TUGAS AKHIR – KS 141501

RANCANG BANGUN APLIKASI MOBILE CLIENT-SIDE BERBASIS ANDROID DENGAN DESIGN PATTERN MODEL-VIEW-VIEWMODEL (STUDI KASUS : PAJAKCERDAS)

DESIGN AND DEVELOPMENT OF ANDROID BASED MOBILE CLIENT-SIDE APPLICATION USING MODEL-VIEW-VIEWMODEL DESIGN PATTERN (CASE STUDY : PAJAKCERDAS)

ACHMAD FARHAN MUSTAQIM
NRP 05211540000151

Dosen Pembimbing
Nisfu Asrul Sani, S.Kom., M.Sc.
Hatma Suryotrisongko, S.Kom., M.Eng.

JURUSAN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – KS 141501

RANCANG BANGUN APLIKASI MOBILE CLIENT-SIDE BERBASIS ANDROID DENGAN DESIGN PATTERN MODEL-VIEW-VIEWMODEL (STUDI KASUS : PAJAKCERDAS)

ACHMAD FARHAN MUSTAQIM
NRP 05211540000151

Dosen Pembimbing
Nisfu Asrul Sani, S.Kom., M.Sc.
Hatma Suryotrisongko, S.Kom., M.Eng.

JURUSAN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

FINAL PROJECT – KS 141501

DESIGN AND DEVELOPMENT OF ANDROID BASED MOBILE CLIENT-SIDE APPLICATION USING MODEL-VIEW-VIEWMODEL DESIGN PATTERN (CASE STUDY : PAJAKCERDAS)

ACHMAD FARHAN MUSTAQIM
NRP 05211540000151

Supervisor
Nisfu Asrul Sani, S.Kom., M.Sc.
Hatma Suryotrisongko, S.Kom., M.Eng.

DEPARTMENT OF INFORMATION SYSTEMS
Faculty of Information Technology and Communication
Sepuluh Nopember Institute of Technology
Surabaya 2019

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI MOBILE CLIENT-SIDE BERBASIS ANDROID DENGAN DESIGN PATTERN MODEL-VIEW-VIEWMODEL (STUDI KASUS : PAJAKCERDAS)

TUGAS AKHIR
Disusun untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
ACHMAD FARHAN MUSTAQIM
NRP. 05211540000151

Surabaya, Juli 2019

KEPALA
DEPARTEMEN SISTEM INFORMASI

Mahendrawati ER, S.t., M.Sc., Ph.D
NIP. 19761011 200604 2 001

LEMBAR PERSETUJUAN

RANCANG BANGUN APLIKASI MOBILE CLIENT-SIDE BERBASIS ANDROID DENGAN DESIGN PATTERN MODEL-VIEW-VIEWMODEL (STUDI KASUS : PAJAKCERDAS)

TUGAS AKHIR
Disusun untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
ACHMAD FARHAN MUSTAQIM
NRP. 05211540000133

Disetujui Tim Penguji : Tanggal Ujian : Juli 2019
Periode Wisuda : September 2019

Nisfu Asrul Sani, S.Kom., M.Sc. **(Pembimbing I)**

Hatma Suryotrisongko S.Kom., M.Eng. **(Pembimbing II)**

Bekti Cahyo Hidayanto, S.Si, M.Kom **(Penguji I)**

Ahmad Mukhlason, S.Kom., M.Sc. Ph.D **(Penguji II)**

RANCANG BANGUN APLIKASI MOBILE CLIENT-SIDE BERBASIS ANDROID DENGAN DESIGN PATTERN MODEL-VIEW-VIEWMODEL (STUDI KASUS : PAJAKCERDAS)

Nama Mahasiswa	**:**	**Achmad Farhan Mustaqim**
NRP	**:**	**05211540000151**
Jurusan	**:**	**Sistem Informasi**
Dosen Pembimbing I	**:**	**Nisfu Asrul Sani, S.Kom., M.Sc.**
Dosen Pembimbing II	**:**	**Hatma Suryotrisongko S.Kom., M.Eng.**

ABSTRAK

Sistem operasi Android semakin lama semakin populer dikalangan para pengguna ponsel pintar. Google Play Store yang kita kenal sebagai marketplace utama pada Android telah memiliki jutaan aplikasi yang siap untuk diunduh berkat antusiasme pengembang aplikasi yang sangat tinggi. Menjadi seorang pengembang aplikasi berbasis Android merupakan sebuah tantangan tersendiri, salah satunya adalah ketatnya persaingan antar aplikasi untuk merebut pangsa pasar yang ada. Adanya persaingan tersebut secara otomatis akan memaksa pengembang aplikasi untuk terus memperbarui dan menambah berbagai fitur pada aplikasinya.

Kedua hal tersebut tidaklah mudah untuk ditangani. Untuk mengatasi tantangan tersebut, salah satu solusi yang sering digunakan bagi para pengembang aplikasi untuk terus bersaing dengan baik adalah dengan mengimplementasikan sebuah design pattern pada aplikasi mereka. Implementasi sebuah design pattern diharapkan mampu meningkatkan daya saing aplikasi dikarenakan sifatnya yang mendukung aspek maintainability dan modularity. Kedua aspek tersebut merupakan aspek pendukung utama bagi terciptanya aplikasi yang bersaing dan sustainable.

Akan tetapi, keputusan untuk mengimplementasi sebuah software design pattern seringkali menimbulkan permasalahan yang baru. Sebuah design pattern membutuhkan beberapa aturan-aturan yang wajib dipatuhi bagi para pengembang aplikasi demi terlaksananya implementasi design pattern tersebut. Para pengembang aplikasi yang belum terbiasa dengan hal tersebut menjadi kesulitan, bahkan bisa menjadi penghalang bagi terciptanya aplikasi mereka.

Dengan menggunakan software design pattern MVVM(Model-View-ViewModel), penulis akan membandingkan tahap pengembangan aplikasi pada studi kasus aplikasi PajakCerdas dengan menggunakan design pattern MVVM dibandingkan dengan menggunakan design pattern MVC. Untuk menjaga objektivitas penilaian, akan digunakan software metrics tools yang bernama JHawk 6 pada saat tahap perbandingan berlangsung. Diharapkan hasil dari penelitian ini adalah terlaksananya perbandingan yang objektif sehingga dapat menghasilkan kesimpulan mengenai signifikansi penggunaan software design pattern untuk pengembangan aplikasi Android pada studi kasus ini.

Kata kunci : Android, Design Pattern, MVVM, MVC, JHawk 6.

DESIGN AND DEVELOPMENT OF ANDROID BASED MOBILE CLIENT-SIDE APPLICATION USING MODEL-VIEW-VIEWMODEL DESIGN PATTERN (CASE STUDY : PAJAKCERDAS)

Name	**:**	**Achmad Farhan Mustaqim**
NRP	**:**	**05211540000151**
Department	**:**	**Information Systems**
First Supervisor	**:**	**Nisfu Asrul Sani, S.Kom., M.Sc.**
Second Supervisor	**:**	**Hatma Suryotrisongko S.Kom., M.Eng.**

ABSTRACT

The Android operating system is increasingly popular among smart phone users. The Google Play Store that we know as the main marketplace on Android has millions of applications ready to download thanks to the enthusiasm of so many application developers. Being an Android-based application developer is a challenge, one of which is the tight competition between applications to capture existing market share. The existence of such competition will automatically force application developers to continue to update and add various features to the application.

Both of these are not easy to handle. To overcome this challenge, one of the solution that is often used by application developer is to continue to compete well is to implement a design pattern on their application. The implementation of a design pattern is expected to be able to improve the competitiveness of applications due to its nature that supports aspects of maintainability and modularity. Both of these aspects are the main supporting aspects for the creation of competitive and sustainable applications.

However, the decision to implement a pattern design software often raises new problems. A design pattern requires several rules that must be adhered to for application developers to implement the design pattern. Application developers who are not familiar with it will face a numerous problem, it can even be a barrier to the creation of their applications.

By using MVVM (Model-View-View-Model) design pattern, the author will compare the application development stage in the PajakCerdas application case study using the MVVM design pattern compared to using the MVC (Model-View-Controller) design pattern. To maintain the objectivity of the assessment, a software metrics tool called JHawk 6 will be used when the comparison stage takes place. It is expected that the results of this study are the implementation of objective comparisons so as to produce conclusions regarding the significance of the use of software design patterns for the development of Android applications in this case study.

Keywords : Android, Design pattern, MVVM, MVC, JHawk 6.

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, Tuhan Semesta Alam yang telah memberikan kekuatan serta hidayah-Nya kepada penulis sehingga penulis dapat menyelesaikan tugas akhir ini yang merupakan salah satu syarat kelulusan di Jurusan Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya.

