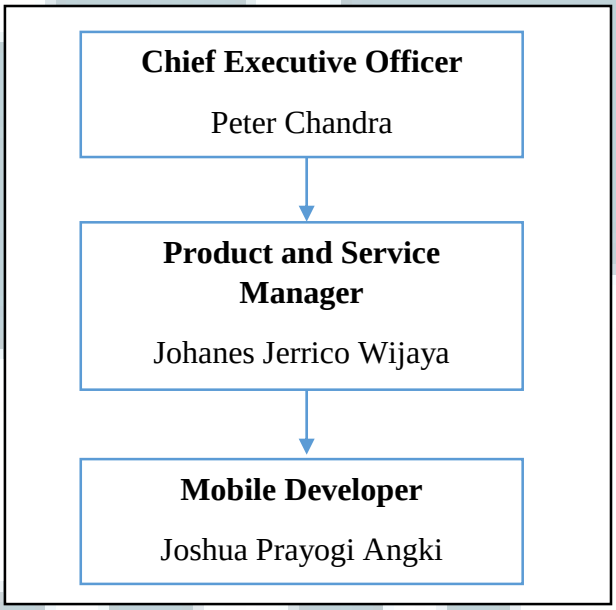
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama bekerja di Moonlay Technologies, penulis bersama dengan para peserta magang lainnya mengerjakan sebuah proyek bernama Loyalti Express. Penulis ditugaskan sebagai *Web Designer* pada dua minggu pertama, kemudian dialihkan menjadi *Mobile Programmer* pada divisi yang sama, yaitu *Product and Service* sampai kegiatan magang berakhir. Kerja magang dikoordinasi dan diawasi secara langsung oleh Bapak Johanes Jerrico Wijaya selaku *senior programmer* dan manajer *Product and Service*.



Bagan 3.1 Koordinasi Kerja Magang

Secara keseluruhan, sistem Loyalti Express ini merupakan perpaduan antara *web* dan *mobile application* yang nantinya akan diluncurkan sebagai produk unggulan dari Moonlay Technologies dan bersifat komersial untuk dijual ke pihak luar. Untuk pengembangan *web application* dilakukan oleh tim yang lainnya. Penulis hanya berfokus pada *mobile application* yang sudah menjadi tanggung jawab penulis. Adapun pengembangan *mobile application* dari sistem ini dilaksanakan oleh tiga orang

programmers yaitu Adrian Armet dan Bernadinus Realino Edi Gunawan yang bertanggung jawab terhadap *layouting* dan *flow program*, dan saya sendiri Joshua Prayogi Angki yang bertanggung jawab terhadap pembuatan *model*, *controller*, *module* dan *library* dari aplikasi tersebut.

3.2 Tugas yang Dilakukan

Selama periode kerja magang, dua minggu pertama penulis ditugaskan sebagai *web programmer*. Penulis melakukan *layouting web page* dan *email page* pada sistem Loyalti Express ini. Penulis tidak akan membahas terlalu banyak mengenai tugas sebagai *web programmer*. Setelah itu, penulis dialihkan sebagai *mobile programmer* yang bertanggung jawab dengan modul internal dari *mobile application* dari sistem Loyalti Express ini. Beberapa modul yang menjadi tanggung jawab penulis adalah *Dynamic Promo with Widget*, *Autoscroll and Paging Control*, *JSON to Backbone JS Mapper*, dan *Dynamic modelling and controller.*

Pada *Dynamic Promo with Widget*, penulis bertugas mengatur isi dari konten yang terdapat dalam setiap *layout* agar dapat dimodifikasi secara dinamis. Nantinya konten ini juga akan dipadukan dengan *local database* untuk *styling* dan *modelling layout* dari sistem Loyalti Express ini. Prinsip *mobile application* dari sistem Loyalti Express ini adalah penjualan *voucher* promo dari berbagai produk yang dipasarkan. Karena *voucher* akan terus menerus berganti secara berkala, maka penulis ditugaskan untuk membuat mekanisme *content builder* secara dinamis menggunakan *widget.*

Widget ini berfungsi untuk membuat sebuah *content view* yang bersifat independen dan dapat di *inject* secara langsung ke aplikasi android yang membutuhkan fitur tersebut. Dengan begitu, *programmer* yang bertanggung jawab terhadap *layouting* dan *content* hanya tinggal menggunakan *content widget* yang sudah dibuat dan melakukan modifikasi sesuai dengan spesifikasi yang diinginkan. Mekanisme untuk melakukan modifikasi juga cukup sederhana dengan pertimbangan bahwa pengguna modul *widget* yang penulis buat belum tentu paham dengan keseluruhan flow dalam penambahan *content widget*. Penambahkan dokumentasi dan tutorial penggunaan widget tersebut juga dilakukan agar tidak membingungkan *programmer* yang lainnya.

Selain menghemat waktu karena *layout programmer* hanya tinggal memasukkan *widget* ketika dibutuhkan konten baru, pembuatan modul ini juga dilakukan guna meningkatkan efisiensi kerja dari pengembangan konten. Selain itu, *widget* yang dibuat juga menjadi *reusable* karena dapat digunakan berulang – ulang untuk setiap konten yang sama ataupun memiliki beberapa kriteria kemiripan baik dalam satu aplikasi maupun aplikasi yang berbeda dan masih berbasis android.

Autoscroll and Paging Control merupakan dua jenis fitur yang berbeda, tapi digunakan secara bersamaan dalam sistem ini. Kedua fitur ini digunakan ketika terjadinya *idle* dalam *flow* program ketika sedang berjalan. Penulis membuat mekanisme berupa pergerakan otomatis secara beraturan dari sekumpulan gambar *slide show*. Bentuk dari autoscroll ini adalah berupa *ImageView*. *ImageView* sendiri di dalam aplikasi Android digunakan untuk menampilkan gambar pada aplikasi. Pada modul ini, penulis menggunakan fitur *ImageView* yang dipadukan dengan fitur *timer* untuk menentukan apakah aplikasi sedang *idle* atau tidak, *automatic scroll* untuk perpindahan halaman dan *page synchronization* untuk mengatur posisi *page* dan gambar yang sedang berjalan.

