk selanjutnya akan dipanggil dan dilakukan perbandingan pada saat Sign-In.class dipanggil.

```

//checking login state
SharedPreferences pref = getSharedPreferences(PREF_NAME, MODE_PRIVATE);
String status = pref.getString(KEY_STATUS, "null");
System.out.println("status pref: " + status);
if (status.equals("ok")) {
    Intent intent = new Intent(this, Navigation.class);
    startActivity(intent);
    finish();
}

```

Dari baris kode berikut dapat dilihat bahwa saat status Sign-In pengguna adalah "ok" maka tampilan layar akan langsung dioperasikan pada Navigation.class menggunakan kode deskripsi Intent, sedangkan jika status Sign-In pengguna berupa "null" maka aplikasi akan tetap dijalankan pada SignIn.class.

Navigation

Navigation merupakan main activity yang digunakan pada aplikasi ini. Navigation berperan sebagai parent fragment dari aplikasi ini. Pada Navigation dapat dilakukan beberapa fitur diantaranya scanning Beacon yang merupakan tujuan utama dari aplikasi ini dibuat, personalisasi informasi (push notification dari unit, interest, dan jurusan pengguna yang dilakukan inputnya pada Sign-Up), dan attend class dimana pengguna tidak perlu lagi melakukan absensi karena absensi dengan sangat sederhana dapat dilakukan melalui aplikasi uBeacon.

Pada saat aplikasi menjalankan Navigation.class, secara otomatis aplikasi akan melakukan running pada MyServiceNotification.class, yang dinyatakan dalam baris kode dibawah:

```
Intent i = new Intent(this, MyServiceNotification.class);
bindService(i,notifConnection, Context.BIND_AUTO_CREATE);
```

Service diatas dimaksudkan sebagai background process berupa Service dimana pada background, aplikasi akan dijalankan sebagai subscriber selama aplikasi di-running. Hal ini bertujuan untuk membuat aplikasi siap menerima notification dari backend. Service pada bagian notification ini akan dibahas lebih lanjut pada bagian Service.

Pada awal Navigation.class dijalankan, aplikasi akan membaca dan mengambil file SharedPreferences berupa nama, username, email, dan password yang sebelumnya telah diperoleh dari proses Sign-In untuk dipassing ke Navigation Header. Pada bagian Navigation header akan ditampilkan nama pengguna serta email pengguna seperti gambar diatas. Berikut adalah baris kode deskripsi Navigation Header:

```
SharedPreferences pref = getSharedPreferences(Data.SPREF_NAME,
MODE_PRIVATE);
nameHeader = pref.getString("Name", "");
emailHeader = pref.getString("Email", "");
username = pref.getString("Username", "");
password = pref.getString("Password", "");

NavigationView navigationView = (NavigationView) findViewById(nav_view);
navigationView.setNavigationItemSelectedListener(this);
View header = navigationView.getHeaderView(0);
name = (TextView) header.findViewById(R.id.username);
email = (TextView) header.findViewById(R.id.email);
imagepick = (Button) header.findViewById(R.id.btn_pick);
imageView = (ImageView) header.findViewById(R.id.profile_image);
name.setText(nameHeader);
email.setText(emailHeader);
imagepick.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
        photoPickerIntent.setType("image/*");
        startActivityForResult(photoPickerIntent, SELECT_PHOTO);
    }
});
```

Selain melakukan setting pada tampilan nama dan email pengguna, dapat pula dilakukan perubahan pada profile picture pengguna. Pada aksi ini diperlukan integrasi dari fitur ini juga smartphone pengguna. Fitur ini akan meminta pengguna untuk melakukan pencarian gambar pada folder gallery smartphonenya untuk selanjutnya dipilih satu dan diset menjadi profile picture pada Navigation Header. Untuk melakukan aksi ini saat button “Change Profile Picture” akan dilakukan pergantian operasi dengan baris kode berikut:

```
imagepick.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
        photoPickerIntent.setType("image/*");
        startActivityForResult(photoPickerIntent, SELECT_PHOTO);
    }
});
```

```
});
```

Pada operasi mengganti profile picture ini digunakan `Intent.ACTION_PICK`. Operasi ini merupakan perintah built in dari Android yang dapat digunakan untuk membantu pengguna mengambil sebuah gambar dari sebuah data source. Pada pengaplikasiannya digunakan sebuah button “Change Profile Picture” yang akan memanggil `Intent.ACTION_PICK` dan meminta pengguna untuk mengambil gambar dari gallery smartphone mereka. Metode `setType()` digunakan untuk mendefinisikan kriteria filter yang digunakan. Metode ini akan memanggil default galley sehingga pengguna dapat memilih gambar pada gallery smartphonenya. Selanjutnya setelah gambar dipilih, hasilnya akan kembali ke main activity dalam hal ini `Navigation.class` dan hasilnya akan dikembalikan pada metode `onActivityResult()`. Pada `onActivityResult()` akan diterima Uri dari gambar untuk selanjutnya dikonversi menjadi `Bitmap` dan ditampilkan pada layout yang telah disediakan. Berikut adalah barisan kode yang digunakan:

```
Override
protected void onActivityResult(int requestCode, int resultCode, Intent
imageReturnedIntent) {
    super.onActivityResult(requestCode, resultCode, imageReturnedIntent);

    switch(requestCode) {
        case SELECT_PHOTO:
            if(resultCode == RESULT_OK) {
                try {
                    final Uri imageUri = imageReturnedIntent.getData();
                    System.out.println("uri" + imageUri);
                    final InputStream imageStream =
getContentResolver().openInputStream(imageUri);
                    System.out.println("stream" + imageStream);
                    final Bitmap selectedImage =
BitmapFactory.decodeStream(imageStream);
                    System.out.println("selected" + selectedImage);
                    imageView.setImageBitmap(selectedImage);
                    SharedPreferences pref =
getSharedPreferences(PREF_PROFPIC, MODE_PRIVATE);
                    SharedPreferences.Editor editor = pref.edit();
                    editor.putString(KEY_PROFPIC, imageUri.toString());
                    editor.commit();

                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }
            }
    }
}
```

Untuk memperoleh Uri dari gambar yang telah dipilih dan melakukan konversi Uri menjadi `Bitmap` digunakan baris kode sebagai berikut:

```
imageStream = getContentResolver().openInputStream(status);
Bitmap selectedImage = BitmapFactory.decodeStream(imageStream);
```

Selanjutnya untuk menyimpan state gambar pengguna, maka Uri dari gambar akan disimpan pada `SharedPreferences` sebagai berikut:


```
//save Uri from selected picture
SharedPreferences pref = getSharedPreferences(PREF_PROFPIC, MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.putString(KEY_PROFPIC, imageUri.toString());
editor.commit();
```