Terima kasih penulis sampaikan kepada pihak-pihak yang telah mendukung, memberikan saran, motivasi, semangat, dan bantuan baik berupa materiil maupun moril demi tercapainya tujuan pembuatan tugas akhir ini. Secara khusus penulis akan menyampaikan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Segenap keluarga besar terutama kedua orang tua dan adik penulis, Bapak Zaenal Mustaqim, S.E., Ibu Sri Hartati Kurnianda, dan Athallah Fakhri Mustaqim yang senantiasa mendoakan, memberikan motivasi dan semangat, sehingga penulis mampu menyelesaikan Pendidikan S1 ini dengan baik.
2. Bapak Dr. Ir. Aris Tjahyanto, M.Kom. sebagai dosen wali penulis selama menempuh pendidikan di Departemen Sistem Informasi.
3. Ibu Mahendrawati ER, ST, M.Sc, Ph.D, selaku Kepala Departemen Sistem Informasi ITS, Bapak Nisfu Asrul Sani, S.Kom., M.Sc selaku Kaprodi S1 Sistem Informasi ITS serta seluruh dosen pengajar beserta staf dan karyawan Departemen Sistem Informasi, FTIK ITS Surabaya selama penulis menjalani kuliah.
4. Bapak Nisfu Asrul Sani, S.Kom., M.Sc. dan Bapak Hatma Suryotrisongko, S.Kom., M.Eng. selaku dosen pembimbing yang telah banyak meluangkan waktu untuk membimbing, mengarahkan, dan mendukung dengan memberikan ilmu, petunjuk, dan motivasi dalam penyelesaian Tugas Akhir ini.

5. Bapak Bekti Cahyo Hidayanto, S.Si, M.Kom. dan Bapak Ahmad Muklason, S.Kom., M.Sc., Ph.D. selaku dosen penguji yang telah memberikan kritik, saran, dan masukan penyempurnaan Tugas Akhir ini.
6. Teman-teman Sistem Informasi Angkatan 2015 (LANNISTER) yang senantiasa menemani dan memberikan motivasi bagi penulis selama perkuliahan hingga dapat menyelesaikan Tugas Akhir ini.
7. Rama Rahmanda, S.Kom dan Ginanjar Prabowo, S.Kom selaku kakak tingkat penulis yang senantiasa membimbing semenjak masa ospek hingga masa penyusunan Tugas Akhir ini.
8. Serta seluruh pihak-pihak lain yang tidak dapat disebutkan satu per satu, yang telah banyak membantu penulis selama perkuliahan hingga dapat menyelesaikan Tugas Akhir ini.

Penyusunan laporan tugas akhir ini masih jauh dari kata sempurna sehingga penulis menerima adanya kritik maupun saran yang membangun untuk perbaikan di masa yang akan datang. Semoga buku tugas akhir ini dapat memberikan manfaat bagi pembaca.

Surabaya, 22 Juli 2018

Penulis,

Achmad Farhan Mustaqim

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

Pada bab pendahuluan akan diuraikan proses identifikasi masalah penelitian yang meliputi latar belakang masalah, perumusan masalah, batasan masalah, tujuan tugas akhir, manfaat kegiatan tugas akhir dan relevansi terhadap pengerjaan tugas akhir. Berdasarkan uraian pada bab ini, harapannya gambaran umum permasalahan dan pemecahan masalah pada tugas akhir dapat dipahami.

1.1 Latar Belakang

Sistem operasi Android sebagai salah satu dari sekian banyak sistem operasi berbasis *mobile* dapat dikatakan merupakan sistem operasi tersukses dilihat dari jumlah penggunanya dibandingkan dengan sistem operasi yang lain. Menurut data yang dibawakan oleh StatCounter, terhitung pada bulan Januari tahun 2019, Android mendapatkan pangsa pasar sebesar 74.45%, unggul jauh dari kompetitor terdekatnya yakni iOS yang "hanya" mendapatkan pangsa pasar tidak lebih dari 25%, atau lebih tepatnya sebesar 22.85%[1]. Tidak hanya itu, Google Play yang merupakan sumber utama peredaran aplikasi di Android tercatat akhir-akhir ini telah mempunyai setidaknya 2.9 juta aplikasi didalamnya, unggul dari pesaing terdekatnya yakni iOS App Store yang mempunyai 1.85 juta aplikasi[2].

Banyaknya aplikasi yang ada didalam sebuah ekosistem tentu akan menimbulkan persaingan yang ketat. Persaingan tersebut bisa jadi merupakan sebuah fenomena yang memang sengaja diciptakan untuk mendorong para pengembang aplikasi untuk terus meningkatkan daya saing aplikasi mereka. Maka dari itu, tidak heran apabila tiap aplikasi yang ada didalamnya melakukan pembaruan aplikasi untuk meningkatkan stabilitas, kemampuan ataupun fitur yang nantinya akan meningkatkan daya saing aplikasi tersebut. Sebuah studi pada tahun 2014 mengenai pengembangan aplikasi *mobile* m enyebutkan

bahwa pada 100 pengembang aplikasi *mobile*, rata-rata dari mereka membutuhkan kurang lebih 18 minggu untuk menyelesaikan satu buah aplikasi native mereka pada platform iOS, Android maupun HTML5[3]. Tidak berhenti disitu, setelah sukses menyelesaikan sebuah aplikasi, pengembang aplikasi yang sukses rata-rata akan memulai siklus untuk selalu memutakhirkan aplikasinya dengan jumlah pembaruan berkisar antara 1 hingga 4 kali dalam sebulan[4].

Tentu hal ini akan menjadi tantangan terbesar bagi seorang pengembang aplikasi karena siklus pengembangan ini seakan tidak ada habisnya. Seorang pengembang aplikasi diharapkan mampu untuk menulis kode aplikasi yang baik, menjamin semua fitur didalamnya bebas bug, dan selalu memperbarui aplikasinya berdasarkan kebutuhan konsumen untuk tetap mempertahankan daya saing yang tinggi pada aplikasi mereka. Berbagai macam kebutuhan diatas memerlukan skill programming yang tinggi, yang tentunya bergantung pada diri masing-masing pengembang aplikasi. Namun begitu, ada beberapa aspek pendukung yang mampu memudahkan para pengembang aplikasi untuk mencapai tujuan tersebut.

Maraknya pengembangan aplikasi *mobile* dengan berbagai pendekatan termasuk kedalamnya. Teknologi WebView atau Web Apps disinyalir memudahkan pengembang untuk menghemat usaha pembangunan aplikasi. Para pengembang aplikasi pada mulanya akan dihadapkan pada pilihan yang cukup menentukan nasib aplikasi kedepan, yakni memilih untuk mengembangkan aplikasi berbasis Native atau memilih untuk mengimplementasikan teknologi WebView. Pada studi kasus yang akan diteliti, penulis akan memilih untuk mengembangkan aplikasi secara Native, karena didasarkan pada penelitian sebelumnya yang menunjukkan bahwa sebanyak 69% dari total 328 pengujian terkait performa, aplikasi yang dikembangkan secara *native* lebih unggul[5].

Demi mendukung aspek *maintainability*[6] dan *testability* [7], terdapat beberapa *Software Design pattern* yang dapat digunakan oleh para pengembang aplikasi *mobile*. *Design pattern* biasa digunakan ketika aplikasi diproyeksikan untuk skala menengah hingga besar, bukan untuk aplikasi dengan skala kecil karena hanya akan menambah kompleksitas aplikasi yang dirasa tidak perlu [7]. Pemilihan untuk menggunakan *Software Design pattern* atau tidak menggunakannya juga merupakan tantangan tersendiri. Beberapa *design pattern* yang ada menawarkan kelebihan dan kekurangannya masing-masing. Pemilihan *design pattern* yang tepat diharapkan akan menunjang proses pengembangan aplikasi. Selain itu, keputusan untuk tidak menggunakan sebuah *design pattern* bisa jadi juga tepat adanya.

Saat ini terdapat banyak sekali *Software Design pattern* yang dapat kita pilih untuk digunakan, seperti yang tercantum pada halaman ini[8]. Dari beberapa *design pattern* yang ada, penulis tertarik pada sebuah *design pattern* Model-View-ViewModel atau yang sering disebut dengan MVVM. Model-View-ViewModel(MVVM) adalah salah satu *Architectural software design pattern* yang dirancang untuk mengembangkan *platform* antar muka yang mutakhir dimana View-nya ialah kewajiban dari seorang *graphic designer*, bukan lagi kewajiban dari seorang pengembang. MVVM mempunyai tiga komponen utama, seperti yang tertera pada namanya, yakni Model, View dan ViewModel.