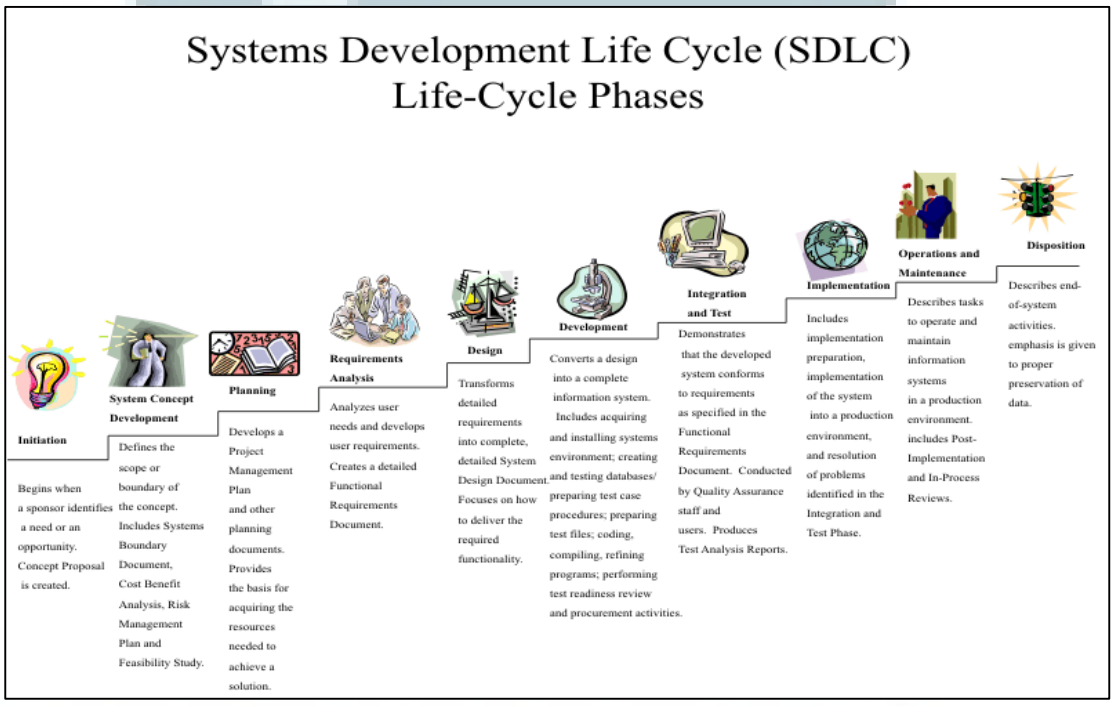
Prinsip kerjanya adalah ketika *user* menjalankan aplikasi, maka *user* dapat menggunakan fitur – fitur yang terdapat di dalam aplikasi tersebut. Namun, ada suatu saat dimana aplikasi akan dibiarkan berhenti atau berada dalam status *idle*. Seperti layaknya *screen saver* pada computer yang biasa kita gunakan, *autoscroll* ini juga mempunyai fungsi dan kegunaan yang sama. Ketika dalam berada posisi *idle*, maka program akan secara otomatis menjalankan fitur *Autoscroll and Paging Control* yang dibuat oleh penulis untuk menampilkan beberapa *slide show* beserta dengan *page control* untuk memberi panduan keberadaan posisi gambar berdasarkan *page view* pada program.

Fitur *Autoscroll and Paging Control* ini juga dapat digunakan secara dinamis, dalam artian konten gambar ataupun sesuatu yang akan ditampilkan dapat diatur dan dimodifikasi sesuai dengan keinginan *programmer*. Mekanisme modifikasi konten gambar juga tidak terlalu sulit. Gambar disimpan dalam *local database* dan hanya tinggal mengirim parameter yang dibutuhkan untuk memanggil. Sistem *callback* untuk

konten juga sudah dibuat secara sistematis pada modul ini oleh penulis untuk memudahkan *programmer* dalam pengaksesannya.

JSON to Backbone JS Mapper dan *Dynamic Modelling and Controller* adalah dua *assignment*s yang penulis akan fokuskan dalam laporan ini. *JSON to Backbone JS Mapper* bersama dengan *model* dan *controller* pada aplikasi ini bersifat saling melengkapi dan bekerja secara sinergis untuk mendapatkan *output* yang akan dihasilkan ataupun diproses di dalam aplikasi ini.

Penulis ditugaskan untuk membuat sebuah *tool* yang nantinya dapat digunakan untuk *data parsing* dan *mapping* dalam pembuatan aplikasi Loyalti Express ini. Pada dasarnya, pembuatan *tool* bisa dikategorikan sebagai pengembangan dari detail metode *System Development Life Cycle* dalam implementasinya. Oleh karena itu, untuk mengeneralisasi pekerjaan yang dilakukan selama pengembangan *tool* ini, penulis akan mengacu pada fase – fase dalam System *Development Life Cycle* seperti yang dirumuskan pada gambar 3.2 berikut ini.



Gambar 3.1 *System Development Life Cycle – Life Cycle Phases*

Sumber : The Department of Justice (2003)

Sistem dari *JSON to Backbone JS Mapper* ini akan dikembangkan sebagai sebuah *tool* yang nantinya akan diimplementasikan pada aplikasi Loyalti Express yang berbasis Android menggunakan Titanium Studio. Pada tiga fase pertama, yaitu *initiation, system concept development,* dan *planning* dimulai oleh penulis dengan berkoordinasi secara langsung dengan *project manager* yaitu Bapak Johanes Jerrico Wijaya. Selama persiapan awal ini, penulis tidak dapat mengambil peran terlalu banyak dikarenakan perencanaan dan perancangan awal dari sistem ini tidak dipublikasikan dan disosialisasikan oleh perusahaan guna meminimalisir keluarnya informasi yang bersifat internal. Maka dari itu, penulis hanya difokuskan dalam pembuatan sistem dari hasil perencaaan awal yang sudah ada.

Setelah persiapan awal selesai, penulis diberi tanggung jawab penuh untuk melanjutkan fase – fase dalam pembuatan sistem *JSON to Backbone JS Mapper* ini. *Timetable* pengerjaan *JSON to Backbone JS Mapper* ini berdasarkan fase – fase tersebut dapat dilihat pada table 3.1 berikut ini.

Tabel 3.1 *Timetable* Pengerjaan *JSON to Backbone JS Mapper* pada Loyalti Express

No.	**Kegiatan**	**Agustus**				**September**	
		1	**2**	**3**	**4**	**1**	**2**
1	Studi literatur						
2	*Requirement analysis*						
3	*Design*						
4	*Development*						
5	*Integration and test*						
6	*Implementation*						
7	*Operations and maintenance*						
8	Penulisan dokumentasi dan laporan magang						

3.2.1 Requirement Analysis

Sebelum masuk ke analisis pembuatan sistem, perlu diketahui bahwa Appcelerator Titanium Studio, Alloy *framework* menggunakan Backbone JS dalam implementasinya. Backbone adalah sebuah *framework* Javascript yang menggunakan *design-pattern Model, View,* dan *Controller*. Dalam Backbone JS sendiri sudah ada mekanisme dalam pengambilan data, tapi hanya berlaku pada *model* dan *collection,* atau dapat dikategorikan sebagai *local database.* Berdasarkan analisis yang ada, Backbone JS memiliki beberapa kekurangan terkait integrasi dengan data, antara lain :

1. Bentuk model yang sulit untuk dimanipulasi. Dalam pengembangan ini, diperlukan sebuah *tool* yang berhubungan dengan *data injection* dan model secara dinamis, sedangkan bentuk model tidak dapat disesuaikan setiap kali ada data berbeda yang diproses.
2. Bentuk data berupa JSON. Seperti yang diketahui, bentuk struktur JSON untuk setiap data kemungkinan besar berbeda-beda satu sama lain. Hal ini akan menimbulkan kesulitan membangun hubungan *eksternal data* dan *internal model* dari aplikasi yang akan dikembangkan.

Maka dari itu, dibuatlah sebuah *tool* yang digunakan *untuk mapping external data* ke dalam *internal model*, yaitu *JSON to Backbone JS Mapper* ini. Penulis melakukan analisis terhadap kebutuhan dari sistem *JSON to Backbone JS Mapper tool* berdasarkan tiga permasalahan, yaitu kebutuhan *programmer* yang akan menggunakan *tool,* kriteria data yang di input ke dalam *tool,* dan bagaimana mekanisme kerja dari *tool* ini.

Berdasarkan hasil analisis kriteria yang sudah dilakukan tersebut, penulis mendapatkan beberapa fitur yang dibutuhkan dalam *tool* ini. Perinciannya adalah sebagai berikut :

1. Data JSON yang kita miliki harus bisa diintegrasikan secara dinamis pada Backbone JS. Backbone JS sebenarnya sudah memiliki mekanisme internal sendiri dalam *data parsing,* sehingga yang menjadi fokus adalah

mendatangkan data eksternal untuk dapat digunakan dan digabungkan dengan fungsi internal dari Backbone JS ini.