Selanjutnya setelah disimpan, Uri akan dipanggil setiap aplikasi diluncurkan sesuai dengan baris kode berikut:

```
//get Uri that stored in SharedPreferences
SharedPreferences prefss = getSharedPreferences(PREF_PROFPIC, MODE_PRIVATE);
Uri status = Uri.parse(prefss.getString(KEY_PROFPIC, ""));

try {
    //final Uri imageUri = status;
    System.out.println("uri" + status);
    InputStream imageStream;
    imageStream = getContentResolver().openInputStream(status);
    Bitmap selectedImage = BitmapFactory.decodeStream(imageStream);
    imageView.setImageBitmap(selectedImage);
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

Pada Navigation.class juga dilakukan pengaturan pada drawer yang ada, dimana akan dilakukan pengaturan untuk setiap item yang telah dideklarasikan baik dalam .xml maupun .java. Beberapa baris pengaturan yang dilakukan adalah sebagai berikut:

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == nav_category) {
        //Set the fragment initially
        CategoryFragment fragment = new CategoryFragment();
        FragmentTransaction fragmentTransaction =
            getSupportFragmentManager().beginTransaction();
        fragmentTransaction.replace(fragment_container, fragment);
        fragmentTransaction.commit();
    } else if (id == nav_unitnotif) {
        //Set the fragment initially
        NewsFragment fragment = new NewsFragment();
        FragmentTransaction fragmentTransaction =
            getSupportFragmentManager().beginTransaction();
        fragmentTransaction.replace(fragment_container, fragment);
        fragmentTransaction.commit();
    }
}
```

Untuk setiap pilihan menu, digunakan icon dan title yang telah dilakukan pengaturan layoutnya seperti pada sepenggal file .xml sebagai berikut:

```
<group android:checkableBehavior="single">

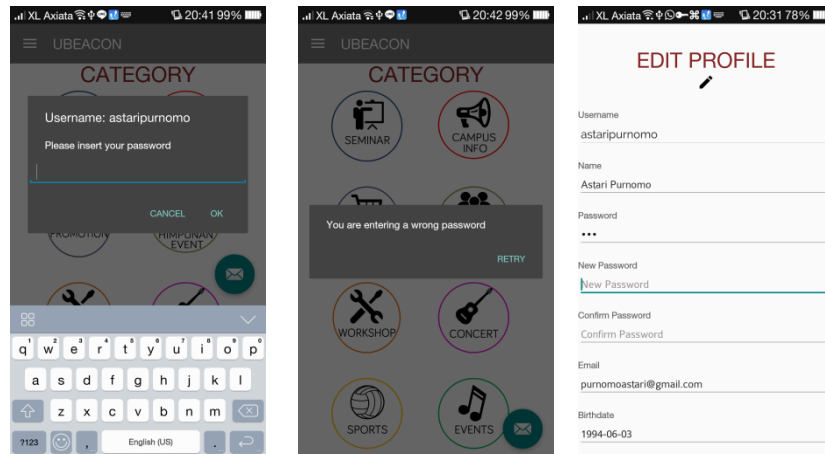
    <item
        android:id="@+id/nav_category"
        android:icon="@drawable/ic_filter_8"
        android:title="Category" />
```

Saat icon pada layout ditekan maka aplikasi akan melakukan perubahan tampilan sesuai dengan baris deskripsi diatas. Untuk setiap menu akan digunakan tampilan dalam sebuah fragment yang telah diatur layoutnya dalam sebuah FrameLayout

fragment container. Pada perubahan setiap tampilan akan digunakan transaksi setiap fragment dalam metode replace. Metode ini berupa pergantian layout lama menjadi layout fragment baru dalam sebuah wadah fragment container.

Edit Profile

Pada Edit Profile, pengguna diminta untuk melakukan validasi sebelum melakukan perubahan pada profile mereka sebelumnya. Validasi yang dilakukan berupa masukan dari pengguna yaitu password. Jika password yang dimasukan salah, maka perubahan pada profile pengguna tidak dapat dilakukan. Aplikasi akan melakukan perbandingan input pengguna dengan value password pengguna yang sebelumnya telah disimpan pada SharedPreferences.



Gambar 17 Edit Profile©Dokumentasi Penulis

Jika pengguna berhasil melakukan validasi maka tampilan akan berubah ke EditProfile.class seperti tampilan diatas. Edit Profile terdiri dari layout yang diatur dalam.xml dan pengaturan pada setiap komponen layout yang dilakukan dalam file .java.

Pada Edit Profile, semua masukan dari pengguna sebelumnya yang disimpan dalam database, dan disimpan pada Shared Preference saat proses Sign-In dilakukan, dipanggil. Data tersebut kemudian dipanggil di Edit Profile dan ditampilkan (di-parsing) sesuai dengan layoutnya. Berikut adalah baris kode yang digunakan untuk menampilkan data diri pengguna pada tampilan EditProfile.class:

```
SharedPreferences pref = getSharedPreferences(Data.SPREF_NAME,
MODE_PRIVATE);
nameHeader = pref.getString("Name", "");
emailHeader = pref.getString("Email", "");
username = pref.getString("Username", "");
password = pref.getString("Password", "");
birthdate = pref.getString("Birthdate", "");
nim = pref.getString("NIM", "");
occupation = pref.getString("Occupation", "");
interest = pref.getString("Interest", "");
units = pref.getString("Unit", "");

//parsing
UsernameTxt = (EditText) findViewById(R.id.Username);
UsernameTxt.setEnabled(false);
UsernameTxt.setInputType(InputType.TYPE_NULL);
UsernameTxt.setFocusable(false);
UsernameTxt.setText(username);

PasswordTxt = (EditText) findViewById(R.id.Password);
PasswordTxt.setText(password);
```

```
NameTxt = (EditText) findViewById(R.id.Name);  
NameTxt.setText(nameHeader);
```

Pengguna dapat melakukan perubahan pada setiap kolom inputan, kecuali pada bagian Username. Hal ini dikarenakan Username merupakan id khusus yang dimiliki pengguna dalam database, sehingga tidak dapat diganti. Selanjutnya pengguna dapat menyimpan hasil perubahannya dengan menekan button “SAVE”.

```
SaveBtn.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        //publish  
        JSONObject jsonObject = new JSONObject();  
        try {  
            jsonObject.put("_id", UsernameTxt.getText().toString());  
            jsonObject.put("oldpassword", PasswordTxt.getText().toString());  
            jsonObject.put("password", NewPasswordTxt.getText().toString());  
            jsonObject.put("fullname", NameTxt.getText().toString());  
            jsonObject.put("nim", NIMTxt.getText().toString());  
            jsonObject.put("email", EmailTxt.getText().toString());  
            jsonObject.put("birthdate", BirthdateTxt.getText().toString());  
            jsonObject.put("occupation", OccupationTxt.getText().toString());  
            jsonObject.put("interest",  
multiSelectionSpinner.getSelectedItemsAsString());  
            jsonObject.put("unit",  
unitSelectionSpinner.getSelectedItemsAsString());  
  
            String cacheDir = getCacheDir().getAbsolutePath();  
  
            MqttClient client = null;  
  
            client = new MqttClient("tcp://test.mosquitto.org:1883",  
"ubeacon/user/editprofile", new MqttDefaultFilePersistence(cacheDir));  
  
            client.connect();  
            client.setCallback(EditProfile.this);  
            client.subscribe("ubeacon/user/editprofile/" +  
UsernameTxt.getText().toString());  
            MqttTopic mqttTopic = client.getTopic("ubeacon/user/editprofile");  
            MqttMessage mqttMessage = new  
MqttMessage(jsonObject.toString().getBytes());  
            mqttMessage.setQos(1);  
  
            mqttTopic.publish(mqttMessage);  
  
            catch (MqttException e) {  
                throw new RuntimeException(e);  
            } catch (JSONException e) {  
            }  
        }  
    }  
});
```

Seperti pada pemrosesan yang dilakukan pada Sign-In dan Sign-Up, proses yang sama juga dilakukan pada Edit Profile. Pada bagian ini, input dari pengguna di konversi menjadi JSON file dan selanjutnya dibuat koneksi melalui MQTT via internet pada topic `ubeacon/user/editprofile`. Setelah koneksi berhasil dibuat, JSON file akan di-publish ke alamat topic. Selanjutnya akan diterima balasan dari backend pada topic `ubeacon/user/editprofile/` + `UsernameTxt.getText().toString()` untuk dilanjutkan proses selanjutnya. Bila perubahan berhasil dilakukan, maka akan terjadi pergantian tampilan kembali ke menu utama, dan bila perubahan profile gagal, akan muncul error text pada nama pengguna.