Komponen view merupakan komponen yang menampilkan antar muka dari aplikasi. Pada MVVM, diharapkan *designer* akan termudahkan karena adanya pemisahan yang tegas, sehingga *designer* bisa mengimplementasikan karyanya tanpa harus menunggu diimplementasikan oleh pengembang kode sebuah aplikasi. Komponen Model merupakan komponen yang merepresentasikan data, sama seperti pada arsitektur pada umumnya. Sedangkan komponen View-Model, yang berarti *User interface's model*, ditujukan untuk mengatur keadaan atau

menegaskan keberadaan View. Komponen ini akan meneruskan data dan operasi lainnya kepada komponen View dan juga mengatur logika dan perilaku yang ada pada View[9].

Setelah itu, dilakukan *testing* terhadap aplikasi yang telah dibangun. Unit test bertujuan untuk memastikan seluruh kode program dapat berjalan sesuai alur yang diharapkan dan hasil dari seluruh skenario pengujian sesuai dengan ekspektasi [10]. Hal tersebut dilakukan untuk mengetahui bahwa aplikasi telah siap untuk digunakan.

Dengan demikian, pengerjaan tugas akhir ini diharapkan dapat menjadi referensi bagi setiap pengembang aplikasi *mobile* berbasis android untuk menentukan apakah sebuah *design pattern* memang diperlukan untuk pembangunan aplikasi *mobile* skala menengah.

1.2 Rumusan Masalah

Merujuk pada latar belakang yang telah dikemukakan sebelumnya, maka rumusan masalah pada tugas akhir ini adalah sebagai berikut :

1. Bagaimana membangun aplikasi *mobile* berbasis Android dengan menggunakan *design pattern* MVVM?
2. Bagaimana menganalisa aplikasi *mobile* berbasis Android dengan menggunakan *software metrics tools* JHawk 6?
3. Bagaimana mengetahui signifikansi penggunaan *design pattern* MVVM dalam tujuan meningkatkan aspek *maintainability*?

1.3 Batasan Masalah

Merujuk pada latar belakang yang telah disebutkan diatas, Batasan masalah dalam tugas akhir ini adalah :

1. Aplikasi yang digunakan untuk penelitian dirancang secara spesifik untuk proses evaluasi dan pengiriman data pajak bangunan pada *client-side*.

2. Aplikasi dibangun untuk digunakan pada sistem operasi Android.
3. *Design pattern* yang akan digunakan adalah MVVM.
4. Pengukuran dan penilaian menggunakan empat metrik utama yang telah disediakan oleh JHawk 6 yakni LOC, Halstead Effort, Cyclomatic Complexity yang bisa membentuk maintainability index.

Dengan demikian, selesainya tugas akhir ini adalah terbentuknya dua aplikasi *client-side* yang sudah teruji pada tahapan pengujian dengan *design pattern* yang berbeda untuk kemudian dibandingkan dalam empat metrik utama yang sudah disiapkan pada *software metrics tools* JHawk 6.

1.4 Tujuan Penelitian

Berdasarkan latar belakang dan rumusan masalah yang telah disebutkan sebelumnya, adapun tujuan dari penelitian tugas akhir ini adalah mendapatkan keputusan yang tepat tentang penggunaan *design pattern* tertentu pada kasus aplikasi tingkat menengah pada penelitian ini.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dapat diperoleh dari tugas akhir ini adalah sebagai berikut :

1. Mendukung penelitian tentang penggunaan *design pattern* yang sesuai dan tepat guna.
2. Memberikan tambahan wawasan dan menyediakan referensi dalam pengembangan aplikasi *mobile* yang *sustainable*.

1.6 Relevansi

Tugas akhir ini disusun untuk memenuhi salah satu syarat kelulusan sebagai Sarjana Komputer di Departemen Sistem Informasi Fakultas Teknologi Informasi dan Komunikasi (FTIK) Institut Teknologi Sepuluh Nopember (ITS). Tugas akhir ini sesuai dengan penerapan mata kuliah dari laboratorium

Infrastruktur dan Keamanan Teknologi Informasi (IKTI) Departemen Sistem Informasi FTIK ITS, yaitu Teknologi Open Source Terbaru dan Teknologi Bergerak. Selain itu, juga berkaitan dengan mata kuliah wajib, yaitu Pemrograman Berbasis Objek serta Konstruksi dan Pengujian Perangkat Lunak.

BAB II
TINJAUAN PUSTAKA

Pada bagian ini akan dijelaskan mengenai penelitian sebelumnya dan dasar teori yang dijadikan acuan atau landasan dalam pengerjaan tugas akhir ini.

2.1 Penelitian Sebelumnya

Penelitian yang dijadikan acuan dalam pengerjaan tugas akhir ini disajikan pada tabel 2.1 sebagai berikut :

Tabel 2.1 Penelitian Sebelumnya

1. A Comparison of Android Native App Architecture - MVC, MVP and MVVM
Penulis, Tahun : Tian Lou; 2016 [11]
Hasil : Berbagai macam arsitektur dan *design pattern* yang ada menimbulkan banyak pertanyaan mengenai apa *design pattern* terbaik yang harus dipakai untuk meningkatkan kualitas perangkat lunak. Untuk itu, sebuah thesis diterbitkan yang membahas dan membandingkan berbagai macam *design pattern* yang umum yang sering digunakan oleh khayalak umum dengan tujuan sebagai acuan memilih *design pattern* yang tepat bagi mereka.
Keterkaitan : Penelitian tugas akhir yang dilakukan ini berkaitan dengan penggunaan berbagai macam *design pattern*. Adapun *design pattern* yang akan digunakan pada penelitian ini adalah MVVM.
2. Impact of Design patterns on Software Maintainability
Penulis, Tahun :

Fatimah Mohammed Alghamdi dan M. Rizwan Jameel Qureshi; 2012 [12]

Hasil :
Penggunaan *design pattern* pada perangkat lunak didasari pada perubahan yang sering terjadi pada proses pengembangan perangkat lunak. *Design pattern* diharapkan dapat memfasilitasi perubahan tersebut dengan mudah. Hasil survei membuktikan bahwa penerapan *design pattern* yang digunakan harus sesuai dengan kebutuhan sistem, hal tersebut untuk memudahkan *maintainance* dan proses pengembangan dikemudian hari.

Keterkaitan :
Penelitian tugas akhir yang dilakukan berkaitan dengan pengunaan *design pattern* untuk meningkatkan aspek *maintainability*.

3. Performance analysis of a software developed with and without design patterns: A case study

Penulis, Tahun :
Emre Kazan., Mehmet Canturk., Muhammet Bastan; 2016 [7]

Hasil :
Penggunaan *software design pattern* yang tepat akan meningkatkan aspek performa dari sebuah perangkat lunak, tepatnya pada bagian *testability*. Dengan menggunakan *software design pattern*, arsitektur kode program menjadi lebih rapi dan hal tersebut membantu mendukung pengujian yang bisa dilakukan pada sebuah kode program. Selain itu, perubahan-perubahan yang mungkin terjadi ketika *software* telah dirilis menjadi lebih mudah untuk diimplementasikan.

Keterkaitan :
Penelitian pada tugas akhir ini mempunyai tujuan untuk membangun aplikasi *mobile* yang unggul bukan hanya pada sisi pengembang saja, namun juga unggul pada sisi *user* atau pengguna dengan cara meningkatkan performa dari program itu sendiri.

4. A Tale of Two Fashions: An Empirical Study on the Performance of Native Apps and Web Apps on Android

Penulis, Tahun : Yun Ma., Xuanzhe Liu., Yi Liu., Yunxin Liu., dan Gang Huang; 2017 [5]
Hasil : Kurang lebih sebanyak 69% dari total 328 kasus pengembangan aplikasi *native* dan aplikasi berbentuk *Web Apps* pada sistem operasi android menunjukkan bahwa aplikasi *native* mempunyai lebih banyak kelebihan daripada aplikasi berbentuk *web apps* dari sisi pengembang maupun dari sisi *user*. Walaupun *web apps* mempunyai kelebihannya sendiri, namun hal tersebut tidak serta merta menutupi kelemahan dari *web apps* yang dirasa lebih banyak daripada kelebihannya.
Keterkaitan : Penelitian tugas akhir ini akan dikembangkan dengan teknologi *native* android untuk kualitas perangkat lunak yang lebih baik, dan tidak menggunakan teknologi *web apps*.

2.2 Dasar Teori

Berikut merupakan teori-teori yang mendukung serta berkaitan dalam pengerjaan tugas akhir.

2.2.1 Software Quality

IEEE mendefinisikan *Software Quality* sebagai komposisi dari karakteristik dan atribut pada *software* untuk memenuhi keinginan *stakeholder*[13]. Pada *Software Requirement Engineering*, *Software Quality* juga dikategorikan sebagai *non-functional requirement*.