2. Data yang akan diproses oleh *tool* harus bisa digunakan untuk beberapa API yang berhubungan dengan aplikasi. Beberapa API yang digunakan pada aplikasi adalah facebook API dan twitter API. Data yang didapatkan dari API adalah berupa user data untuk keperluan login pada aplikasi. *Tool* ini dapat berhubungan dengan berbagai macam API lainnya dengan catatan data yang di input adalah berupa JSON.
3. Diperlukan *mapping* dalam pembacaan data JSON yang ada. Untuk menghindari duplikasi dari pembacaan data, maka diperlukan mekanisme *mapping* terhadap *path* yang ada dalam setiap JSON. *Programmer* hanya tinggal mengatur kriteria mapping yang diinginkan.
4. Mekanisme untuk mengambil data dari *server* dengan *HTTP Client* dan API yang digunakan kemudian harus dapat dihubungkan dengan *model* dan *controller* yang ada dalam aplikasi tersebut secara fleksibel.
5. Enkapsulasi data – data yang akan dihasilkan berdasarkan *mapping* untuk digunakan oleh *programmer.* Data – data yang sudah diproses harus dalam bentuk siap untuk di *inject* ke dalam aplikasi.
6. Teknik *parsing* yang efisien untuk menghemat waktu dalam *mapping* data yang dibutuhkan. Programmer tidak perlu mengetahui seluk beluk dari isi *tool* dan mereka hanya tinggal menggunakannya saja.
7. Mekanisme pengiriman parameter yang dibutuhkan oleh programmer untuk mengembalikan data yang diinginkan.
8. Ada mekanisme untuk dapat melakukan penyimpanan data ke dalam *local database* pada aplikasi. Penyimpanan ini dapat menggunakan model sudah terdapat dalam Backbone JS.

3.2.2 Design

Pada tahap perancangan, dilakukan pembuatan dan analisis algoritma yang dibutuhkan untuk *JSON to Backbone JS Mapper* dan mekanisme untuk mengintegrasikan *tool* dengan aplikasi yang menggunakannya. Sebagai awal, penulis membuat mekanisme penerimaan data ke dalam *JSON to Backbone JS Mapper*. Untuk penerimaan data dari API, bisa langsung digabungkan dengan *JSON to Backbone JS Mapper* ini untuk dihubungkan ke dalam aplikasi. Namun, untuk penerimaan data dari *database server*, perlu ditambahkan fitur *HTTP Client Request* untuk dapat diintegrasikan dengan *server* dari aplikasi yang bersangkutan. Secara umum, fungsi dari *tool* ini memang di rancang untuk dapat mengambil data dari segala sumber yang dapat berhubungan dengan aplikasi dengan catatan bentuk data yang akan di *parse* adalah JSON.

Karena struktur dan bentuk JSON tidak semuanya sama, maka *tool* ini juga di rancang sedemikian rupa sehingga dapat diintegrasikan dengan aplikasi secara mudah dan akurat. Untuk mekanisme *parsing* JSON, penulis menggunakan metode rekursif untuk pembacaan semua data yang ada. Dengan menggunakan rekursif ini, diharapkan pembacaan data menjadi lebih tepat dan akurat dibandingkan dengan *data scan* berdasarkan urutan struktur JSON karena sekali lagi, struktur JSON yang akan di *parsing* belum tentu sama.

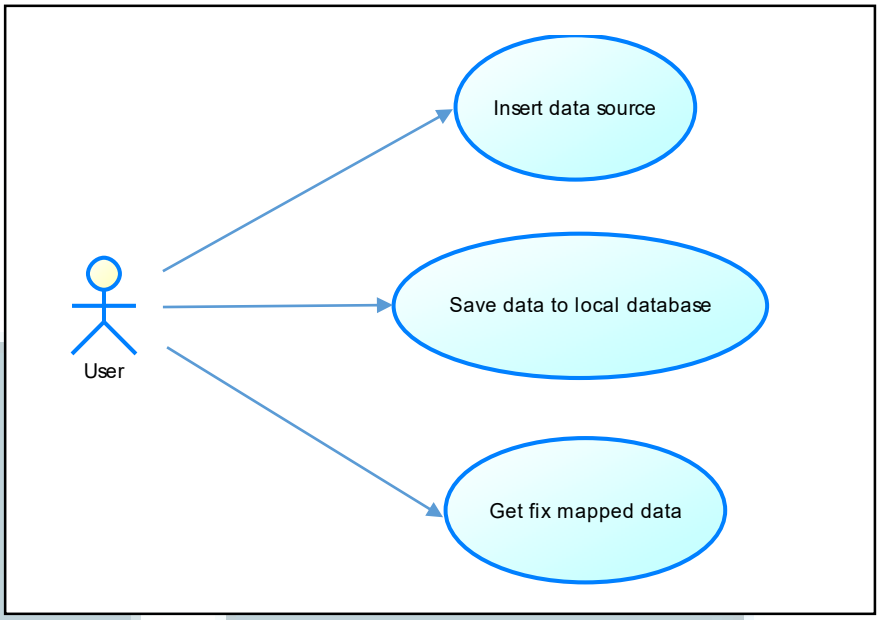
Berbagai mekanisme cara dan metode sudah digunakan untuk dapat menyelaraskan struktur JSON yang didapat, mulai dari *push and pop* untuk mengatur data yang diambil dan dikeluarkan, sampai menggunakan *dynamic modelling* pada aplikasi. Berdasarkan pertimbangan dan analisis selama percobaan awal, *push and pop* dinilai tidak efisien karena memakan waktu yang cukup lama dan memberatkan aplikasi untuk melakukan *parsing* jika data yang diambil tergolong banyak. Untuk *dynamic modelling* juga dinilai kurang efektif karena *programmer* masih harus berhubungan dengan model, dalam artian untuk setiap struktur data yang akan digunakan dalam aplikasi, *programmer* juga harus membuat model yang baru. Terlalu banyak model dalam aplikasi juga dinilai tidak efisien berdasarkan percobaan.

Pada akhirnya diperoleh solusi yang mendapatkan respon cukup baik dari *project manager*, yaitu menggunakan metode *mapping* untuk hubungan data dan *modelling* pada aplikasi. Prinsip kerja dari *mapping* ini cukup sederhana, namun bisa menghasilkan data yang akurat. *Programmer* yang akan menggunakan hanya tinggal membuat *mapping variable* dari JSON di dalam *application model*, kemudian *tool* akan langsung menghubungkan model sesuai dengan struktur *mapping* yang dibuat oleh *programmer*. Cukup efisien, sederhana dan dapat diintegrasikan ke berbagai jenis aplikasi yang membutuhkan. Untuk *output*, *programmer* hanya tinggal menggunakan model yang sudah terhubungan dengan *tool* dalam *controller*.

Untuk membuat suatu rancangan sistem, tentunya dibutuhkan beberapa gambaran dan rancangan teknis tentang alur kerja dari sistem yang akan dibuat. Tidak ada database yang digunakan selama proses pengembangan ini. Maka dari itu, penulis menggunakan *dummy data* selama proses *testing* Dalam pembuatan *JSON to Backbone JS Mapper* ini, digunakan *Unified Modelling Language* (UML) untuk menggambarkan *detail* sistem secara menyeluruh. UML adalah bahasa visual yang digunakan untuk menjelaskan, memberikan spesifikasi, merancang, membuat model, dan mendokumentasikan aspek-aspek dari sebuah sistem yang dibangun. Pembuatan UML ini bertujuan juga untuk membuat patokan dalam pengembangan sistem.