Jika proses perubahan profile berhasil dilakukan, pengguna akan menerima balasan dari backend berupa status keberhasilan dan seluruh data diri terbaru pengguna. Database pengguna ini selanjutnya akan disimpan kembali pada SharedPreferences.

2.2.5.8 Push-Notification sebagai fitur personalisasi pengguna.

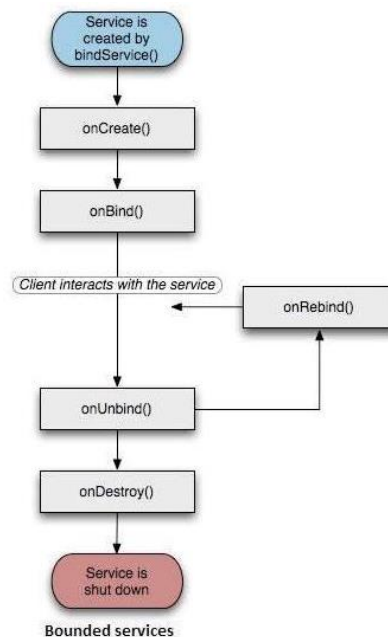
Push notification merupakan fitur tambahan yang dimiliki oleh aplikasi uBeacon. Fitur ini berupa distribusi informasi kepada pengguna tanpa dilakukannya request oleh pengguna. Aplikasi diprogram sebagai subscriber selama aplikasi di-running. Dengan membuat aplikasi berperan sebagai subscriber maka aplikasi siap menerima informasi dari server kapanpun informasi dikirimkan selama ada koneksi internet. Untuk melakukan koneksi dan proses selama aplikasi di-running tanpa mengganggu pengguna dalam menggunakan fitur lain dalam aplikasi maka proses ini dilakukan menggunakan Service sebagai background process.

Service

Pada aplikasi ini, digunakan service yang akan melakukan proses pada Background sehingga tidak akan mengganggu aktivitas pengguna selama menggunakan aplikasi. Service adalah komponen yang beroperasi di background untuk melakukan operasi dalam jangka panjang tanpa memerlukan interaksi dengan pengguna. Service juga akan tetap bekerja meskipun aplikasi dalam state destroy.

State service yang digunakan pada aplikasi ini adalah bound service. Bound service menyediakan interaksi antara client (dalam hal ini Navigation.class) dengan service tersebut. Service akan terikat (bound) dengan aplikasi saat komponen dalam aplikasi memanggil `bindService()`. Bound service akan menyediakan interaksi antara client-service yang memungkinkan terjadinya interaksi dari elemen dan komponen dalam aplikasi untuk berinteraksi dengan service seperti mengirimkan perintah, menerima hasil, dan melakukan proses lain.

Berikut adalah siklus dari Bound Service:



Gambar 18 Flowchart onBoundService © Dokumentasi Penulis

Pada permulaannya akan dilakukan persiapan pada Service sehingga siap untuk digunakan.

```
private final IBinder notifBinder = new NotificationBinder();
```

Baris kode diatas merupakan inisialisasi yang digunakan untuk memulai metode binding antara service dan java class. IBinder digunakan sebagai interface. Metode selanjutnya yang digunakan adalah onBind(). Sistem memanggil metode ini saat ada komponen lain yang ingin di-bind dengan service ketika komponen tersebut memanggil bindService(). Pada implementasinya harus disediakan interface yang digunakan client (Navigation.class) untuk berkomunikasi dengan service dengan mengembalikan sebuah IBinder, seperti baris kode berikut:

```
@Override
public IBinder onBind(Intent intent) {
    return notifBinder;
}
```

Ketika tidak ada komponen atau client yang akan di-bind, maka interface ini harus mengembalikan null.

Untuk melakukan binding juga diperlukan sebuah class yang memiliki kemampuan untuk melakukan bind antara client dan service. Class ini akan mengikat keseluruhan service dan client. Berikut adalah baris komponen yang menyatakan class tersebut:

```
//create class that has an ability to bind the service and client(app)
public class NotificationBinder extends Binder{
    MyServiceNotification getService(){
        return MyServiceNotification.this;
    }
}
```

System service telah berhasil dibuat. Selanjutnya dilakukan inisialisasi pada main activity sehingga service dapat bekerja pada saat aplikasi diluncurkan. Pada Navigation.java dilakukan inisialisasi sebagai berikut:

```
Intent i = new Intent(this, MyServiceNotification.class);
bindService(i, notifConnection, Context.BIND_AUTO_CREATE);
```

Dari baris deskripsi kode diatas, bind service dimulai dengan deskripsi **BIND_AUTO_CREATE**. Pada Navigation.java juga dilakukan pengaturan pada koneksi service yang diatur sebagai berikut:

```
private ServiceConnection notifConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        NotificationBinder binder = (NotificationBinder) service;
        notificationService = binder.getService();
        isBind = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        unbindService(notifConnection);
        isBind = false;
    }
};
```

Sesuai dengan baris kode diatas, saat service sedang melakukan koneksi maka bind service akan dilakukan dan status bind adalah true. Sedangkan bila service

melakukan pemutusan koneksi, maka bind service dengan client akan diputus dan status bind berubah menjadi false.

Setelah kulit sistem berhasil dibuat, selanjutnya akan dilakukan pengaturan pada service. Service pada aplikasi ini dimaksudkan sebagai proses background yang menaungi segala peran aplikasi sebagai subscriber. Aplikasi berperan sebagai subscriber untuk menerima pesan dari background berupa push notification dari unit, NIM, dan advertisement (berlaku ke semua pengguna, tidak diberlakukan personalisasi) dengan ketentuan topic sebagai berikut:

```
unit_client.subscribe("ubeacon/user/notification/unit/" + unit);
major_client.subscribe("ubeacon/user/notification/major/" + nim_notif);
advertise_client.subscribe("ubeacon/user/notification/advertise" );
```

Service, dengan bantuan Asynchronous Task melakukan koneksi ke server melalui MQTT via Internet. Penggunaan Asynchronous Task dimaksudkan untuk mengurangi waktu bagi pengguna untuk menunggu hingga aplikasi berhasil melakukan koneksi ke server. Asynchronous Task membuat aplikasi dapat menjalankan program secara parallel, sehingga sebelum aplikasi berhasil melakukan koneksi di background process, tampilan menu utama atau Sign In sudah bisa ditampilkan kepada pengguna.