Software Quality bisa dibagi menjadi dua kategori: *Development Reqirement* dan *Operational Requirement. Development Requirement* atau kebutuhan pengembangan aplikasi merupakan perspektif dari pengembang aplikasi dimana hal tersebut merupakan kebutuhan yang seharusnya memudahkan untuk kegiatan pengembangan aplikasi, seperti contoh yakni *maintainability* dan *understandability*. Sedangkan *operational requirements* atau kebutuhan operasional

merupakan kebutuhan yang penting dipenuhi bagi kenyamanan pengguna, seperti contoh *usability* dan *performance*.

Software Quality ditentukan oleh banyak faktor, dimana salah satunya yakni *Software Architecture* adalah yang paling penting. Medvidovic dan Taylor mengklaim bahwa *software quality* yang baik tentu mempunyai korelasi dengan *software architecture* yang baik pula [14]. Chen dan kawan-kawan mendeskripsikan *software quality* sebagai *Architectural Significant Requirements* dimana hal tersebut membuktikan bahwa arsitektur aplikasi mempunyai pengaruh yang besar kepada *software quality* [15].

2.2.2 *Architectural Design pattern*

Architectural Design pattern merupakan salah satu dari sekian banyak *Software Design pattern* yang memiliki cakupan yang lebih luas. *Architectural Design pattern* menangani isu tentang berbagai macam permasalahan pada *computer engineering*. Khusus untuk masalah pembangunan perangkat lunak, terdapat tiga *Architectural Software Design pattern* yang sering digunakan yakni MVC, MVP dan MVVM. *Software Design pattern* sendiri dikenal sebagai sebuah solusi umum untuk mengatasi beberapa permasalahan yang terjadi pada dunia *software engineering*. *Design pattern* menyediakan sebuah struktur template yang dapat digunakan dan disesuaikan dengan kondisi yang ada [16].

Design pattern mempunyai 4 komponen utama yakni: nama *pattern*, permasalahan, solusi dan konsekuensinya yang akan menjadi topik utama pada penelitian ini. Nama *pattern* sering mendeskripsikan secara singkat tentang problem desainnya. Nama *pattern* juga lazim merepresentasikan gagasan utama dari sebuah desain, sehingga memudahkan komunikasi dan transfer ilmu antar pengembang aplikasi. *Design pattern* juga seringkali dapat merepresentrasikan konteks serta kapan seorang pengembang aplikasi disarankan untuk menggunakannya [17].

Design pattern dapat dikategorikan menjadi 3, yakni:

- *Creational pattern* berkaitan dengan bagaimana membuat objek.
- *Structural pattern* berkaitan dengan simplifikasi dari realisasi hubungan antar objek/entitas.
- *Behavioral pattern* berkaitan dengan bagaimana memberikan objek suatu tanggung jawab.

Karena terdapat beberapa karakteristik dari arsitektur yang *scalable* adalah *high cohession* dan *loose coupling*. *High cohession* berarti komponen dari perangkat lunak harus fokus dalam mengerjakan hanya satu hal. Sedangkan *loose coupling* berarti komponen perangkat lunak tidak harus tahu mengenai internal antar komponen satu dengan lainnya [17].

2.2.3 Android

Sistem operasi Android sebagai salah satu dari sekian banyak sistem operasi berbasis *mobile* dapat dikatakan merupakan sistem operasi tersukses dilihat dari jumlah penggunanya dibandingkan dengan sistem operasi yang lain. Menurut data yang dibawakan oleh StatCounter, terhitung pada bulan Januari tahun 2019, Android mendapatkan pangsa pasar sebesar 74.45%, unggul jauh dari kompetitor terdekatnya yakni Apple iOS yang "hanya" mendapatkan pangsa pasar tidak lebih dari 25%, atau lebih tepatnya sebesar 22.85%[1]. Tidak hanya itu, apabila dilihat dari jumlah aplikasi yang beredar, Google Play Store yang merupakan sumber utama peredaran aplikasi pada OS Android tercatat akhir-akhir ini telah mempunyai setidaknya 2.9 juta aplikasi didalamnya, unggul dari pesaing terdekatnya yakni iOS App Store yang mempunyai 1.85 juta aplikasi[2].

Android adalah sistem operasi berbasis Linux yang dirancang untuk perangkat bergerak layar sentuh seperti telepon pintar dan komputer tablet. Android awalnya dikembangkan oleh Android, Inc., dengan dukungan finansial dari Google, yang kemudian membelinya pada tahun 2005. Sistem operasi

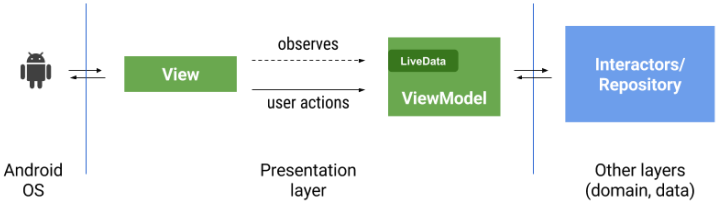
ini dirilis secara resmi pada tahun 2007, bersamaan dengan didirikannya Open Handset Alliance, konsorsium dari perusahaan-perusahaan perangkat keras, perangkat lunak, dan telekomunikasi yang bertujuan untuk memajukan standar terbuka perangkat seluler. Ponsel Android pertama mulai dijual pada bulan Oktober 2008 [18].

Android adalah sistem operasi dengan sumber terbuka, dan Google merilis kodenya di bawah Lisensi Apache. Kode dengan sumber terbuka dan lisensi perizinan pada Android memungkinkan perangkat lunak untuk dimodifikasi secara bebas dan didistribusikan oleh para pembuat perangkat, operator nirkabel, dan pengembang aplikasi [19].

2.2.4 Design Pattern MVVM

"Arsitektur Model / View / View Model (MVVM) adalah arsitektur yang dirancang untuk platform pengembangan UI modern di mana View adalah tanggung jawab seorang desainer." [20].

MVVM memiliki tiga komponen utama seperti yang tercantum pada namanya, Model, View dan view-model. Komponen View merupakan UI aplikasi. Dalam MVVM, UI dirancang untuk lebih *designer-friendly* yang dapat dengan mudah dikerjakan oleh desainer daripada oleh pengembang kode. Model mewakili data, sama seperti model yang dijelaskan dalam arsitektur MVC maupun MVP. View-Model, yang merupakan modelnya view, dimaksudkan untuk mengelola data pada UI. View-model akan meneruskan data dan operasi untuk melihat dan juga mengelola logika dan perilaku tampilan[9]. Komponen View-Model yang baik harus mampu mengolah keberadaan data dengan memperhatikan originalitas data tersebut, tanpa perlu memikirkan bagaimana data nantinya ditampilkan atau dikirimkan[9]. Intinya adalah semua yang berhubungan dengan UI bukan lagi merupakan tanggung jawab pengembang utama aplikasi, namun merupakan tanggung jawab desainer.



Gambar 2.1 Tipikal interaksi antar komponen pada *design pattern* MVVM

Hal yang menjadikan ViewModel spesial adalah tentang bagaimana *design pattern* ini bertindak dalam mengurus data. Dalam hal ini, ViewModel mempunyai ciri-ciri yang dinamakan *Observer Pattern*, sebuah panduan dalam kode program yang bertujuan untuk menciptakan *broadcaster* data dan *observer* data. Mirip seperti koneksi pada pemutar radio musik, ada pihak yang berkewajiban untuk menyebarkan data atau istilahnya adalah broadcaster, dan ada pihak yang berfungsi sebagai penerima data, atau istilahnya adalah observer data. Dengan diimplementasikannya MVVM, otomatis *observer pattern* juga wajib untuk diimplementasikan.

Implementasi *observer pattern* membutuhkan komponen lain untuk dapat berjalan dengan baik, salah satu contohnya adalah komponen *LiveData*. *LiveData* adalah sebuah *class* yang berfungsi sebagai penyimpan data dan berkewajiban untuk selalu dapat dihubungi ketika diperlukan, atau bisa disebut sebagai broadcaster data[21]. Dengan implementasi MVVM, diharapkan terdapat separasi yang jelas antara tanggung jawab pengembang aplikasi dengan desainer aplikasi, sehingga dapat meningkatkan produktivitas dan maintainabilitas aplikasi.