Berikut ini adalah beberapa gambaran sistem yang dirancang dalam bentuk diagram yang nantinya akan di implementasikan secara langsung menjadi sebuah sistem yang utuh.

a. Use Case Diagram

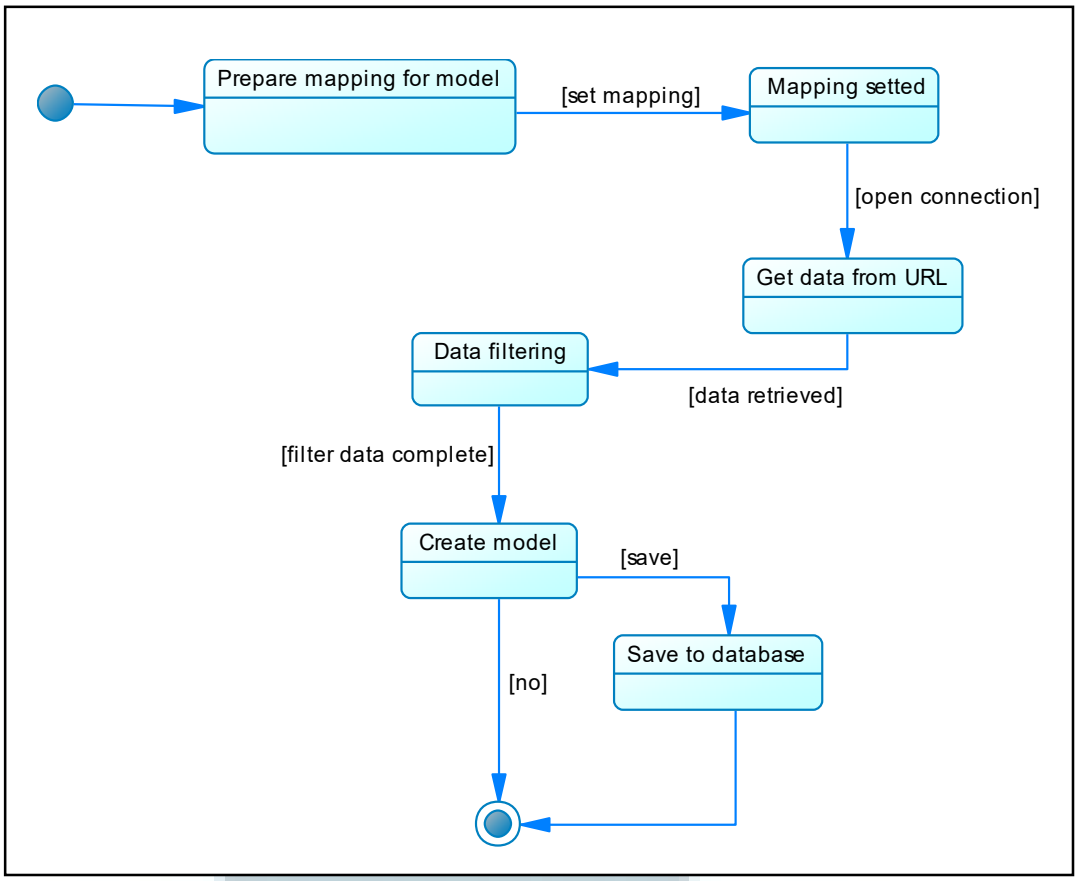


Gambar 3.2 Use Case Diagram

Dalam sistem ini, ada tiga fungsi utama yang dapat dilakukan oleh *programmer* yang menggunakan JSON to Backbone JS Mapper ini. Berikut ini adalah penjelasan singkat dari masing – masing fungsi.

1. Dalam kerjanya, *tool* dapat menerima input berupa sumber dari data yang akan diproses dalam *tool*. Parameter ini dapat berupa URL sumber data, ataupun *object data* yang akan diproses karena *tool* ini dirancang untuk dapat menerima baik dalam bentuk *string* maupun *object*.
2. Sistem ini dilengkapi dengan mekanisme untuk melakukan penyimpanan data hasil dari *mapping* ke dalam *local database*. *User* dapat mengatur sendiri apakah data yang sudah diproses tersebut perlu disimpan atau hanya digunakan untuk keperluan sementara.
3. Yang terakhir adalah user akan mendapatkan *return data* hasil yang merupakan hasil dari *tool processing*. Untuk selanjutnya, data tersebut dapat digunakan untuk keperluan aplikasi yang dikembangkan tersebut.

b. State Chart Diagram



Gambar 3.3 State Chart Diagram

Tahapan awal dari *tool* ini adalah persiapan *mapping* dalam model. Untuk dapat menggunakan metode ini secara efisien, dilakukan modifikasi penambahan *mapping variable* terhadap model yang sudah ada pada Alloy. Dengan adanya *mapping* ini, model dan *tool* sudah memiliki koneksi untuk saling berhubungan.

Proses berikutnya adalah mengambil data yang akan di *parse* dan dicocokkan dengan mapping. Data dikirimkan ketika aplikasi melakukan pemanggilan *class* pada *JSON to Backbone JS Mapper* dalam bentuk parameter berupa *object*. *Object* tersebut berisi URL dari JSON dan kondisi *local* : *true or*

false yang menunjukkan apakah *tool* hanya digunakan untuk *mapping* atau juga akan di simpan dalam *local database*.

Berikut ini adalah beberapa contoh JSON yang harus disesuaikan dengan *tool* ini.

```
{
   "glossary": {
        "title": "example glossary",
        "GlossDiv": {
             "title": "S",
             "GlossList": {
                  "GlossEntry": {
                       "ID": "SGML",
                       "SortAs": "SGML",
                       "GlossTerm": "Standard Generalized Markup Language",
                       "Acronym": "SGML",
                       "Abbrev": "ISO 8879:1986",
                       "GlossDef": {
                            "para": "A meta-markup language,
                                 used to create markup languages such a DocBook.",
                            "GlossSeeAlso": ["GML", "XML"]
                       },
                       "GlossSee": "markup"
                  }
             }
        }
   }
}
```

Gambar 3.4 Contoh JSON 1

(Sumber : http://json.org/example)

```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {
                        "type": "string"
                },
                "lastName": {
                        "type": "string"
                },
        },
        "required": ["firstName", "lastName"]
}
```