Koneksi ke server dituliskan dalam onCreate pada Service. Saat digunakan Asynchronous Task maka baris deskripsi dapat dituliskan sebagai berikut:

```
@Override
public void onCreate() {
    new Connect().execute();
}
```

Fungsi Connect() pada baris deskripsi diatas mengimplementasikan kerja dari metode Asynchronous Task. Fungsi ini bertujuan untuk melakukan koneksi ke topic menggunakan MQTT sehingga uBeacon siap beroperasi sebagai subscriber. Adapun baris kode yang digunakan sebagai berikut:

```
private class Connect extends AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... params) {
        Log.i(TAG, "Service onCreate");
        try {
            String cacheDir = getCacheDir().getAbsolutePath();

            //broker and topic inisialisation
            unit_client = new MqttClient(BROKER_URL, CLIENT_UNIT,
                new MqttDefaultFilePersistence(cacheDir));
            major_client = new MqttClient(BROKER_URL, CLIENT_MAJOR,
                new MqttDefaultFilePersistence(cacheDir));
            advertise_client = new MqttClient(BROKER_URL,
CLIENT_ADVERTISE,
                new MqttDefaultFilePersistence(cacheDir));

            unit_client.setCallback(MyServiceNotification.this);
            major_client.setCallback(MyServiceNotification.this);
            advertise_client.setCallback(MyServiceNotification.this);

            opt = new MqttConnectOptions();
            opt.setKeepAliveInterval(keepAliveInterval);
            opt.setConnectionTimeout(10);

            //connect using Mqtt
            try {
```

```

        unit_client.connect(opt);
        major_client.connect(opt);
        advertise_client.connect(opt);
        Log.i(TAG, "Connect");
    } catch (MqttException e) {
        System.out.print("Connection error!! " + e.toString());
    }
    if (unit_client.isConnected()) {
        //inisialisasi callback method in this fragment
        //subscribe
        unit_client.subscribe("ubeacon/user/notification/unit/"
+ unit);
        Log.i(TAG, "ubeacon/user/notification/unit/" + unit);
    }
    major_client.subscribe("ubeacon/user/notification/major/" + nim_notif);
    advertise_client.subscribe("ubeacon/user/notification/advertise" );
    Log.i(TAG, "ubeacon/user/notification/major/" +
nim_notif);
    Log.i(TAG, "Subscribe");
} else {
    unit_client.connect(opt);
    major_client.connect(opt);
    advertise_client.connect(opt);
    Log.i(TAG, "Connect");
}

} catch (MqttException e) {
    //throw new RuntimeException(e);
}
return null;
}

@Override
protected void onPostExecute(String result) {
    Log.i(TAG, "PostExecute");
}

@Override
protected void onPreExecute() {
}

@Override
public IBinder onBind(Intent intent) {
    return notifBinder;
}
}

```

Implementasi dari Asynchronous Task memiliki tiga fungsi bawaan yaitu:

```

@Override
protected String doInBackground(String... params) {}

@Override
protected void onPostExecute(String result) {}

@Override
protected void onPreExecute() {}

```

Pada fungsi `doInBackground(String... params)` dilakukan koneksi aplikasi ke topic menggunakan MQTT via Internet. Dilakukan koneksi ke tiga topic yang berbeda. Selanjutnya setelah koneksi berhasil dibuat, maka peran dari Asynchronous Task telah berakhir.

Selanjutnya operasi kembali ke Service. Saat koneksi telah berhasil dibuat, implementasi dari MqttCallback dilakukan pada fungsi `messageArrived(String s, MqttMessage mqttMessage)`. Pada fungsi ini akan dilakukan penyimpanan informasi pada `SharedPreferences`. Selanjutnya dibangun sebuah fungsi `Notification` yang memudahkan pengguna untuk mengetahui adanya informasi baru. Berikut adalah baris deskripsi yang menyatakan kerja dari `notification`:

```
Random random = new Random();
int m = random.nextInt(9999 - 1000) + 1000;

manager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

Intent intent = new Intent(this,
com.example.astaripurnomo.ubeacon.Notification.class);
intent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);

PendingIntent pendingIntent = PendingIntent.getActivity(this, 1, intent,
PendingIntent.FLAG_UPDATE_CURRENT);

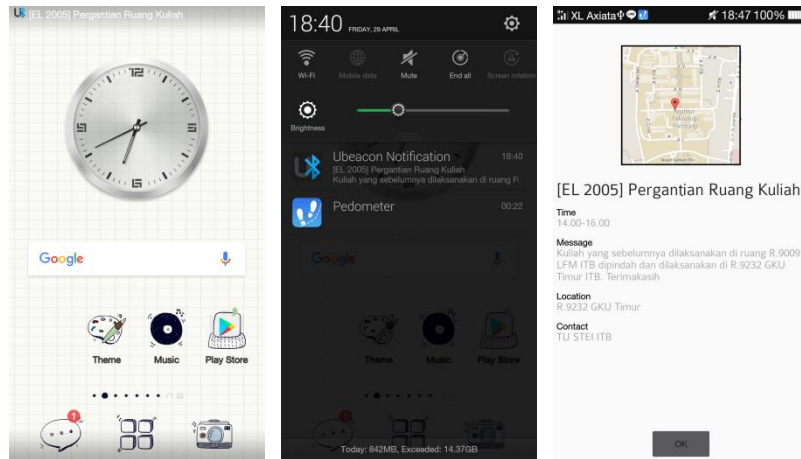
Notification.Builder builder = new Notification.Builder(this);
builder.setAutoCancel(false);
builder.setTicker(title);
builder.setContentTitle("Ubeacon Notification");
builder.setContentText(title);
builder.setSmallIcon(R.drawable.icon);
builder.setContentIntent(pendingIntent);
builder.setOngoing(true);
builder.setAutoCancel(true);
builder.setDefaults(Notification.DEFAULT_SOUND);

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
    builder.setSubText(notifications); //API level 16
}

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
    builder.build();
}

myNotification = builder.getNotification();
manager.notify(m, myNotification);
```

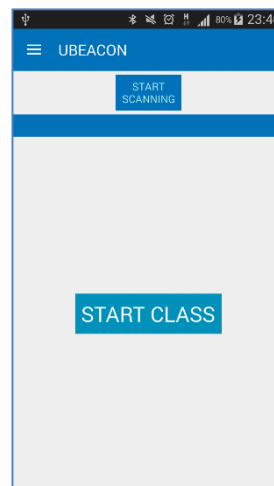
Pada implementasinya digunakan bilangan random supaya notifikasi yang muncul tidak menumpuk dan meniadakan informasi yang sebelumnya telah muncul. Dilakukan pengaturan sehingga pengguna bisa dengan mudah mengetahui adanya informasi baru. Pengaturan yang dilakukan antara lain, dilakukannya pengaturan pada suara, sehingga kapanpun notifikasi muncul, ada ringtone sebagai penanda bagi pengguna. Selanjutnya notifikasi juga dilengkapi dengan autocancel, sehingga saat pengguna menekan notifikasi, icon pada notifikasi akan segera menghilang tepat setelah ditekan. Saat pengguna menekan notifikasi maka akan terlihat tampilan `Notification.class` yang menunjukkan secara lengkap informasi yang dikirimkan oleh server. Berikut adalah tampilan dari push-notification:



Gambar 19 Tampilan Notifikasi ©Dokumentasi Penulis

2.2.5.9 Personalisasi dengan fitur presensi mahasiswa dalam kelas perkuliahan..

Fitur presensi mahasiswa dalam kelas perkuliahan diimplementasikan untuk memudahkan mahasiswa dalam melakukan pendaftaran presensi mahasiswa dalam suatu perkuliahan. Fitur ini dibuat dalam sebuah kelas(class) Java berupa *AttendClassFragment.java*. Berikut tampilan dari fitur Attend Class yang terdapat pada aplikasi.