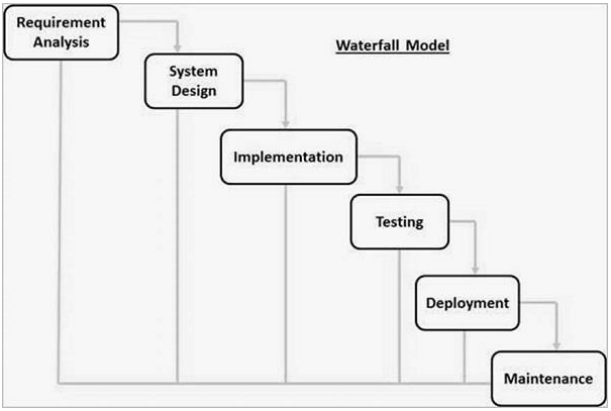
2.2.5 Software Development Life Cycle

Software Development Life Cycle (SDLC) adalah metodologi atau pedoman yang digunakan untuk merancang, mengembangkan, dan menguji suatu perangkat lunak [22]. SDLC bertujuan untuk menghasilkan perangkat lunak berkualitas yang sesuai dengan spesifikasi. SDLC terdiri dari

informasi yang menjelaskan cara mengembangkan, memelihara, mengganti, dan mengubah atau meningkatkan perangkat lunak tertentu [22]. Ada berbagai model SDLC yang dapat digunakan untuk memastikan keberhasilan dalam proses pengembangan perangkat lunak, seperti model *waterfall*, model *spiral*, model *agile*, model *prototype*, dan sebagainya.

- Model Waterfall

Model *waterfall* sebagai model dengan siklus berurutan linier yang mudah dimengerti dan digunakan [23]. Dalam model *waterfall*, seluruh proses pengembangan perangkat lunak dibagi menjadi fase yang terpisah. Hasil dari satu fase akan digunakan sebagai input untuk fase berikutnya secara berurutan, sehingga setiap fase harus diselesaikan sebelum memulai fase berikutnya [23]. Ilustrasi berikut adalah representasi dari berbagai fase model *waterfall*.



Gambar 2.2 Ilustrasi Model Waterfall

2.2.6 Unified Modeling Language

Unified Modeling Language (UML) dibuat untuk membantu menentukan, memvisualisasikan, dan mendokumentasikan sistem perangkat lunak, termasuk struktur, dan desainnya sesuai dengan spesifikasi [24]. UML dapat memodelkan hampir semua jenis perangkat lunak dengan berbagai bahasa

pemrograman. Fleksibilitasnya memungkinkan untuk memodelkan perangkat lunak yang dibangun secara object oriented dengan menyeluruh dari berbagai sisi. Untuk itu, UML menyediakan 13 tipe diagram yang dapat digunakan sesuai dengan kebutuhan pengembangan perangkat lunak, diagram-diagram tersebut dibagi menjadi 2 elemen yang berbeda [25], yaitu :

- *Structure diagrams* mewakili aspek struktural dari suatu sistem, adapun diagram yang termasuk kategori ini, yaitu : *class* diagram, component diagram, composite structure diagram, deployment diagram, object diagram, dan package diagram.
- *Behavior diagrams* mewakili aspek dinamis dari suatu sistem, adapun diagram yang termasuk kategori ini, yaitu : activity diagram, communication diagram, interaction overview diagram, sequence diagram, state machine diagram, timing diagram, dan use case diagram

2.2.7 JHawk 6 Software Metrics Tools

Dikutip dari laman resmi JHawk[26], *Software metrics* adalah sebuah satuan pengukuran software atau perangkat lunak untuk kepentingan proses pengembangan aplikasi. Pengukuran seperti ini adalah proses yang vital, dimana jika kita melewatkannya maka akan terdapat kesulitan, bahkan dapat membuat ketidaktahuan mengenai adanya kesalahan atau bug dalam perangkat lunak yang sedang kita kembangkan. Selain itu, kita juga akan melewatkan kesempatan untuk mengukur apakah perangkat lunak yang sedang dikembangkan tersebut akan mampu untuk dikembangkan kembali menjadi lebih baik nantinya. Pengukuran terhadap perangkat lunak tersebut seringkali terkait pada beberapa hal, yaitu:

- Banyaknya waktu untuk mengembangkan sebuah perangkat lunak
- Banyaknya perangkat lunak yang bisa dikembangkan

- Banyaknya waktu untuk memelihara perangkat lunak
- Jumlah kesalahan yang ada
- Kualitas perangkat lunak.

Pada JHawk 6, semua aspek penting yang diperlukan untuk mengukur sebuah software telah tersedia didalamnya untuk digunakan. JHawk sendiri sudah sering digunakan dalam beberapa penelitian yang memerlukan data tentang kualitas sebuah perangkat lunak, seperti pada beberapa paper ini [18], [27], [28].

2.2.8 Testing

Testing adalah proses mengevaluasi sistem atau komponennya dengan maksud untuk mengetahui bahwa sistem sudah memenuhi persyaratan yang ditentukan atau tidak [29]. *Testing* juga berfungsi untuk mengidentifikasi kesenjangan, kesalahan, atau ketentuan yang hilang yang bertentangan dengan ketentuan yang sebenarnya. Terdapat berbagai metode *testing* yang dapat digunakan dalam pengembangan perangkat lunak, salah satunya metode *testing* yang paling umum adalah Unit Testing.

- Unit Testing

Unit testing merupakan salah satu teknik pengujian yang fokus kepada modul individu untuk diuji dan untuk menentukan, apakah ada masalah dalam pengembangan perangkat lunak yang diuji tersebut [28]. Hal ini berkaitan dengan penggunaan secara fungsional dari modul itu sendiri. Tujuan utamanya adalah mengisolasi setiap unit pada sistem untuk mengidentifikasi, menganalisis, dan memperbaiki kesalahan yang ditemukan.

Unit testing digunakan untuk mengurangi kesalahan pada fitur yang baru dikembangkan atau mengurangi bug saat mengubah fungsi yang ada. *Unit testing* juga dapat memperbaiki desain dan memungkinkan *refactoring* kode program yang lebih baik [28].

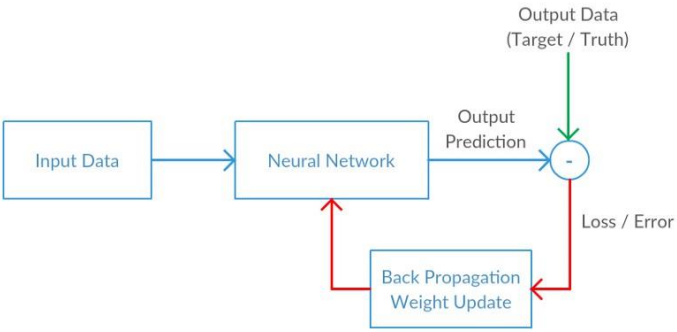
2.2.9 PajakCerdas

PajakCerdas merupakan sebuah aplikasi berbasis android yang proses bisnisnya didesain secara khusus untuk memenuhi kebutuhan stakeholder yang menginginkan pemrosesan data pajak bangunan dengan cepat. Pada umumnya, pemrosesan data pajak memerlukan beberapa langkah sebelum data tersebut siap untuk diolah menjadi informasi yang berguna bagi para *stakeholder*. Dengan menggunakan aplikasi PajakCerdas ini, pengolahan data akan dimudahkan dengan cara mempercepat pengiriman data-data yang diperlukan dengan tiga komponen utama yang tersedia pada aplikasi ini, yakni :

- Masukan, terdiri dari beberapa variabel pendukung dan foto objek pajak
- Proses, terdapat rumus perhitungan pajak dan implementasi *machine learning framework* Tensorflow
- Luaran, yang menghasilkan 3 nilai utama yakni Nilai Jual Objek Pajak, Nilai Jual Kena Pajak dan terakhir adalah Nilai Pajak Bumi dan Bangunan

2.2.10 Tensorflow

Tensorflow(Lite Release 1.12.3) adalah *library* perangkat lunak yang bersifat *open source* yang dapat digunakan untuk pengolahan dan perhitungan data numerik menggunakan *data flow graphs*. Dengan kata lain, Tensorflow adalah kumpulan *tools* untuk pengolahan data pada bidang *machine learning*[30]. Penggunaan *data flow graphs* terbukti sesuai untuk diimplementasikan pada *machine learning* berbasis *neural network*[31]. *Neural network* adalah model yang terinspirasi oleh bagaimana neuron dalam otak manusia bekerja. Tiap neuron pada otak manusia saling berhubungan dan informasi mengalir dari setiap neuron tersebut.