Gambar 3.5 Contoh JSON 2

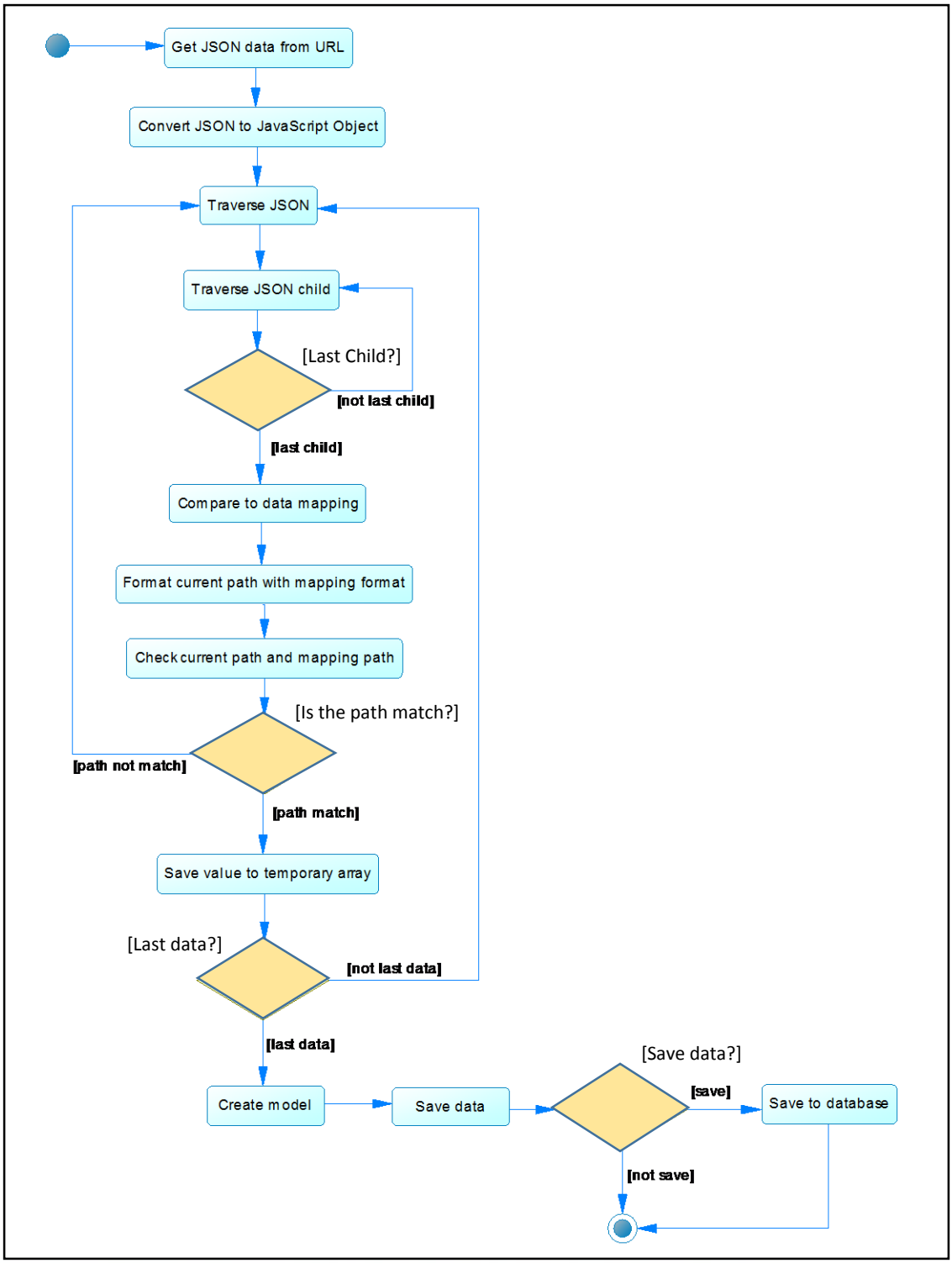
(Sumber : http://json-schema.org/examples.html)

Bentuk JSON diatas adalah beberapa rancangan yang diberikan sebagai bahan patokan dan percobaan terhadap *tool* nantinya. Masih ada banyak sekali macam JSON yang harus disesuaikan. Maka dari itu, dalam tahap perancangan ini, semua kemungkinan harus dipersiapkan untuk menghindari adanya kesalahan karena *tool* yang bisa jadi tidak *compatible* dengan JSON lain.

Untuk metode penambahan *mapping variable* pada model akan dijelaskan dalam proses implementasi. Dalam tahap ini, diasumsikan bahwa data sudah diterima dan siap diproses. Maka dari itu, proses selanjutnya adalah *data filtering*, dimana data akan disaring dan diseleksi kemudian disesuaikan dengan *mapping variable*.

Data yang sudah berhasil diproses dan *valid*, dapat langsung digunakan dalam aplikasi. Data tersebut juga dapat disimpan dalam *local database* berdasarkan kondisi yang didapat dari *object parameter* yang dikirim saat *tool* digunakan.

c. Activity Diagram



Gambar 3.6 Activity Diagram

Activity diagram diatas menggambarkan kegiatan yang dilakukan selama *data filtering* dalam *tool*. Berikut ini penjabaran dari diagram diatas dan fungsi dari masing-masing *activity* yang tercantum.

1. *Get JSON data from URL*, menjadi activity pertama dimana *tool* mengambil JSON yang akan diproses melalui URL yang dikirimkan oleh aplikasi yang menggunakan *tool*. Aplikasi mengirimkan parameter berupa object yang berisi JSON URL dan *save condition* ketika aplikasi melakukan pemanggilan terhadap *class* dalam *tool.*
2. *Convert JSON to JavaScript Object.* Alloy *framework* yang digunakan dalam pengembangan aplikasi ini menggunakan bahasa dasar JavaScript, sedangkan JSON yang didapat masih berupa *string* biasa. Maka dari itu, JSON harus di *convert* terlebih dahulu menjadi *JavaScript Object* agar dapat digunakan.
3. *Traverse JSON*. Untuk melakukan pengecekan terhadap isi JSON, dibutuhkan mekanisme untuk bisa masuk ke data yang paling dalam. Maka dari itu, dalam *tool* ini digunakan metode *recursive* untuk mendapatkan isi data dalam JSON satu persatu secara *detail* dan akurat. Saat *tool* memulai proses *data filtering*, setiap *child* yang terdapat didalamnya diperiksa dan tidak boleh ada yang terlewat.
4. *Traverse JSON child. Activity* ini dilakukan selama proses belum mencapai ujung dari child setiap JSON yang di *filter*. *Recursive* dilakukan dengan kondisi apakah sudah mencapai *child* paling akhir atau belum. Jika sudah maka akan dilanjutkan ke proses berikutnya dan jika belum, proses pengulangan akan terus dilakukan sampai kondisi terpenuhi.
5. *Compare to data mapping.* Setelah mendapatkan *path* dan *value*, langkah berikutnya adalah membuat perbandingan antara *path* yang kita dapat dengan *mapping* yang sudah dibuat. Bentuk path yang kita dapat tidak *compatible* dengan *data mapping* yang dibuat. Hal ini dikarenakan *path* akan berbentuk *array of object* ketika ditelusuri. Maka dari itu dibutuhkan

mekanisme untuk mengubah *format path* agar sesuai dengan format *mapping* pada model yang dibuat.

6. *Format current path with mapping format.* Dalam tahap inilah dilakukan penyesuaian dalam format *current path* yang ada. Bentuk path yang berupa *array of object* di *convert* menjadi sebuah *string* biasa dan disesuaikan dengan *mapping format.*
7. *Check current path and mapping path*. Setelah format sesuai, dilakukan pengecekan terhadap *path* yang sudah kita dapatkan dari proses *traverse JSON* dengan isi *mapping variable* yang ada pada model. Jika ternyata ada yang cocok, maka data akan telebih dahulu disimpan ke dalam *temporary array.* Sedangkan jika tidak ada yang cocok, maka proses *recursive* activity *Traverse JSON* dilanjutkan ke data yang berikutnya. Proses ini dilakukan sampai semua data pada JSON diperiksa.
8. *Save value to temporary array.* Sebelum dimasukkan ke dalam model, data-data yang sudah cocok dengan *mapping variable* disimpan dalam sebuah *array of object* yang menyimpan *path* beserta dengan isi *value.*
9. *Create model.* Setelah data filtering selesai, barulah dibuat mekanisme untuk pembuatan model. Model sendiri sudah merupakan bawaan dasar dari Backbone JS, sehingga *tool* ini bisa langsung berhubungan dan berintegrasi dalam input dan manipulasi data. Karena *tool* ini dapat berhubungan dengan semua jenis model, *programmer* sebagai pengguna hanya tinggal mencocokkan isi model dengan *mapping variable* yang sudah dibuat tanpa harus mengetahui secara detail kerja dari *JSON to Backbone JS Mapper* ini.
10. *Save to database.* Ini merupakan mekanisme terakhir dari *JSON to Backbone JS Mapper.* Pada saat pengiriman *parameter*, bisa disisipkan kondisi apakah data yang sudah didapat akan disimpan atau tidak.