Gambar 20 Tampilan Attend Class ©Dokumentasi Penulis

Pengguna hanya perlu untuk menekan sebuah button “Start Class” untuk memulai kelas. Berikut potongan kode untuk mengaplikasikan fitur *Attend Class*.

```
public class AttendClassFragment extends Fragment implements MqttCallback
{
    ...
    ...
    //Timer
    static Timer timer;
    static int interval;

    public AttendClassFragment() {
        // Required empty public constructor
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
        container,
        Bundle savedInstanceState) {
```



```

        super.onCreateView(inflater, container, savedInstanceState);
        final View rootView =
inflater.inflate(R.layout.fragment_attend_class,
                                                         container,
false);
        button1 = (Button)rootView.findViewById(R.id.main_button);

        try {
            String cacheDir =
getActivity().getCacheDir().getAbsolutePath();

            client = new MqttClient("tcp://test.mosquitto.org:1883",
"ubeacon/user/presence",
            new MqttDefaultFilePersistence(cacheDir));

            client.connect();
            client.setCallback(AttendClassFragment.this);

        } catch (MqttException e) {
            throw new RuntimeException(e);
        }

        button1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                button1.setClickable(false);
                button1.setText("OK");

                int delay = 2000;

                int period = 120000;
                timer = new Timer();
                interval = Integer.parseInt(count);

                timer.scheduleAtFixedRate(new TimerTask() {
                    @Override
                    public void run() {
                        System.out.println(setInterval());
                        ...
                        ...
                        ...
                        i++;

                        if (i == count2){
                            showNotification("Class is finished", "Your
presence has been successfully marked");
                        }
                    }
                }, delay, period);
            }
        });

        return rootView;
    }
    ...
    ...
    ...
}

```

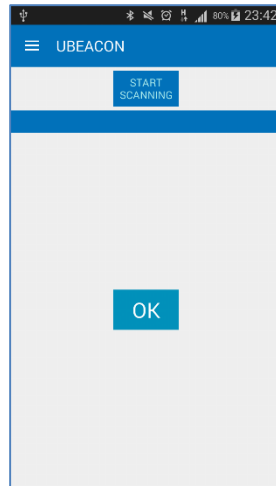
Pada proses ini aplikasi akan menggunakan fitur MQTT untuk menghubungkan data pengguna dengan server. Selanjutnya, ketika pengguna menekan tombol *Start Class*, maka akan terjadi proses dimana pengguna akan mengirimkan payload kepada server selama 2 jam. Adapun pengiriman payload terjadi secara otomatis dengan menggunakan fitur timer dengan interval 5 menit. Hal tersebut dilakukan untuk

memastikan kebenaran bahwa pengguna berada di dalam kelas selama jam pelajaran kuliah.

Perhitungan yang akan terjadi adalah dengan durasi jam pelajaran kuliah dengan asumsi 2 jam (120 menit), jika pengguna terus mengirim payload selamat 5 menit maka akan ada 24 kali pengiriman payload kepada server untuk selanjutnya di proses di server. Ketentuan akan perhitungan kehadiran akan dilakukan pada server IBM Bluemix.

Adapun, setelah pengguna melakukan klik button *Start Class*, maka tampilan akan berubah seperti sebagai berikut.

Gambar 11 Tampilan uBeacon Admin Dashboard bagian notifikasi Advertise. ©Dokumentasi Penulis



Gambar 21 Tampilan hasil akhir attend class ©Dokumentasi Penulis

2.2.6 Permasalahan dan Solusi

Permasalahan yang sedang dialami saat ini adalah dalam pengelolaan informasi yang diterima oleh aplikasi android terhadap beacon. Pengelolaan tampilan ketika *container* listview mencapai kapasitas maksimal nya belum dapat diatasi.

2.3 Implementasi Sub-sistem *Beacon*

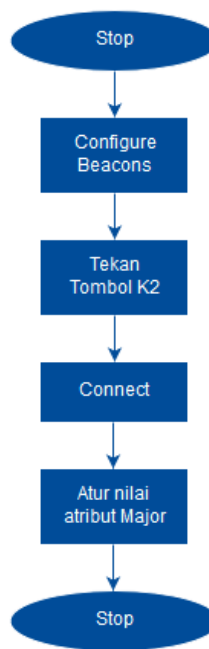
2.3.1 Pendahuluan

Sub-sistem perangkat beacon berperan untuk memancarkan signal Bluetooth low energy(BLE) secara kontinu kepada devais di sekitarnya. Signal tersebut terdiri dari berbagai atribut identifier tertentu antara lain UUID, Major, Minor, RSSI, dan MAC Address. Atribut-atribut tersebut akan digunakan oleh devais yang memiliki *mobile application* Ubeacon untuk mengakses berbagai informasi yang telah diatur pada Backend Application pada sistem Ubeacon. Oleh karena itu, beacon dengan atribut yang berbeda-beda akan menghasilkan tampilan informasi yang berbeda pula. Pada tahap ini akan dilakukan implementasi untuk menguji keluaran signal beacon, jarak(*range*) terjauh dari modul beacon, serta jarak-jarak yang menjadi nilai *threshold* untuk seorang mahasiswa dapat melakukan *Attend Class*.

2.3.2 Struktur Implementasi

Dalam mengimplementasikan sub-sistem perangkat beacon hanya perlu dilakukan konfigurasi perangkat beacon agar setiap perangkat memiliki atribut masing-masing. Oleh karena itu, perlu dilakukan konfigurasi menggunakan sebuah software bernama

Cubeacon Tools App. Berikut tahap yang perlu dilakukan untuk melakukan konfigurasi perangkat Beacon yang digunakan.



Gambar 12 Flowchart konfigurasi beacon. ©Dokumentasi Penulis

2.3.3 Environment

Dalam mengimplementasikan sub-sistem *mobile application* digunakan *software* dan *hardware* sebagai berikut.

1. Perangkat Beacon merk Cubeacon

Perangkat beacon merupakan salah satu perangkat utama dalam sistem ini yang memancarkan signal melalui Bluetooth low energy yang akan diterima oleh devais android yang akan memproses signal tersebut pada aplikasi yang di install pada hp android tersebut. Perangkat Beacon dengan jumlah tertentu akan digunakan, namun dalam pengujiannya akan digunakan 3 buah beacon. Untuk beacon dalam jumlah banyak perlu dilakukan konfigurasi perangkat beacon supaya memiliki ciri yang berbeda-beda.

2. Aplikasi *Cubeacon Tools App* pada devais Android

Aplikasi Cubeacon Tools App memiliki beberapa fungsi utama. Dalam hal ini Cubeacon Tools App digunakan untuk melakukan konfigurasi atribut Major pada perangkat Beacon sehingga beacon akan memiliki atribut unik yang dapat membedakan perangkat yang satu dengan yang lainnya. Atribut tersebut merupakan atribut penentu yang menjadi penentu pemrosesan yang dilakukan oleh mobile application pada perangkat android..

2.3.4 Prosedur Implementasi

Dalam melaksanakan implementasi dari sub-sistem perangkat beacon ini dibutuhkan beberapa langkah prosedur sebagai berikut.

1. Melakukan pemilihan perangkat beacon yang sesuai dengan apple standard iBeacon protocol.
2. Melakukan konfigurasi atribut Major pada beacon jika devais digunakan berjumlah lebih dari 3.

3. Menempatkan beacon pada tempat yang diinginkan.
4. Melakukan pengujian signal dan identifier beacon yang dipancarkan.
5. Melakukan pengukuran jarak terjauh yang dapat dicapai beacon.
6. Melakukan pengukuran jarak *threshold* untuk dapat melakukan *attend class*.

