BAB III

Landasan Teori

Pada bab ini akan dibahas mengenai beberapa landasan teori yang digunakan untuk perancangan dan pembuatan aplikasi survei berbasis IOS yang bisa dijadikan sebagai acuan.

3.1 *Crowdsourcing*

Menurut Paul Whitla (2009), *crowdsourcing* merupakan suatu metode baru yang mereferensi dari kegiatan *outsourcing* yang dilakukan oleh perusahaan ke suatu komunitas atau perkumpulan *online* dimana perusahaan menawarkan suatu hadiah kepada komunitas tersebut bila berhasil menyelesaikan beberapa tugas yang ditentukan oleh perusahaan (Whitla, 2009). Perusahaan melakukan *crowdsourcing* dikarenakan komunitas bisa melakukan tugas-tugas dengan lingkup yang luas dan tugas-tugas tersebut dapat terselesaikan dengan cepat daripada dikerjakan oleh orang perusahaan sendiri. Alasan lain dari dilakukannya *crowdsourcing* adalah bisa mendapatkan akses ke komunitas yang sangat besar dengan potensial dan *skill* yang sangat beragam dan mau untuk melakukan tugas yang diberikan dengan waktu yang sudah ditentukan dan terkadang meringankan beban dari pelaksana *crowdsourcing* tersebut.

Berdasarkan hasil penelitian dari Daren C. Brabham (2008), kegiatan *crowdsourcing* tidak bisa disamakan dengan *open source* dikarenakan *open source* memberikan akses ke elemen penting dari produk yang dibuat kepada orang awam dengan tujuan untuk pengembangan kolaboratif

pada produk yang sudah ada dengan transparansi yang berkontinu dan distribusi yang gratis dari produk tersebut, sedangkan *crowdsourcing* lebih menekankan mengenai pemberian suatu permasalahan untuk dipecahkan dan akan diberikan suatu hadiah kepada komunitas yang bisa menyelesaikan tugas yang diberikan (Barbham, 2008). *Crowdsourcing* merupakan pengembangan dari konsep *open source* dimana dibutuhkan komunitas dari pihak luar untuk berpartisipasi dalam kegiatan atau tugas yang ditawarkan, namun bila *open source* biasanya diikuti oleh komunitas atas dasar hobi atau memang untuk memenuhi kebutuhan dari komunitas tersebut, *crowdsourcing* diikuti oleh komunitas dikarenakan ada suatu hadiah yang akan diberikan oleh pelaksana *crowdsourcing* bila berhasil menyelesaikan tugas yang diberikan.

Crowdsourcing menurut hasil penelitian dari Paul Whitla (2009), memiliki beberapa tahapan dalam pengimplementasiannya oleh suatu lembaga atau perusahaan (Whitla, 2009). Langkah pertama adalah perusahaan atau lembaga mengidentifikasi terlebih dahulu tugas dan fungsi yang sedang dihadapi dan diujicoba. Daripada tugas tersebut dilanjutkan oleh pihak internal, maka lebih baik tugas tersebut dilepas kepada komunitas luar yang sudah diundang untuk menyelesaikan tugas tersebut dengan imbalan yang diberikan oleh lembaga atau perusahaan. Komunitas yang diundang oleh lembaga bisa saja hanya orang-orang yang merasa tertarik untuk menyelesaikan tugas tersebut, atau bisa juga orang-orang yang memiliki keahlian dalam bidang pekerjaan tersebut, atau bisa juga kombinasi

dari kedua jenis tersebut. Komunitas yang bersedia untuk melakukan pekerjaan tersebut akan diberikan waktu yang sudah ditentukan oleh lembaga atau perusahaan, kemudian bila sudah menyelesaikan tugas tersebut maka lembaga atau perusahaan akan mengecek hasil pekerjaan apakah sudah valid atau belum, bila sudah maka perusahaan atau lembaga akan memberikan hadiah kepada komunitas yang mengerjakan tugas tersebut.