Penulis hanya bertanggung jawab terhadap *tool internal* dan pengembangan modul dari aplikasi Loyalti Express ini. Untuk rancangan dan analisis yang berhubungan dengan *layout* dan *user interface*, dikerjakan oleh *application designer* beserta dengan *programmer* lainnya. Sehingga dalam

pengerjaan *tool* ini, penulis hanya difokuskan pada pembuatan *tool.* Rancangan data yang dibuat dan digunakan juga hanya bersifat *dummy*. Penulis hanya membuat data yang menjadi kriteria penggunaan *tool* sesuai dengan spesifikasi dari aplikasi Loyalti Express ini.

Secara keseluruhan, penulis benar – benar merancang sebuah *tool* yang dapat dikatakan murni terpisah dari pengembangan *layout* dan *flow program* secara keseluruhan. Programmer lain hanya tinggal menggabungkan dan menggunakan fitur dari *tool* ini tanpa tahu sistem dan cara kerja dari *JSON to Backbone JS Mapper* ini.

3.2.3 Development

Selama proses pengembangan dari sistem ini, ada beberapa perangkat keras dan perangkat lunak yang akan digunakan oleh penulis. Berikut ini adalah penjabaran perangkat keras yang digunakan oleh penulis.

1. Laptop ASUS A46CB sebagai perangkat yang digunakan selama pengembangan dengan spesifikasi umum sebagai berikut.
   a. Processor Intel Core i5-3217U (1.70 GHz, Cache 3MB)
   b. VGA NVIDIA GeForce GT740M 2GB
   c. RAM dengan kapasitas 4GB
   d. Harddisk dengan kapasitas 500GB
2. Smartphone Samsung I9300 Galaxy S III sebagai perangkat untuk melakukan *testing* dengan spesifikasi sebagai berikut.
   a. Processor Quad-core 1.4 GHz Cortex-A9
   b. Chipset Exynos 4412 Quad
   c. Operating System Android v4.0.4 (Ice Cream Sandwich)
   d. RAM dengan kapasitas 1GB
   e. Internal Memory dengan kapasitas 16GB
3. Perangkat input yaitu *mouse* dan *keyboard.*
4. Komputer *development server and storage.*

Beberapa perangkat lunak yang digunakan oleh penulis selama tahap pengembangan sistem antara lain sebagai berikut.

1. Sistem operasi Windows 8 untuk komputer penulis.
2. Appcelerator Titanium Studio IDE, digunakan sebagai *coding editor* berbasis JavaScript untuk mengembangkan *Multi-platform mobile application*.
3. Alloy Framework, merupakan framework berbasis *Model, View, and Controller* pada Titanium Studio untuk pengembangan *mobile application.*
4. Android Software Development Kit untuk dapat menjalankan fitur – fitur android pada *emulator*.
5. Android Virtual Emulator, digunakan untuk menjalankan simulasi dari aplikasi yang dibuat.
6. SQLite manager untuk *local database* dari aplikasi Android yang dibuat.
7. Mozilla Firefox yang digunakan untuk melakukan simulasi pembuatan *java script* sebelum dimasukkan ke dalam *tool*.
8. Google Chrome web browser yang digunakan untuk mencari informasi perihal pengembangan sistem.
9. Microsoft Office Excel 2013 untuk membuat timesheet dan laporan pertanggung jawaban kerja.
10. Microsoft Office Word 2013 untuk membuat dokumentasi dan laporan kerja magang.

3.2.4 Integration and Test

Selama pengembangan sistem, ada beberapa pengujian yang dilakukan terhadap mekanisme kerja dari *JSON to Backbone JS Mapper* ini. Pertama adalah mekanisme penerimaan data ke dalam *tool*. *Tool* ini hanya menerima data berupa *JSON object.* Maka dari itu, dilakukan pengecekan apakah data yang masuk adalah benar sebuah *JSON object.* Setelah dilakukan konfirmasi kebenaran jenis data, ada mekanisme pengecekan selanjutnya untuk melihat bentuk struktur *mapping* dari model yang dibuat oleh *programmer* yang

menggunakan *tool*. Bentuk mapping harus sesuai dengan kriteria *tool*. Selain itu, penulis juga melakukan testing terhadap output dari *tool*. Untuk menghasilkan data yang akurat, maka harus dilakukan *testing* untuk semua jenis data untuk menghindari kesalahan yang tidak perlu. Dengan begitu, *tool* ini dapat menjamin data dapat terintegrasi dengan baik selama penggunaannya dalam aplikasi.

Proses pengecekan ini dilakukan dengan berbagai macam data, dan berbagai macam bentuk *JSON object* untuk menghindari kesalahan yang terjadi baik dalam pembacaan maupun *parsing* data. Pembuatan mekanisme pengecekan juga memakan waktu yang cukup lama karena banyaknya jenis data yang mempunyai kemungkinan untuk masuk ke dalam *tool*. Maka dari itu, proses testing sudah dilakukan selama proses pengembangan berlangsung. Proses *testing* ini berjalan beriringan dengan implemetasi coding agar dapat diselaraskan dan dapat langsung diantisipasi ketika ada kesalahan karena mekanisme *parsing* ini cukup panjang dan rumit.

3.2.5 Implementation

Dalam proses implementasi ini, penulis menggabungkan mekanisme pembacaan dan *parsing* berdasarkan analisa algoritma yang sudah dibuat pada tahap awal untuk direalisasikan dalam bentuk program utuh. *Tool* ini merupakan sebuah fitur yang dapat di *inject* secara langsung ke dalam program yang membutuhkan. Maka dari itu, implementasi *coding* dari *tool* ini juga harus bersifat *general* sehingga *compatible* untuk berbagai rancangan aplikasi.

Tool ini juga harus dapat digunakan untuk berbagai macam data yang berasal dari API. Maka dari itu, mekanisme kerja *tool* dibuat umum dan dapat bekerja untuk segala *data resource* dengan catatan bentuk data merupakan *JSON object*. Setelah membuat mekanisme pembacaan data, selanjutnya dibutuhkan penggabungan fungsi antara *model*, *controller* dan *JSON Parse tool* ini. Maka dibuatlah suatu mekanisme untuk menggabungkan dengan menggunakan *mapping* yang sudah penulis jelaskan pada perancangan awal. Dengan *mapping* ini, programmer dapat mengambil data sesuai dengan *model* yang sudah diatur

dan dibuat sejak awal. Untuk menghindari adanya kesalahan dalam *input mapping*, maka penulis membuat kriteria *mapping* yang cukup sederhana untuk meminimalisir kesalahan yang mungkin terjadi selama proses implemetasi aplikasi yang dibuat oleh *programmer* lainnya.

Model dalam aplikasi digunakan untuk tempat pembuatan *mapping* dan melakukan pengecekan apakah data yang dikembalikan sudah cocok dengan hasil *mapping* atau belum. *Controller* digunakan sebagai tempat untuk menjalankan *tool* dan berperan sebagai penghubung antara aplikasi dan model.