2.3.5 Progres Implementasi

Perangkat beacon yang digunakan adalah beacon dengan merk Cubeacon yang setiap pembeliannya didapatkan 1 paket beacon terdiri dari 3 beacon berwarna merah, hijau, dan biru yang setiap pembeliannya memiliki atribut identifier yang sama. Oleh karena itu, jika dalam sebuah sistem digunakan lebih dari 3 beacon dengan merk Cubeacon, maka perangkat beacon lain harus memiliki nilai atribut identifier yang berbeda sehingga dapat menghasilkan informasi yang berbeda pula.

Pengaturan atribut identifier pada perangkat Beacon merk Cubeacon dilakukan dengan menggunakan aplikasi Cubeacon Tools App. Dalam hal ini, pengaturan atribut identifier pada beacon hanya dilakukan pada atribut major, yang secara umum menggambarkan nomor urutan perangkat beacon yang digunakan. Oleh karena itu, perangkat beacon yang memiliki nomor urut di atas 3 akan dilakukan konfigurasi sehingga mengurut dari nomor 4, 5, 6, dst. menyesuaikan dengan jumlah Beacon yang ada.

Berikut hasil konfigurasi Beacon yang dihasilkan :

No. Beacon	Identifier	
Red Beacon	UUID	cb10023f-a318-3394-4199-a8730c7c1aec
	Major	1
	Minor	284
Green Beacon	UUID	cb10023f-a318-3394-4199-a8730c7c1aec
	Major	2
	Minor	284
Blue Beacon	UUID	cb10023f-a318-3394-4199-a8730c7c1aec
	Major	3
	Minor	284
Red Beacon 2	UUID	cb10023f-a318-3394-4199-a8730c7c1aec
	Major	4
	Minor	284
Green Beacon 2	UUID	cb10023f-a318-3394-4199-a8730c7c1aec
	Major	5
	Minor	284
Blue Beacon 2	UUID	cb10023f-a318-3394-4199-a8730c7c1aec
	Major	6
	Minor	284

Dalam implementasi nya, beacon akan di deteksi dengan menggunakan fitur Monitor Beacon yang telah dijelaskan pada bagian sebelumnya (bagian *mobile application*). Secara garis besar, pengujian dilakukan pada background application dimana ketika Bluetooth smartphone pengguna berada pada keadaan ON dan memasuki daerah(*region*) beacon, maka keberadaan bluetooth beacon akan terdeteksi oleh smartphone dan akan di akses informasinya melalui server.

Selanjutnya, dalam implementasi nya dilakukan pula pengujian terhadap jarak terjauh beacon yang dapat di deteksi. Implementasi nya dilakukan dengan menggunakan aplikasi android, beacon, dan sebuah lokasi *outdoor* untuk meminimalisir pengurangan kekuatan signal(*signal strength*) dari Bluetooth beacon. Untuk dapat mendeteksi keberadaan beacon dapat dilakukan dengan Beacon Monitoring, namun untuk mengetahui signal strength dan jarak nya dapat dilakukan dengan implementasi kode berikut :

```
...  
Data.PROXIMITY_VALUE = beacon.getDistance();  
Data.PROXIMITY_STATUS = getProximityString(beacon.getDistance());  
...
```

Selain itu, dengan menggunakan script kode yang sama akan dilakukan pengukuran terhadap nilai *threshold* pada 4 buah ruang kelas kuliah di ITB. Hal ini dilakukan dengan menggunakan beacon, smartphone dengan aplikasi yang sama, serta alat pengukur jarak(meter) gulung dengan jarak maksimal 50m.

Pengujian akan dilakukan dengan menempatkan posisi beacon pada bagian tengah kelas, kemudian diambil 4 titik sudut terjauh dari kelas dan diukur nilai jarak yang terukur pada aplikasi. Nilai rata-rata yang terbaca akan menjadi nilai terbesar yang akan menentukan posisi seseorang jika sedang berada dalam lingkup ruang kelas dan dapat dihitung hadir dalam sistem kehadiran yang telah di desain.

2.3.6 Permasalahan dan Solusi

Dalam mengimplementasikan sub-sistem perangkat *beacon* terdapat permasalahan sebagai berikut.

1. Beacon sebagai sebuah perangkat elektronik tidak dapat ditempatkan di sembarang tempat karena akan rawan terhadap pencurian. Pada sistem ini, beacon perlu untuk ditempatkan pada suatu wadah yang tidak mencolok perhatian orang yang berlalu-lalang. Oleh karena itu, perlu dibuat suatu wadah atau packaging khusus untuk perangkat Beacon agar tidak diketahui orang akan keberadaannya.

2.4 Lampiran

Bagian ini berisi *source code* dari semua sub-sistem yang telah dibuat. *Source code* terdiri dari *back-end application project* yang di-*deploy* ke *server*, serta *Android Studio project* yang dikompilasi menjadi aplikasi Android.

2.4.1 Lampiran Source Code Back-end Application Project

package.json

```
{
  "name": "uBeacon",
  "version": "0.0.1",
  "description": "uBeacon Back-end Application",
  "scripts": {
    "start": "node app.js"
  },
  "dependencies": {
    "express": "*",
    "body-parser": "*",
    "cfenv": "*",
    "nano": "*",
    "cloudant": "*",
    "mqtt": "*",
    "request": "*"
  },
  "repository": {}
}
```

manifest.yml

```
---
applications:
- disk_quota: 1024M
  host: backend1
  name: backend1
  path: .
  domain: au-syd.mybluemix.net
  instances: 1
  memory: 512M
```

app.js

```
// Express framework
var express = require('express');
var app = express();

// Cloud foundry module for deploying to IBM Bluemix
var cfenv = require('cfenv');
var appEnv = cfenv.getAppEnv();

// MQTT module and connect to broker
var mqtt = require('mqtt');
var client = mqtt.connect('mqtt://test.mosquitto.org');

// Nano module to read database from IBM Cloudant
var nano = require('nano');

// Cloudant module and initiate IBM Cloudant authentication
var cloudant2 = require('cloudant');
var me = appEnv.services.cloudantNoSQLDB[0].credentials.username;
var pass = appEnv.services.cloudantNoSQLDB[0].credentials.password;
var cloudant = cloudant2({account:me, password:pass});

var bodyParser = require('body-parser');
app.use(bodyParser.json());
```

```

app.use(express.static('public'));

var https = require('https');

// POST notification to back-end, then MQTT publish to mobile apps
app.post('/post/notification/unit', function (req, res){
  var notificationData = req.body;
  var unitData = req.body.unit;
  var notification_unit_topic = 'ubeacon/user/notification/unit/' + unitData;
  delete notificationData.unit;
  var doc = JSON.stringify(notificationData);
  console.log(notification_unit_topic);
  client.publish(notification_unit_topic, doc, function(){
    console.log("JSON sent (unit notification)");
  });
});

// POST notification to back-end, then MQTT publish to mobile apps
app.post('/post/notification/major', function (req, res){
  var notificationData = req.body;
  var majorData = req.body.major;
  var batchData = req.body.batch;
  var notification_major_topic = 'ubeacon/user/notification/major/' + majorData +
batchData;
  delete notificationData.major;
  delete notificationData.batch;
  var doc = JSON.stringify(notificationData);
  client.publish(notification_major_topic, doc, function(){
    console.log("JSON sent (major notification)");
  });
});

//POST notification to back-end, then MQTT publish to mobile apps
app.post('/post/advertise', function (req, res){
  var notificationData = req.body;
  var doc = JSON.stringify(notificationData);
  client.publish("ubeacon/user/notification/advertise", doc, function(){
    console.log("JSON sent (advertise)");
  });
});

// GET attendance data from back-end to be displayed on admin page
app.get('/presence/EL2001-01', function (req, res){
  res.json(att_EL2001_01);
  console.log("GET /presence/EL2001-01");
});

app.get('/presence/EL2002-01', function (req, res){
  res.json(att_EL2002_01);
  console.log("GET /presence/EL2002-01");
});

app.get('/presence/EL2003-01', function (req, res){
  res.json(att_EL2003_01);
  console.log("GET /presence/EL2003-01");
});

app.get('/presence/EL4091', function (req, res){
  res.json(att_EL4091);
  console.log("GET /presence/EL4091");
});

// Function to forward beacon data request to IBM Presence Insight
function ToPresenceInsight_BeaconConnector(jsoninput_beaconconnector){
  var options = {
    hostname: 'presenceinsights.au-syd.bluemix.net',
    port: 443,
    path: '/conn-beacon/v1/tenants/1g4a05cq/orgs/mylu08tp',
    method: 'POST',
    headers: {