Berdasarkan hasil penelitian dari Aditi Misra dkk (2014), pengumpulan data dengan menggunakan metode *crowdsourcing* merupakan suatu cara yang paling efisien secara finansial pada kasus dimana para pengikut *crowdsourcing* merupakan grup kecil tertapi sangat antusias dan termotivasi untuk mengikuti proyek *crowdsourcing* yang dijalankan (Misra, Gooze, Watkins, Asad, & Le Dantec, 2014), sehingga penting bagi pelaksana *crowdsourcing* untuk membuat suatu kriteria tertentu bagi pengguna yang ingin mengikuti *crowdsourcing* sehingga nantinya bisa didapatkan data yang revelan dan akurat.

3.2 Pola Arsitektur Model View Presenter

Modal View Presenter (MVP) adalah salah satu pola arsitektur yang digunakan di dalam pengembangan suatu perangkat lunak komputer baik untuk versi *desktop,* web, maupun *mobile* dimana MVP bisa diterapkan di segala jenis bahasa pemrograman seperti *Java, C#, Swift, C++,*dan lain lain. MVP merupakan pengembangan dari pola arsitektur yang sudah terkenal dan sampai sekarang masih digunakan, yaitu *Model View Controller (MVC)* dimana perbedaan mendasar dari MVC dan MVP adalah

penggunaan kelas untuk pengolahan data nya, bila di MVC digunakan *controller* maka di MVP digunakan kelas *presenter*. Di pola arsitektur MVC, *controller* bertugas untuk melakukan pengubahan nilai – nilai yang ada di model kemudian hasil tersebut diberitahukan kembali kepada *view* untuk diolah oleh *view*. Pada pola arsitektur MVP, *presenter* bertugas untuk mengontrol aliran data dengan model, kemudian pengolahan data dilakukan di *presenter* kemudian setelah data diolah maka presenter hanya memanggil fungsi yang sudah dideklarasikan di *view*, sehingga *view* hanya melakukan perubahan tampilan saja tanpa mengubah data sedikitpun.

Menurut Coskun Aygun dan Emre Kazan (2015), pola arsitektur MVP memiliki esensi untuk menerima *user interface code* di kelas *view* dan membawanya ke setiap kelas *presenter* yang berbeda, sehingga *presenter* bisa berjalan dan di tes secara independen tanpa mempengaruhi pembuatan dan pengolahan kode dari *view* (Agyun & Kazan, 2015). Dengan sifatnya yang independen, penggunaan pola arsitektur MVP akan mengurangi terjadinya *force close* di bagian kelas *view* dikarenakan pengolahan data dilakukan semua oleh *presenter*, sehingga walaupuan terdapat kesalahan dalam pengolahan data di *presenter* tidak akan membuat program *force close*. Kelebihan lain dari pola arsitektur MVP adalah memperpendek kode yang digunakan baik di dalam *view* dikarenakan semua pengolahan data dilimpahkan ke *presenter* dan di dalam satu *view* bisa diimplementasikan lebih dari satu *presenter* untuk mengolah data.

Pengolahan arsitektur MVP memiliki beberapa kelemahan dibandingkan dengan pola arsitektur MVC,

dimana menurut M. Rizwan Jameel dan Fatina Sabir (2013) pola arsitektur MVP susahnya untuk mengimplementasikan cara untuk menghubungkan kelas *view* dan kelas *presenter* serta model yang yang tidak memperhatikan *presenter* akan gampang untuk diubah sehingga bisa berpengaruh terhadap hasil yang didapat dari *view* (Qureshi & Sabir, 2013). Kesusahan dalam menghubungkan *view* dengan *presenter* sendiri dikarenakan ada beberapa bahasa pemrograman yang tidak memiliki *user interface* bawaan sehingga harus membuatnya secara manual, serta cara kerja *user interface* tiap bahasa pemrograman juga berbeda, misalnya dalah *user interface* di *java* sangat berbeda dengan *user interface* yang digunakan di bahasa *swift* walaupun struktur nya hampir mirip. Kelemahan lainnya dari pola arsitektur MVP adalah susahnya dalam pecalakan kesalahan yang terjadi pada aplikasi, hal ini dikarenakan aplikasi tidak akan mengalami *force close* yang diakibatkan oleh kesalahan di *presenter* sehingga tidak bisa diketahui kesalahan yang terjadi. Agar kesalahan di *presenter* bisa dilacak perlu dilakukan pencatatan log untuk setiap proses pengolahan data serta pemberian *error handling* yang baik sehingga kesalahan bisa dilacak dengan men – *debug* aplikasi untuk melihat log yang ditampilkan tiap prosesnya.