Berikut ini adalah contoh cara pemanggilan *tool* dengan parameter, beserta dengan bentuk *mapping variable* yang terdapat dalam model yang diintegrasikan dengan controller.

```
Ti.include("/lib/RemoteDataHandler.js");
var _dataHandler = new RemoteDataHandler();
```

Gambar 3.7 Coding pemanggilan *tool* ke dalam aplikasi

Setelah melakukan deklarasi dari class RemoteDataHandler, kemudia programmer sudah bisa menggunakan *tool* dengan mengirimkan parameter berupa bentuk model yang akan digunakan, beserta dengan aktivitas yang dilakukan jika data fetch sudah selesai diproses. Data yang dikembalikan seperti dapat dilihat pada gambar dibawah ini adalah variable *dataModel*.

```
_dataHandler.fetch({
        model:'card',
        success:function(dataModel) {
                data.pop();
                dataVoucher.pop();
                dataChop.pop();
                buildTableView(dataModel);
        },
        data: {
                code: '1231',
                page: page
        }
});
```

Gambar 3.8 Coding fetch data dari mapper

Untuk lebih jelasnya lagi, berikut ini adalah bentuk model yang digunakan pada aplikasi.

```
exports.definition = {
        config: {
                columns: {
                        "id": "integer primary key autoincrement",
                   "headerFrame" : "text",
                   "headerColor" : "text",
                   "headerFontColor" : "text",
                   "bodyColor" : "text",
                   "bodyFontColor" : "text",
                   "footerFrame" : "text",
                   "footerColor" : "text",
                   "footerFontColor" : "text",
                   "logo" : "text",
                   "title" : "text",
                   "picPromo" : "text",
                   "detail" : "text",
                   "target" : "text",
                   "info" : "text",
                   "type" : "text"
                },
                adapter: {
                        type: "sql",
                        collection_name: "card",
                        db_name: "loyalty",
                        idAttribute: "id",
                        base_url: "http://192.168.1.33:9012/service/api/cards",
                        mapping : {
                                "headerFrame":"Template.FrameHeaderBlack",
                                "headerColor":"Template.HeaderColor",
                        "headerFontColor" : "Template.HeaderFontColor",
                           "bodyColor" : "Template.BodyColor",
                        "bodyFontColor" : "Template.BodyFontColor",
                           "footerFrame" : "Template.FrameFooterBlack",
                           "footerColor" : "Template.FooterColor",
                        "footerFontColor" : "Template.FooterFontColor",
                                "logo":"Company.Logo",
                                "title":"Title",
                                "picPromo" : "Details.Url",
                                "detail-1" : "Details.Username",
                                "detail-2" : "Details.Current",
                                "target" : "Details.Target",
                        "info" : "Info",
                                "type": "Type"

                        }
```

Gambar 3.9 Card.js model

3.2.6 Operation and Maintenance

Tahap *operation* untuk saat ini diserahkan kepada *programmer* yang menggunakan *JSON to Backbone JS Mapper* ini karena penulis hanya bertugas untuk membuat *tool* ini, tidak untuk implementasi *tool* pada aplikasi yang dibuat. Beberapa contoh modul implementasi aplikasi yang menggunakan *tool* ini adalah *ImageView data fetch*, *Contact list generator*, *Login activity for application*, dan *Dynamic Promo Control* pada aplikasi Loyalti Express ini. Untuk beberapa bentuk model dan coding pada JSON to Backbone JS Mapper ini akan dicantumkan dalam lampiran yang juga terdapat daplam laporan praktik kerja magang ini. Tahap *maintenance* sudah pernah sekali dilakukan karena terjadi salah perhitungan algoritma pada implementasinya dan mengakibatkan kesalahan dalam *mapping* data. Namun sudah diperbaiki dan sampai sekarang belum ada kesalahan yang ditemukan.

3.2.7 Penulisan dokumentasi dan laporan magang

Penulisan dokumentasi dan laporan magang dilakukan selama proses pengembangan sistem berlangsung. Setiap harinya, penulis juga diwajibkan untuk membuat perincian kerja harian sebagai control dari kantor untuk mengukur kinerja karyawan. Untuk dokumentasi, penulis cantumkan didalam *tool* yang bersangkutan agar dapat dengan mudah dipelajari oleh *programmer* yang akan menggunakannya. Penulis juga membuat tutorial untuk penggunaan *tool*. Hal ini dimaksudkan untuk memudahkan programmer dalam akses data.

Sebagai gambaran yang lebih jelas, berikut ini penulis tampilkan dalam bentuk tabel yang berisi detail tentang realisasi kerja magang yang dilaksanakan oleh penulis selama bekerja sebagai *programmer* di PT Moonlay Technologies

Tabel 3.2 Realisasi Kerja Magang

Minggu	Kegiatan
1	- Pengenalan pekerjaan dan pembentukan tim - Membuat layout dari web page aplikasi Loyalti Express
2	- Membuat layout dari email page aplikasi Loyalti Express - Membuat layout dengan kriteria Cross Browser Compatibility - Implementasi Media Queries untuk email page
3	- Pengenalan tentang Appcelerator Titanium Studio - Mendalami konsep JavaScript dan pembekalan konsep Alloy framework yang digunakan untuk Appcelerator - Membuat layout untuk main page dari mobile apps Loyalti Express
4	- Pengaturan resolusi apps terhadap gadget - Pengenalan widget untuk promo pada aplikasi Loyalti Express - Membuat promo secara dinamis menggunakan widget - Pengaturan Cross gadget resolution compatibility
5	- Pembelajaran view dan fitur dalam Alloy - Membuat autoscroll untuk image slideshow - Membuat paging control yang digunakan untuk image show
6	- Studi literature tentang SQLite pada android - Membuat model dan database injection untuk *JSON Mapper* - Membuat mekanisme fetch JSON dari server untuk *JSON Mapper* - Implementasi metode recursive untuk JS object pada *JSON Mapper*
7	- Implementasi metode recursive untuk JS object pada *JSON Mapper* - Membuat dynamic modelling untuk *JSON Mapper* - Membuat mekanisme data filtering untuk *JSON Mapper*
8	- Membuat mekanisme mapping dan adjusting data ke dalam database - Revisi dan penyelesaian *tool*

3.3 Kendala yang ditemukan

PT Moonlay Technologies mulai berfokus pada pembuatan produk untuk dapat dipasaarkan kepada pelanggan. Maka dari itu, tentunya akan terus ada pengembangan dan pembuatan sistem baru. Beberapa *tool* dan modul yang dibutuhkan juga diharapkan agar bersifat *reusable* supaya bisa tetap digunakan untuk pengembangan sistem selanjutnya, termasuk *JSON to Backbone JS Mapper* yang dibuat penulis ini.

Untuk membuat sebuah *tool* yang dapat terintegrasi dengan banyak macam data dan berbagai jenis aplikasi tidaklah mudah dan menghadapi berbagai macam kendala. Berikut ini adalah beberapa kendala yang ditemukan dalam membangun *JSON to Backbone JS Mapper* untuk memenuhi kebutuhan dalam pengembangan sistem Loyalti Express baik kendala teknis maupun non teknis.