```

```

        'Content-Type': 'application/json',
        'Authorization': 'Basic NTJnMDBBrbTpiNXhQdzZKLW5USnE='
    }
};
var req = https.request(options, function (res){
    console.log("Status: "+res.statusCode);
    //console.log("Headers: \n", res.headers);
    res.on('data', function (data){
        var data2 = data.toString();
        //console.log("Response:", data2);
    });
});
req.write(jsoninput_beaconconnector);
req.end();
req.on('error', function(e){
    console.log("Error:", e);
});
}

// Function to forward registration data to IBM Presence Insight
function ToPresenceInsight_RegisterDevices(jsoninput_registerdevices) {
    var name = jsoninput_registerdevices.fullname;
    var descriptor = jsoninput_registerdevices._id;
    var registered = true;
    var registrationType = "Auto";
    var email = jsoninput_registerdevices.email;
    var birthdate = jsoninput_registerdevices.birthdate;
    var occupation = jsoninput_registerdevices.occupation;
    var interest = jsoninput_registerdevices.interest;
    var jsonoutput =
('{"name":"'+name+'","descriptor":"'+descriptor+'","registered":'+true+',"registrationType":"'+registrationType+'","unencryptedData":{"email":"'+email+'","birthdate":"'+birthdate+'","occupation":"'+occupation+'","interest":"'+interest+'"}}');
    var options = {
        hostname: 'presenceinsights.au-syd.bluemix.net',
        port: 443,
        path: '/pi-config/v1/tenants/1g4a05cq/orgs/mylu08tp/devices',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Basic NTJnMDBBrbTpiNXhQdzZKLW5USnE='
        }
    };
};
var req = https.request(options, function (res){
    console.log("Status: "+res.statusCode);
    //console.log("Headers: \n", res.headers);
    res.on('data', function (data){
        var data3 = data.toString();
        //console.log("Response:", data3);
    });
});
req.write(jsonoutput);
req.end();
req.on('error', function(e){
    console.log("Error:", e);
});
}

// Initiate connection with MQTT broker
client.on('connect', function() {
    // Subscribe to necessary topics
    client.subscribe('ubeacon/user/request');
    client.subscribe('ubeacon/user/signup');
    client.subscribe('ubeacon/user/signin');
    client.subscribe('ubeacon/user/editprofile');
    client.subscribe('ubeacon/user/presence');
});

// Handles arriving messages
client.on('message', function(topic, message, packet) {

```



```

// Switch between topics
switch (topic){

    // Topic for beacon data request
    case 'ubeacon/user/request':
        // Parse incoming message
        var request_1 = message.toString();
        var request_2 = JSON.parse(request_1);
        // Extract UUID, major, minor
        var proximityUUID = request_2.bnm[0].data.proximityUUID;
        var majorUUID = request_2.bnm[0].data.major;
        var minorUUID = request_2.bnm[0].data.minor;
        var descriptor = request_2.bnm[0].descriptor;
        var UUID = (proximityUUID + '-' + majorUUID + '-' + minorUUID);
        // Read 'beacon' database
        var db_beacon = cloudant.db.use('beacon');
        var user_topic_response = "ubeacon/user/response/" + descriptor;
        db_beacon.get(UUID, function(err, req, res){
            delete req._rev;
            delete req._id;
            req.descriptor = descriptor;
            var doc = JSON.stringify(req);
            // Send corresponding beacon database
            client.publish(user_topic_response, doc, function(){
                console.log("JSON sent (request)");
            });
        });
        ToPresenceInsight_BeaconConnector(request_1);
        break;

    // Topic for user data sign-up
    case 'ubeacon/user/signup':
        // Parse incoming message
        var signup_1 = message.toString();
        var signup_2 = JSON.parse(signup_1);
        var signup_3 = signup_2._id;
        // Read 'user' database
        var db_user = cloudant.db.use('user');
        var user_topic_signup = "ubeacon/user/signup/" + signup_3;
        db_user.get(signup_3, function(err, req, res){
            // Conditional: If username not used, registration success, otherwise failed
            if (!req) {
                db_user.insert(signup_2, function(err, req, res) {
                    console.log("Registration success");
                });
                var response = "{\"status\":\"Registration success\"}";
                client.publish(user_topic_signup, response, function(){
                    console.log("Response sent");
                });
            } else {
                console.log("Username taken");
                var response = "{\"status\":\"Username taken\"}";
                client.publish(user_topic_signup, response, function(){
                    console.log("Response sent");
                });
            }
        });
        ToPresenceInsight_RegisterDevices(signup_2);
        break;

    // Topic for user data sign-in
    case 'ubeacon/user/signin':
        // Parse incoming message
        var signin_1 = message.toString();
        var signin_2 = JSON.parse(signin_1);
        var signin_3 = signin_2._id;
        var signin_4 = signin_2.password;
        // Read 'user' database
        var db_user = cloudant.db.use('user');

```

```

var user_topic_signin = "ubeacon/user/signin/" + signin_3;
db_user.get(signin_3, function(err, req, res){
    // Conditional: If username and password match success, otherwise failed
    if (req) {
        if (req.password === signin_4) {
            console.log("Sign-in success");
            var test = req;
            test.status = 'Sign-in success';
            var response = JSON.stringify(test);
            client.publish(user_topic_signin, response, function(){
                console.log("Response sent");
            });
        } else {
            console.log("Wrong password");
            var response = "{\"status\":\"Wrong password\"}";
            client.publish(user_topic_signin, response, function(){
                console.log("Response sent");
            });
        }
    } else {
        console.log("Username not exist");
        var response = "{\"status\":\"Username not exist\"}";
        client.publish(user_topic_signin, response, function(){
            console.log("Response sent");
        });
    }
});
break;

// Topic for edit user data
case 'ubeacon/user/editprofile':
    // Parse incoming message
    var edit_1 = message.toString();
    var edit_2 = JSON.parse(edit_1);
    var edit_3 = edit_2._id;
    var edit_4 = edit_2.oldpassword;
    // Read 'user' database
    var db_user = cloudant.db.use('user');
    var user_topic_edit = "ubeacon/user/editprofile/" + edit_3;
    db_user.get(edit_3, function(err, req, res){
        // Conditional: If username and password match success, otherwise failed.
        Password unchanged if new password value is null;
        if (req) {
            if (req.password === edit_4) {
                if (edit_2.password == "") {
                    edit_2.password = edit_2.oldpassword;
                }
                delete edit_2.oldpassword;
                edit_2._rev = req._rev;
                db_user.insert(edit_2, function(err, req, res) {
                    console.log("Data successfully changed");
                });
                edit_2.status = "Data successfully changed";
                var doc = JSON.stringify(edit_2);
                client.publish(user_topic_edit, doc, function(){
                    console.log("Response sent");
                });
            } else {
                console.log("Authetctication Failed");
                var response = "{\"status\":\"Wrong password\"}";
                client.publish(user_topic_edit, response, function(){
                    console.log("Response sent");
                });
            }
        } else {
            console.log("Username not exist");
            var response = "{\"status\":\"Username not exist\"}";
            client.publish(user_topic_edit, response, function(){
                console.log("Response sent");
            });
        }
    });
});