Cara kerja pola arsitektur MVP sendiri tidak terlalu kompleks dan hampir mirip dengan MVC, dimana saat kelas *view* mengalami suatu *event* seperti tombol diklik maka kelas *view* akan memanggil suatu fungsi dari kelas *presenter* yang sudah dideklarasikan di dalam *view*, kemudian di dalam fungsi tersebut *presenter* akan memanggil fungsi dari *user interface* yang

diimplementasikan oleh kelas *view* seperti fungsi untuk memunculkan *progress bar*, lalu akan dilakukan pengolahan data yang melibatkan model. Setelah pengolahan data selesai maka *presenter* akan memanggil fungsi *user interface* untuk mengirimkan hasil pengolahan data ke *view*. Jika dilihat dari cara kerja tersebut maka *view* hanya bersifat pasif saja yang tugasnya hanya memberikan notifikasi kepada *presenter* untuk melakukan pengolahan data dan mendapatkan hasil pengolahan data tersebut untuk ditampilkan, berbeda dengan pola arsitektur MVC yang kelas *view* nya masih bersifat aktif dan bisa mengakses kelas model.

3.3 *Gamification*

Gamfication menurut Hee Jung Park dan Jae Hwan Bae (2014) adalah suatu pergerakan baru untuk membuat efek di aplikasi *non-game* dengan mengimplementasikan mekanika dan pola pikir dari *game* yang membuat aplikasi menjadi seru dan menyenangkan (Jung Park & Hwan Bae, 2014). Penerapan *gamification* di suatu aplikasi bisa memberikan rasa nyaman dan bisa menarik pengguna untuk berpartisipasi dalam memecahkan suatu permasalahan dengan menggunakan metode yang menyenangkan. Penggunaan *gamification* memberikan keuntungan bagi pengguna dimana pengguna bisa mendapatkan suatu hadiah dari penyelesaian suatu masalah atau target yang harus dipenuhi di aplikasi, dimana hadiah dari aplikasi bisa berupa kenaikan level, kenaikan poin, kenaikan ranking serta pemberian hadiah lainnya tergantung dari aplikasi tersebut sehingga tetap bisa menjaga keikutsertaan pengguna dalam menggunakan aplikasi tersebut. *Gamification* menggunakan teknik dengan melihat peluang

dari sifat alami dari orang – orang yaitu keinginan untuk berkompetisi, mendapatkan status dan penghargaaan dan suatu kebanggaan diri. *Gamification* bisa memenuhi kebutuhan alami dari orang – orang dengan memberikan suatu penghargaan atas suatu penyelesain masalah serta memberikan nuansa kompetisi dengan memberikan ranking untuk tiap pengguna sehingga pengguna menjadi lebih tertantang dan semakin terlarut dalam penggunaan aplikasi tersebut.

Gamification pada saat ini sudah menjadi poin *marketing* yang sangat penting bagi para pengembang dikarenakan dari *gamification* pengembang bisa mendapatkan pemasukan yang tinggi dimana menurut Tracy Harwood dan Tony Gary (2015) pendapatan total dunia dari *gamification* pada tahun 2012 mencapai $938 juta yang tentu sangat menggiuran bari para pengembang dan perusahaan pembuat aplikasi (Harwood & Garry, 2015) . Monetisasi dari hasil *gamification* memang menjadi salah satu cara dari pengembang untuk mendapatkan penghasilan dari aplikasi yang telah dibuat, biasanya aplikasi tersebut bersifat gratis dalam pengunduhannya. Monetisasi dalam *gamification* sendiri tidak mengganggu pengguna dalam menggunakan aplikasi dikarenakan tidak seperti iklan – iklan yang muncul dengan sendirinya, pengguna bebas untuk memilih apakah ingin melakukan pembelian untuk *gamification* ataupun tidak.