3.3.1 Kendala Teknis

Beberapa kendala teknis yang menjadi pertimbangan penulis selama melakukan pengembangan sistem adalah sebagai berikut.

a. Penggunaan Appcelerator Titanium Studio dengan Alloy *framework* yang baru pertama kali penulis gunakan dalam pengembangan aplikasi. Untuk membangun detail dari sebuah sistem, tentunya dibutuhkan keahlian yang cukup dalam menguasai bahasa pemrograman yang digunakan selama proses pengembangan. Kendala waktu pengembangan yang cukup lama karena dibarengi dengan studi literatur untuk membangun *tool* sesuai dengan kebutuhan programmer.
b. Bentuk data yang digunakan agar dapat disesuaikan dengan semua jenis aplikasi. Untuk melakukan *parsing* dan *mapping*, tentunya terlebih dahulu harus ditetapkan bentuk data yang cocok agar dapat digunakan secara akurat.
c. Dalam penggunaannya, *tool* ini berfungsi untuk menghubungkan data dengan *local database.* Hal ini menjadi kendala bagaimana membangun mekanisme yang tepat dan akurat sehingga dapat digunakan untuk berbagai jenis data dan terintegrasi dengan baik dalam semua aplikasi berbasis Alloy *framewowk.*

d. Mekanisme untuk pemanggilan *tool*. Banyaknya jenis data yang bisa digunakan membuat pengiriman data dari aplikasi ke dalam *tool* juga dapat berubah-ubah. Dibutuhkan satu mekanisme khusus dalam pengiriman data cocok untuk berbagai aplikasi.
e. Kerja *tool* harus cepat dan ringan. Dalam pembuatan sistem Loyalti Express, dibutuhkan akses kerja *tool* yang cepat dalam proses data. Maka dari itu, pemilihan algoritma dalam kerja *tool* juga harus tepat dan akurat untuk mengurangi *processing time* yang tidak perlu.

3.1.2 Kendala Non Teknis

Beberapa kendala non teknis yang menjadi perhatian penulis selama melakukan pengembangan sistem adalah sebagai berikut.

a. Kebutuhan dari sistem yang sering berubah-ubah dari PT Moonlay Technologies menyebabkan proses pembangunan dan pengembangan sistem menghabiskan waktu yang melebihi penjadwalan awal. Sistem yang seharusnya bisa selesai dalam dua bulan akhirnya memerlukan waktu sampai empat bulan untuk dapat selesai.
b. Keterbatasan *resource* dari yang disediakan kantor. Dalam pembuatan *mobile application* tentunya dibutuhkan *device* yang memadai untuk melakukan *testing* dan tidak disediakan oleh pihak kantor. Maka dari itu, *programmer* hanya berpatokan pada *testing* di *android emulator*. Hal ini juga dinilai cukup menghambat karena proses *debug* pada *emulator* cenderung memakan waktu yang cukup lama.
c. Ada beberapa penerapan konsep yang berbeda pada pengembangan sistem dengan teori-teori yang diberikan selama proses perkuliahan.

3.4 Solusi atas kendala yang ditemukan

Setiap kendala dalam pengembangan sebuah sistem, pasti ada solusi yang dapat ditemukan untuk menyelesaikan permasalahan tersebut. Untuk memenuhi kebutuhan-kebutuhan yang sudah dipaparkan dan mengatasi kendala-kendala yang dihadapi, maka didapatkan beberapa solusi dan keputusan sebagai berikut.

3.4.1 Solusi untuk kendala teknis

Beberapa solusi yang ditemukan menyelesaikan beberapa kendala teknis dalam pengembangan sistem adalah sebagai berikut.

a. Dilakukan studi literatur dan pembelajaran terhadap Titanium Studio, Alloy *framework*, dan Backbone JS selama proses pengembangan. Selain itu, diperlukan juga pembelajaran terhadap beberapa project yang sudah pernah dikerjakan oleh PT Moonlay Technologies sebagai patokan dalam pembuatan *JSON to Backbone JS Mapper* ini.
b. Diputuskan bahwa data yang akan digunakan akan diselaraskan menggunakan JSON. JSON sendiri merupakan bentuk data yang paling mudah diterapkan dan dapat memuat banyak *content* dalam konteks aplikasi Loyalti Express ini. Bentuk JSON kemudian akan di *convert* menjadi *JavaScript Object* untuk dapat digunakan dalam Alloy *framework.* Hal ini dinilai efisien, karena selain bentuknya yang dinamis, mayoritas data yang berasal dari API juga berbentuk JSON object sehingga dapat dimanipulasi dengan mudah menggunakan *tool* ini.
c. Untuk dapat menyatukan data dengan *local database* mekanisme yang ditemukan cukup efisien dan akurat adalah menggunakan metode *mapping.* Dalam model yang terdapat pada Backbone JS, bisa dibuat dan diatur beberapa *mapping variable* yang dapat disesuaikan dengan *content* dari model yang digunakan dalam aplikasi.
d. Untuk pemanggilan *tool* dari aplikasi yang bersangkutan digunakan *function* dengan *parameter* yang berisi *object* yang terdiri dari *data source* dan kondisi untuk menyimpan data atau tidak ke dalam *local database.* Hal ini dinilai efisien karena *programmer* yang akan menggunakan *tool* tidak perlu repot untuk melakukan analisa dan implementasi data terhadap *tool*, melainkan hanya tinggal mengirimkan *parameter* dan data siap diproses dan digunakan untuk aplikasi. Dengan metode pengiriman parameter ini, *tool* juga menjadi fleksibel dan dapat diintegrasikan ke berbagai aplikasi yang berbasis Appcelerator Titanium Studio menggunakan Alloy *framework.*
e. Untuk membuat suatu mekanisme *tool* yang bergerak cepat dan akurat, dibutuhkan algoritma yang pas dalam *data processing.* Akhirnya kendala ini dapat

diminimalisir dengan menggunakan algoritma *recursive* dalam *data processing* pada *JSON Object.* Sudah dilakukan *testing* dengan berbagai algoritma lainnya dan tidak memuaskan. Metode *recursive* ini yang dinilai paling cocok dan efisien untuk diimplementasikan ke dalam *JSON to Backbone JS Mapper* ini.

3.4.2 Solusi Non Teknis

Beberapa solusi yang ditemukan menyelesaikan beberapa kendala teknis dalam pengembangan sistem adalah sebagai berikut.

a. Untuk menangani seringnya terjadi perubahan selama proses pengembangan sistem dari pihak Moonlay, maka proses pembangunan sistem dibagi menjadi beberapa fase yang terdiri dari tahap penerimaan kebutuhan, proses perancangan, dan proses uji coba. Selain itu, diadakan rapat harian untuk pembahasan kerja untuk menghindari salah koordinasi dalam pengerjaan *tool.* Hal ini dimaksudkan juga agar deadline untuk setiap *project* yang dikerjakan tidak mundur karena adanya kesalahan dasar dalam pengembangan sistem.
b. Dalam pengembangan *mobile application*, pihak kantor tidak menyediakan *resource* yang memadai untuk melakukan *product testing.* Maka dari itu, digunakanlah *device* pribadi sebagai *resource* untuk melakukan *testing*. Selain lebih cepat, *testing* pada *device* juga bisa digunakan untuk mengukur kemampuan *tool* secara langsung karena dijalankan pada device yang nantinya akan dipakai sebagai tempat aplikasi tersebut dijalankan.
c. Diterapkannya konsep-konsep yang memang berbeda dengan teori selama perkuliahan guna mensiasati aplikasi yang dihasilkan agar sesuai dengan permintaan dan kebutuhan. Selain itu, ada banyak sekali faktor yang mempengaruhi proses pengembangan sistem baik secara internal maupun eksternal.