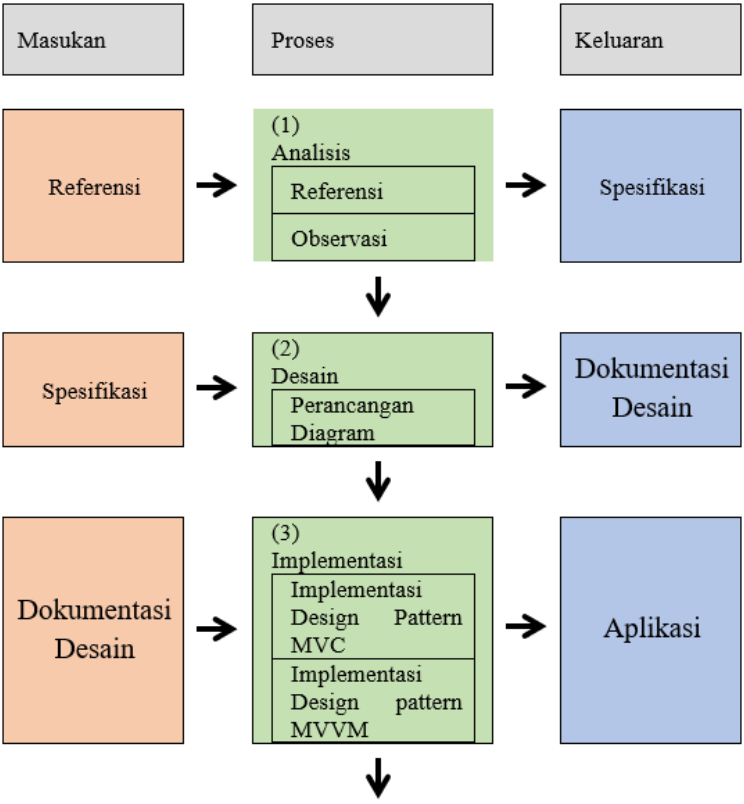
Gambar 2.3 Ilustrasi Neural Network, back propagation

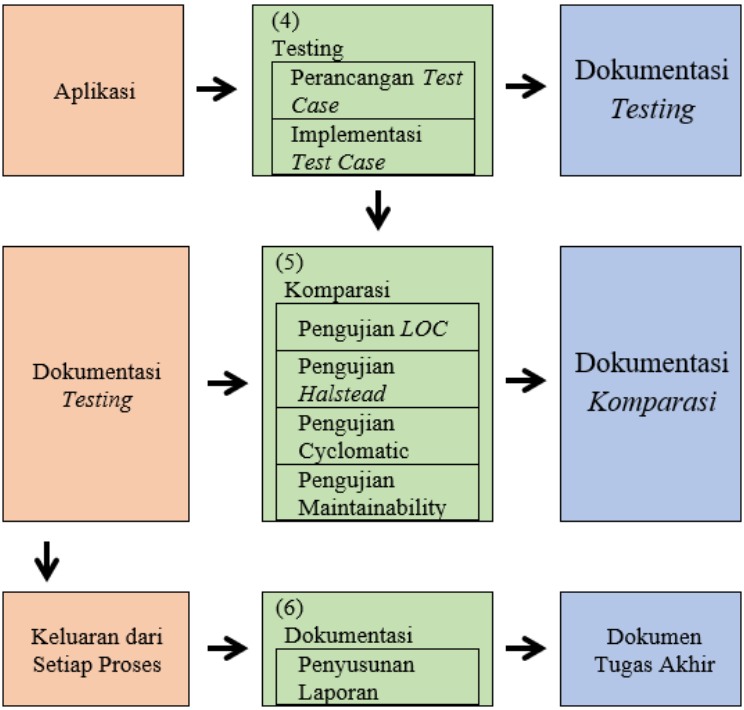
Ketika aliran informasi tersebut sangat banyak dan kompleks, tentu kita membutuhkan alat bantu untuk memecahkan permasalahan tersebut. Disitulah Tensorflow banyak digunakan karena dapat membantu pengembang untuk memecahkan masalah *machine learning* dengan bantuan kekuatan komputasi CPU dan GPU.

Tensorflow terbagi menjadi tiga komponen: TensorFlow API, TensorBoard, dan TensorFlow Serving. Mendefinisikan, melatih dan memvalidasi model machine learning dapat dilakukan dengan menggunakan Tensorflow API. API yang disediakan berbentuk Bahasa C++, namun antarmuka yang menggunakan Bahasa Python juga disediakan. TensorBoard bertugas untuk analisa, *debugging* dan visualisasi model *data flow graph.* Terakhir adalah Tensorflow Serving yang bertugas untuk memungkinkan pembuatan model *machine learning*[31]. Pada kasus penelitian ini, Tensorflow API yang akan digunakan. Untuk penjelasan lebih lanjut, dapat mengakses situs tensorflow.org

BAB III
METODE PENELITIAN

Pada bab ini akan dijelaskan mengenai tahapan ysang dilakukan dalam pengerjaan tugas akhir menggunakan SDLC Waterfall yang telah disesuaikan beserta deskripsi dan penjelasannya.





Gambar 3.1 Metodologi Pengerjaan Tugas Akhir

3.1 Tahap Analisis Kebutuhan

Pada tahap ini akan dilakukan penggalian kebutuhan mengenai aplikasi *mobile* berbasis android yang akan dibuat. Penggalian kebutuhan nantinya akan menjadi data penunjang pada tahapan penelitian berikutnya.

3.1.1. Referensi

Merupakan *input* pertama dari dimulainya pengerjaan tugas akhir. Referensi serta penelitian sebelumnya yang dimaksud merupakan pikiran dan hasil penelitian terdahulu yang dihimpun sebagai landasan berjalannya pengerjaan tugas akhir, ditambah referensi kode program *existing* yang sejenis untuk dimanfaatkan pada kegiatan observasi.

3.1.2. Observasi

Pada tahapan ini akan dilakukan pengamatan pada kebutuhan pengguna akan aplikasi pada tema perpajakan. Pengamatan digunakan untuk mendapatkan gambaran seperti apa aplikasi akan dibuat nantinya, termasuk menuliskan *User Requirements* yang sesuai.

3.1.3. Tujuan dan Spesifikasi Tugas Akhir

Spesifikasi tugas akhir merupakan *output* yang didapat dari tahap analisis kebutuhan. Entitas ini akan mendefinisikan *scope* pengembangan tugas akhir serta teknologi yang dipakai dalam pengembangan produk tugas akhir. Spesifikasi tugas akhir akan digunakan sebagai *input* dalam proses pertama pada tahap desain sistem.

3.2 Tahap Desain

Pada tahap ini dilakukan perancangan dan desain aplikasi *mobile* berbasis android PajakCerdas dengan *design pattern* MVVM dan dengan *design pattern* MVC. Perancangan dan desain menggunakan Unified Modeling Language (UML).

3.2.1 Perancangan Diagram

Pada tahapan ini akan dilakukan pembuatan diagram-diagram untuk pengembangan aplikasi PajakCerdas sesuai kebutuhan pengguna yang didapatkan dari tahapan sebelumnya.

3.2.2 Dokumentasi Desain

Entitas ini merupakan *output* dari tahap desain sistem yang akan digunakan dalam proses pertama pada tahap implementasi.

3.3 Tahap Implementasi

Implementasi merupakan tahapan pembuatan kode program pada Integrated Development Environment(IDE) Android Studio berdasarkan desain yang telah dibuat untuk *design pattern* MVVM dan *design pattern* MVC. Kode program yang

dibuat nantinya akan digunakan pada tahap komparasi dan akan saling dibandingkan satu sama lain.

3.3.1 Implementasi Design Pattern MVC

Pada tahapan ini akan dilakukan implementasi pembuatan aplikasi sesuai *usecase* aplikasi yang menggunakan *design pattern* MVC. Implementasi dilakukan dengan cara penulisan kode program yang sesuai dengan spesifikasi dan usecase untuk memenuhi kebutuhan fungsional dan non fungsional aplikasi. Kode program yang ditulis dan diimplementasikan pada aplikasi PajakCerdas juga mengacu pada penerapan *design pattern* MVC untuk kemudian diteliti pengaruh penerapan *design pattern* tersebut.

3.3.2 Implementasi Design Pattern MVVM

Pada tahapan ini dilakukan pembuatan aplikasi sesuai *usecase* aplikasi yang menggunakan *design pattern* MVVM yang telah dibuat. Implementasi dilakukan dengan cara penulisan kode program yang sesuai dengan spesifikasi dan usecase untuk memenuhi kebutuhan fungsional dan non fungsional aplikasi. Kode program yang ditulis dan diimplementasikan pada aplikasi PajakCerdas juga mengacu pada penerapan *design pattern* MVVM untuk kemudian diteliti pengaruh penerapan *design pattern* tersebut.

3.4 Tahap Testing

Pada tahapan ini, akan dilakukan *testing* terhadap aplikasi yang telah dibangun dengan metode unit testing.

3.4.1 Perancangan *Test Case*

Pada tahapan ini akan dirancang skenario *testing* untuk dilakukan pada tahapan berikutnya. Adapun perencanaan skenario berdasarkan *functional requirement* aplikasi.

3.4.2 Implementasi *Test Case*

Setelah rancangan skenario *testing* siap maka akan dijalankan unit testing dengan menggunakan fitur *testing* yang telah tersedia.

3.4.3 Dokumentasi *Testing*

Entitas ini merupakan *output* dari tahap *testing* yang berisi hasil unit testing yang dilakukan secara mendetail dari keseluruhan *testing* yang dilakukan.