Bentuk penerapan *gamification* yang sangat populer pada aplikasir menurut Darina Dicheva dkk (2014) adalah poin, *badges,* level, dan *leaderboard,* (Dicheva, Dichev, Agre, & Angelova, 2014). Pada jenis penerapan poin, aplikasi menawarkan suatu hadiah kepada pengguna bila

berhasil memperoleh poin dengan jumlah tertentu yang sudah ditentukan, atau membuat pengguna bisa melakukan transaksi pembelian dengan poin yang sudah dikumpulkan. Pada *badges*, aplikasi menawarkan suatu penghargaan kepada pengguna jika berhasil menyelesaiakan suatu tantangan maupun suatu permasalahan yang diberikan oleh aplikasi dimana bila pengguna berhasil mendapatkan penghargaan tersebut, akan muncul kepuasan diri dan kebanggaan atas usaha yamg dilakukan oleh pengguna. Pada penerapan jenis level, pengguna bisa menaikkan level mereka dengan melakukan suatu kegiatan dengan aplikasi tersebut dimana semakin tinggi levelnya semakin banyak hal yang harus dilakukan oleh pengguna di aplikasi tersebut. Pada *leadearboard*, aplikasi mengajak pengguna untuk berkompetisi dengan pengguna lain untuk bisa mencapai *ranking* yang tertinggi dengan cara mengumpulkan poin atau level sehingga membuat pengguna akan sering untuk menggunakan aplikasi tersebut.

3.4 IOS

IOS merupakan salah satu produk sistem operasi yang dikembangkan oleh *Apple* untuk perangkat *mobile* dimana sistem operasi ini hanya dikhususkan untuk perangkat *Iphone*, *Ipad,* dan *Ipod* yang dibuat juga oleh *Apple*. Menurut data dari Nathan Ingraham (2014), perangkat IOS yang tersebar di seluruh dunia berjumlah 800 juta buah dimana terdiri dari 100 juta *Ipoud,* 200 juta *Ipad,* dan 500 juta *Iphone* dari segala versi, walaupun masih kalah dengan perangkat yang mengadopsi sistem operasi Android yang berjumlah 900 juta (Ingraham, 2014). Berdasarkan info statistik tersebut

bisa dilihat bahwa IOS sudah sangat berkembang dan diminati oleh banyak orang.

Menurut Divya Singla dan Luv Mendiratta (2014), sistem operasi IOS memiliki beberapa kelebihan dibandingkan dengan sistem operasi Android, antara lain adalah IOS memiliki sistem yang stabil dan aman untuk *smartphone*, desain *user interface* yang bagus dan elegan yang didukung pula dengan perangkat keras *Apple*, terdapat sedikit *bug* yang didukung oleh standarisasi keamanan yang tinggi, dukungan pembaharuan perangkat yang gratis dan hampir semua perangkat *Apple* bisa melakukan pembaharuan , dan akses yang mudah untuk aplikasi – aplikasi yang ada di *apps store* (Singla & Mendratta, 2014). Kelebihan – kelebihan yang dimiliki IOS membuat sistem operasi ini masih bertahan sampai sekarang dan jumlah perangkat yang mengadaptasi IOS tidak kalah dengan Android.

Pengembangan aplikasi untuk sistem operasi IOS tidak kalah juga dengan Android dimana menurut Nathan Ingraham (2014) pada *app store*, jumlah aplikasi yang beredar adalah 1,2 juta dengan 75 juta unduh serta 300 juta orang mengakses *app store* setiap minggunya (Ingraham, 2014). Dari statistik tersebut bisa dilihat bahwa pertumbuhan aplikasi di sistem operasi IOS sangat besar dan masih menjadi peluna bagi para pengembang aplikasi untuk membuat aplikasi di IOS. Pengembangan aplikasi di IOS sendiri menggunakan *tool Xcode* yang dibuat khusus oleh *Apple* sendiri untuk pembuatan aplikasi dan pengembangan menggunakan bahasa

pemrogramanan *Swift* atau *Objective-C* dimana menurut Mark H. Goadrich dan Michael P. Rogers (2011) pengembangan aplikasi IOS harus menggunakan laptop yang memiliki sistem operasi Mac OS yang dimiliki oleh *Apple* sehingga pengembang aplikasi harus memiliki *macbook* buatan Apple (H. Goadrich & P. Rogers, 2011). Bila pengembang aplikasi kekurangan dana untuk membeli *macbook* baru, maka bisa meminjam laptop teman atau membeli *macbook* bekas dimana tidak masalah *macbook* tersebut bekas dikarenakan pengembangan aplikasi tidak memakan memori yang besar serta tidak membutuhkan prosesor yang kencang.