```

```

    }
  });
  break;

// Topic for attendance data
case 'ubeacon/user/presence':
  // Parse incoming message
  var presence_1 = message.toString();
  var presence_2 = JSON.parse(presence_1);
  var descriptor = presence_2.bnm[0].descriptor;
  var detectedTime = presence_2.bnm[0].detectedTime;
  var proximityUUID = presence_2.bnm[0].data.proximityUUID;
  var majorUUID = presence_2.bnm[0].data.major;
  var minorUUID = presence_2.bnm[0].data.minor;
  var UUID = (proximityUUID + '-' + majorUUID + '-' + minorUUID);
  var proximity = presence_2.bnm[0].data.proximity;
  // Read all database
  var db_courses = cloudant.db.use('courses');
  var db_user = cloudant.db.use('user');
  var db_beacon = cloudant.db.use('beacon');
  // Function to check class attendance name, to be used with findIndex method
  function checkName(object) {
    return object.name === descriptor;
  }
  function checkId(object) {
    return object._id === descriptor;
  }
  if(majorUUID==99){
    break;
  }
  db_beacon.get(UUID, function(err, req, res){
    // Get course name
    var course = req.course;
    // Proximity check
    if (proximity === 'Near' || proximity === 'Immediate'){
      db_courses.get(course, function(err, req, res){
        // Switch by course
        switch(course) {

          case 'EL2001-01':
            var temp1 = req.students;
            var result = temp1.findIndex(checkName);
            if(result>=0){
              var temp2 = req.students[result];
              temp2.timestamp = detectedTime;
              var result2 = att_EL2001_01.attendance.findIndex(checkName);
              if(result2>=0){
                att_EL2001_01.attendance[result2] = temp2;
              } else {
                att_EL2001_01.attendance.push(temp2);
              }
              att_EL2001_01.attendance.sort(compareNim);
              console.log(att_EL2001_01);
            }
            if(descriptor===req.lecturer){
              att_EL2001_01.isLecturerPresent = true;
            }
            break;

          case 'EL2002-01':
            var temp1 = req.students;
            var result = temp1.findIndex(checkName);
            if(result>=0){
              var temp2 = req.students[result];
              temp2.timestamp = detectedTime;
              var result2 = att_EL2002_01.attendance.findIndex(checkName);
              if(result2>=0){
                att_EL2002_01.attendance[result2] = temp2;
              } else {
                att_EL2002_01.attendance.push(temp2);
              }
            }
          }
        }
      });
    }
  });
}

```

```

    }
    att_EL2002_01.attendance.sort(compareNim);
    console.log(att_EL2002_01);
  }
  if(descriptor===req.lecturer){
    att_EL2002_01.isLecturerPresent = true;
  }
  break;

case 'EL2003-01':
  var temp1 = req.students;
  var result = temp1.findIndex(checkName);
  if(result>=0){
    var temp2 = req.students[result];
    temp2.timestamp = detectedTime;
    var result2 = att_EL2003_01.attendance.findIndex(checkName);
    if(result2>=0){
      att_EL2003_01.attendance[result2] = temp2;
    } else {
      att_EL2003_01.attendance.push(temp2);
    }
    att_EL2003_01.attendance.sort(compareNim);
    console.log(att_EL2003_01);
  }
  if(descriptor===req.lecturer){
    att_EL2003_01.isLecturerPresent = true;
  }
  break;

case 'EL4091':
  var temp1 = req.students;
  var result = temp1.findIndex(checkId);
  if(result>=0){
    var temp2 = req.students[result];
    temp2.timestamp = detectedTime;
    var result2 = att_EL4091.attendance.findIndex(checkId);
    if(result2>=0){
      att_EL4091.attendance[result2] = temp2;
    } else {
      att_EL4091.attendance.push(temp2);
    }
    att_EL4091.attendance.sort(compareNim);
    console.log(att_EL4091);
  }
  if(descriptor===req.lecturer){
    att_EL4091.isLecturerPresent = true;
  }
  break;
}

});
}
});
break;
}
});

// Attendance data array
var att_EL2001_01 = {"course":"EL2001-01","coursename":"Rangkaian
Elektrik","isLecturerPresent":false,"attendance":[]};
var att_EL2002_01 = {"course":"EL2002-01","coursename":"Sistem
Digital","isLecturerPresent":false,"attendance":[]};
var att_EL2003_01 = {"course":"EL2003-01","coursename":"Struktur
Diskrit","isLecturerPresent":false,"attendance":[]};
var att_EL4091 =
{"course":"EL4091","coursename":"Cavemen","isLecturerPresent":false,"attendance":[]};

var threshold = 300000;
var routine = 5000;

```

```

function compareNim(a,b){
    return a.nim - b.nim;
}

function checkTime(object){
    var d = new Date();
    var currenttime = d.getTime();
    var differencetime = currenttime-object.timestamp;
    return differencetime >= threshold;
}

Array.prototype.remove = function(from, to) {
    var rest = this.slice((to || from) + 1 || this.length);
    this.length = from < 0 ? this.length + from : from;
    return this.push.apply(this, rest);
};

// Routine check for attendance, delete attendance data if more than 5 minutes of absence
setInterval(function() {
    console.log("5 seconds check");
    var d = new Date();
    var currenttime = d.getTime();
    console.log(currenttime);
    var att2_EL2001_01 = att_EL2001_01.attendance;
    var att2_EL2002_01 = att_EL2002_01.attendance;
    var att2_EL2003_01 = att_EL2003_01.attendance;
    var att2_EL4091 = att_EL4091.attendance;
    console.log("EL2001:", att2_EL2001_01);
    console.log("EL2002:", att2_EL2002_01);
    console.log("EL2003:", att2_EL2003_01);
    console.log("EL4091:", att2_EL4091);
    var result3 = att2_EL2001_01.findIndex(checkTime);
    var result4 = att2_EL2002_01.findIndex(checkTime);
    var result5 = att2_EL2003_01.findIndex(checkTime);
    var result6 = att2_EL4091.findIndex(checkTime);
    if(result3>=0){
        att_EL2001_01.attendance.remove(result3);
        result3 = -1;
    }
    if(result4>=0){
        att_EL2002_01.attendance.remove(result4);
        result4 = -1;
    }
    if(result5>=0){
        att_EL2003_01.attendance.remove(result5);
        result5 = -1;
    }
    if(result6>=0){
        att2_EL4091.attendance.remove(result6);
        result6 = -1;
    }
},routine);

// Listen port and binding with IBM Bluemix Environment
app.listen(appEnv.port, appEnv.bind, function(){
    console.log('App listening on port ' + appEnv.port);
});

```