3.5 Tahap Komparasi

Pada tahapan ini, dilakukan perbandingan antara aplikasi yang dibangun dengan *design pattern* MVVM dan aplikasi yang dibangun menggunakan *design pattern* MVC. Pada tahap ini, output yang diharapkan ialah data mengenai beberapa aspek penting terkait justifikasi pemilihan *design pattern* yang sudah terdapat pada *software metrics tools* JHawk 6.

Pada JHawk 6, terdapat beberapa metrik seperti yang sudah tertulis pada bab sebelumnya. Metrik-metrik tersebut nantinya akan digunakan untuk mengukur kedua aplikasi yang memiliki pendekatan berbeda pada saat tahap implementasinya. Ketika kedua kode program telah siap untuk diuji, keduanya akan dimasukkan kedalam JHawk 6 untuk kemudian akan dianalisis dengan menggunakan metrik-metrik yang telah disebutkan sebelumnya. Pengujian yang ada pada JHawk 6 tercantum dibawah ini :

3.5.1 Pengujian LOC(*Line of Code*)

Pada tahapan ini, kode dari kedua program akan dibandingkan. Perbandingan yang dicari disini adalah banyaknya *statement* kode yang dibutuhkan untuk membuat masing-masing aplikasi.

3.5.2 Pengujian *Halstead Effort*

Pada tahapan ini, pengujian halstead effort akan dilakukan pada kedua kode program. *Halstead effort* sendiri merupakan salah satu metode pengukuran yang dikembangkan oleh Maurice

Halstead untuk menentukan seberapa besar usaha yang dibutuhkan untuk membangun sebuah program aplikasi [32].

3.5.3 Pengujian *Cyclomatic Complexity*

Pada tahapan ini, pengujian cyclomatic complexity akan dilakukan juga pada kedua kode program aplikasi. *Cyclomatic complexity* merupakan sebuah pengukuran yang dikembangkan oleh Thomas McCabe untuk mengukur kompleksitas kode program [32].

Alasan dibalik penggunaan metrik diatas dapat disimpulkan pada rumus dibawah[33] :

$$MI=171-3.42ln(aveE) - 0.23aveV(g') -16.2ln(aveLOC)$$

Gambar 3.2 Rumus penentuan Maintainability Index

Dimana :

aveE = rata-rata nilai *Halstead Effort* yang bisa didapatkan dari persamaan dibawah :

$$E = D \times V$$

D = Tingkat kesulitan untuk membangun kode program
V = Volume perhitungan *Line of Code*.

aveV(g') = rata-rata nilai *Cyclomatic Complexity* per modul
aveLOC = rata-rata *Line of Code* per modul

3.6 Tahap Dokumentasi

Pada tahapan ini, laporan tugas akhir akan dibuat yang akan mendokumentasikan setiap langkah yang telah dilakukan, hasil yang dihasilkan setiap langkah, kesimpulan serta saran untuk penelitian kedepannya.

3.6.1 Penyusunan Buku Tugas Akhir

Pada tahapan terakhir ini akan dilakukan pembuatan laporan dalam bentuk buku tugas akhir yang disusun sesuai format yang telah ditentukan. Buku ini berisi dokumentasi langkah-langkah pengerjaan tugas akhir secara rinci. Buku ini diharapkan dapat bermanfaat sebagai referensi untuk pengerjaan penelitian lain, serta sebagai acuan untuk pengembangan lebih lanjut terhadap topik pengembangan yang serupa.

3.6.2 Dokumen Tugas Akhir

Adapun dokumen tugas akhir merupakan *output* dari keseluruhan proses penelitian yang telah dilakukan sesuai metodologi.

Halaman ini sengaja dikosongkan

BAB IV
PERANCANGAN

Bab ini menjelasakan mengenai perancangan dari desain hingga dihasilkannya luaran tugas akhir.

4.1 Analisis Kebutuhan

Tahap awal dari perancangan dimulai dari mengumpulkan informasi mengenai kebutuhan dan proses bisnis pengambilan data pajak suatu objek kena pajak. Pengambilan informasi dilakukan dengan menggunakan dua buah metode yaitu observasi dan wawancara.

4.1.1 Wawancara

Wawancara dilakukan kepada beberapa pihak yang berkepentingan dengan aplikasi ini, seperti Kaprodi Departemen Teknik Sipil ITS bapak Tri Joko Wahyu Adi, ST., MT.,Ph.D., dan Project Owner aplikasi ini ibu Lila Ayu Ratna Winanda, ST., MT. Narasumber pertama yakni ibu Lila banyak menerangkan teknis aplikasi, variabel yang dibutuhkan, dan banyak kebutuhan fungsional aplikasi yang dijelaskan secara rinci, sedangkan narasumber kedua bapak Tri Joko banyak menjelaskan mengenai konsep inti dari aplikasi PajakCerdas. Dalam wawancara, hal yang ditanyakan adalah bahasan seputar kebutuhan pengguna beserta kemungkinan pengembangan yang dapat dilakukan. Sehingga dari wawancara yang dilakukan didapat beberapa kebutuhan fungsional dan nonfungsional untuk user sebagai berikut :

1. Kebutuhan Fungsional

Tabel 4.1 Kebutuhan Fungsional Aplikasi

Kode	Deskripsi
F1	Aplikasi dapat digunakan untuk menghitung beberapa variabel tanah objek pajak
F2	Aplikasi dapat digunakan untuk menghitung beberapa variabel bangunan objek pajak
F3	Aplikasi dapat digunakan untuk mendapatkan lokasi objek pajak melalui API Google Maps
F4	Aplikasi dapat mengambil gambar atau foto objek pajak
F5	Aplikasi menyediakan fitur menyimpan kedalam database
F6	Aplikasi menyediakan fitur menampilkan data yang tersimpan pada database

2. Kebutuhan Non Fungsional

Tabel 4.2 Kebutuhan Non Fungsional Aplikasi

Kode	Deskripsi
NF1	Aplikasi memilliki tampilan yang menarik
NF2	Aplikasi memiliki performa yang baik

4.1.2 Observasi

Metode ini dilakukan dengan mengamati secara langsung mengenai kebutuhan-kebutuhan utama pengguna aplikasi atau surveyor dari pihak pelaku pajak. Dari hasil observasi, didapati

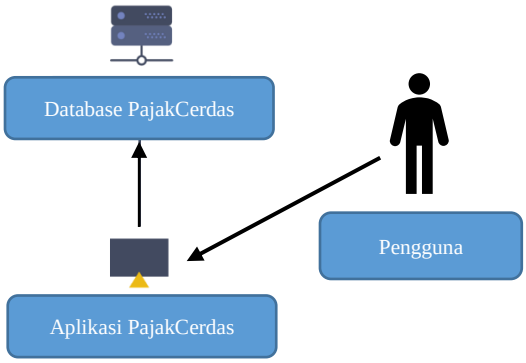
beberapa variabel yang diperlukan untuk mendukung pengambilan keputusan mengenai harga objek kena pajak. Variabel yang didapatkan tercantum pada tabel berikut.

Tabel 4.3 Variabel yang dibutuhkan

No	Variabel	Keterangan
1	Tampak objek	Foto objek yang kemudian akan diidentifikasi menggunakan bantuan machine learning untuk menentukan jenis objek, apakah mewah, sederhana, cukup atau tanah kosong
2	Jenis Bangunan	11 pilihan jenis bangunan yang bisa dipilih, yakni kantor/tempat usaha, pabrik, toko/apotik/rumahsakit/klinik, Gudang, Gedung pemerintahan, apartemen, bangunan parkir, Gedung sekolah, pompa bensin, tangki minyak.
3	Jumlah Lantai	3 pilihan dalam jumlah lantai, yakni 1 lantai, 2 lantai, dan yang terakhir adalah lebih dari 2 lantai
4	Pagar	Terdapat 2 pilihan yakni ada pagar atau tidak ada pagar
5	Garasi	Terdapat 2 pilihan yakni ada garasi atau tidak ada garasi
6	Jenis Konstruksi	Terdapat 2 pilihan yakni standar dan mutu tinggi
7	Jalan Lingkungan	Terdapat 3 pilihan yakni tanah, paving atau aspal
8	Carport	Terdapat 2 pilihan yakni ada carport atau tidak ada carport
9	Taman	Terdapat 2 pilihan yakni ada taman atau tidak ada taman

10	Jenis Tanah	Terdapat 2 pilihan yakni tanah dan bangunan atau fasum
11	Lokasi	Terdapat fitur untuk mendapatkan alamat dan lokasi objek pajak dari API yang disediakan oleh Google Maps
12	Luas Tanah	Merupakan masukan yang menampung data luas tanah dalam meter persegi
13	Luas Bangunan	Merupakan masukan yang menampung data luas bangunan dalam satuan meter persegi
14	Zona tanah	Merupakan masukan yang menampung data zona objek tanah berdasarkan data asli dari pemkot Surabaya

Dengan begitu, didapatkan beberapa variabel yang nantinya akan bisa diolah datanya untuk mendapatkan nilai-nilai pajak yang dibutuhkan seperti taksiran nilai property, Nilai Jual Kena Pajak, dan nilai Pajak Bumi dan Bangunan. Arsitektur aplikasi nantinya akan dapat dilihat pada gambar dibawah ini. 4.1.



Gambar 4.1 Diagram Arsitektur Aplikasi PajakCerdas

Dari hasil wawancara dan observasi yang dilakukan, dibuat pemetaan fungsi yang ada pada aplikasi berdasarkan kebutuhan fungsional yang telah didapat dari hasil wawancara. Adapun tabel pemetaan yang dihasilkan dapat dilihat pada tabel 4.4 dengan aktor utamanya adalah User.

Tabel 4.4 Pemetaan Kebutuhan Fungsional Aplikasi

Kode	Fungsi dalam aplikasi
F1	Hitung Lantai Pilih Jenis Bangunan Pilih Jenis Jalan Lingkungan Pilih Jenis Konstruksi Pilih Jenis Tanah Cek Pagar Cek Carport Cek Garasi Cek Taman Hitung Lebar Bangunan
F2	Hitung Luas Tanah Hitung Luas Bangunan Hitung Zona Tanah
F3	Lihat Database
F4	Ambil Gambar
F5	Ambil Lokasi
F6	Simpan Database

Untuk menghasilkan nilai luaran, aplikasi PajakCerdas membutuhkan perhitungan dengan variabel dan rumus yang telah ditentukan. Adapun variabel dan rumus tersebut tercantum pada penjelasan dibawah :

1. Image Classification, Pada aplikasi PajakCerdas, disini aplikasi mampu untuk membedakan beberapa obyek pajak menjadi 4 golongan, yakni :
   - Tampak Standard, skor : 1
   - Tampak Cukup, skor : 2
   - Tampak Mewah, skor : 3
   - Tanah Kosong, tanpa skor

Aplikasi mampu untuk membedakan keempat obyek tersebut melalui tangkapan gambar kamera dari perangkat pengguna. Hasil dari tangkapan dan klasifikasi gambar diatas akan ditampilkan dalam bentuk scoring seperti yang sudah ditulis diatas. Yang menjadi catatan adalah, skor tersebut akan digunakan untuk menentukan golongan rumah, yang akan disertakan juga skor dari variabel lain seperti jumlah lantai dan sebagainya

2. Scoring, Dari beberapa variabel yang ada, dilakukan juga scoring untuk nantinya hasil dari scoring tersebut akan diolah untuk menentukan golongan rumah.
3. Rumus Nilai Properti Jika berupa tanah kosong

$$Nilai\ properti = (Luas\ tanah\ X\ nilai\ tanah\ sesuai\ dengan\ zona\ nilai\ tanah)$$

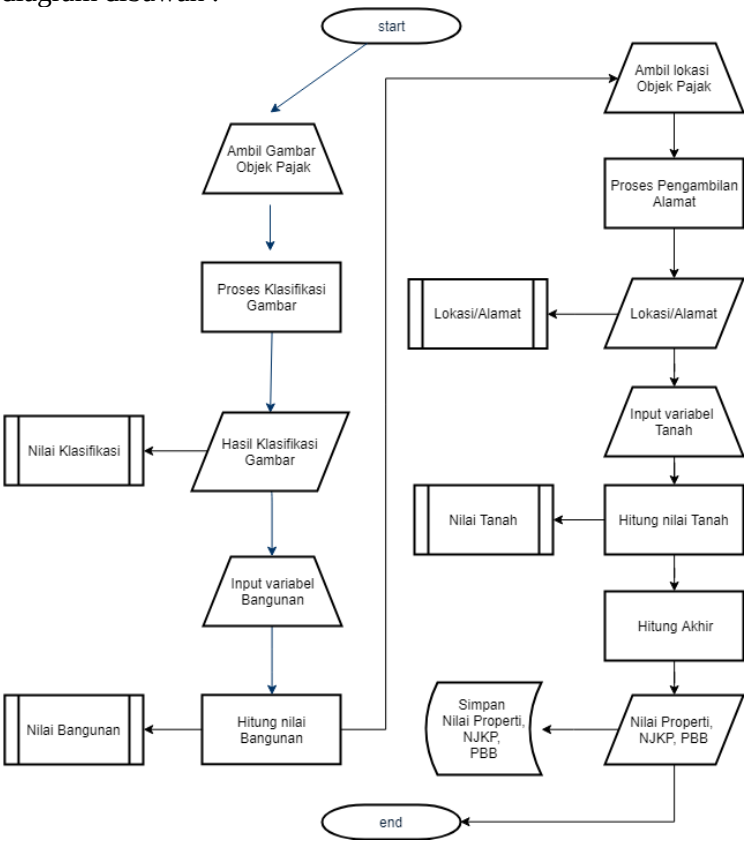
Dimana :

- Luas Tanah : hasil entry luas tanah
- Zona nilai tanah : hasil entry ZNT, zona tersebut diberi kode tanah mulai dari AA, AB, AC hingga FN

4. Nilai Properti Jika berupa tanah+bangunan
    - Nilai properti = (Luas tanah X nilai tanah sesuai dengan zona nilai tanah) + (Nilai Bangunan)
    - Nilai Bangunan = A – B
        - A = [hasil training nilai bangunan X Harga Bangunan sesuai hasil training nilai bangunan X luas bangunan]
        - B = [nilai penyusutan X C]
        - C = (hasil training nilai bangunan X Harga Bangunan sesuai hasil training nilai bangunan X luas bangunan)

- Harga bangunan mewah = Rp 6.000.000,-/meter persegi bangunan
- Harga bangunan sedang = Rp 4.500.000,-/meter persegi bangunan
- Harga bangunan kecil = Rp 3.000.000,-/meter persegi bangunan
- Luas Bangunan : hasil entri pada luas bangunan (meter persegi)
- Nilai penyusutan : 5%

Kesimpulan dari analisa kebutuhan aplikasi dapat dilihat pada diagram dibawah :



Gambar 4.2 Diagram Proses Bisnis Aplikasi PajakCerdas

4.2 Desain Sistem

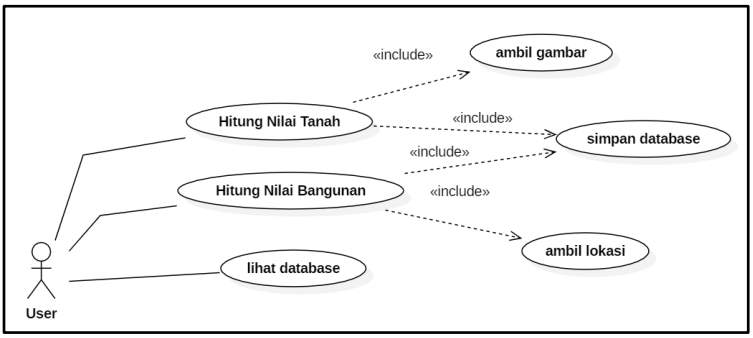
Setelah didapatkan kebutuhan fungsional dan nonfungsional, maka dilakukan desain sistem. Setelah kebutuhan fungsional telah dipastikan tidak berubah dan disepakati maka dimulai tahap desain sistem untuk menjadi dasaran pengembangan aplikasi *mobile* PajakCerdas. Desain sistem dilakukan dengan beberapa tahapan sebagai berikut :

4.2.1 Pembuatan Use Case Aplikasi

Desain use case dibuat dari hasil kebutuhan fungsional (tabel 4.1) yang diiriskan dengan fungsi dalam aplikasi (tabel 4.4). Use case bertujuan untuk menunjukkan interaksi antara aktor (user) dan aplikasi. Selain pembuatan *use case*, juga dilakukan pembuatan deskripsi (Use Case Description) yang berisikan detail behavior dari sebuah use case beserta pola interaksinya, adapun *use case* beserta *use case description* aplikasi *mobile* PajakCerdas sebagai berikut :

4.2.1.1 Use Case

Use case untuk aktor user dapat dilihat pada gambar 4.2. Adapun use case description untuk use case user dapat dilihat pada tabel 4.5 sampai 4.10.



Gambar 4.3 Use Case User

Tabel 4.5 Use Case Description Hitung Nilai Bangunan

U-01	*Use Case Name :* Hitung Nilai Bangunan
Primary actor : User	
Brief Description : Use case ini digunakan user yang telah masuk kedalam aplikasi dan sudah membuat tiket untuk menghitung nilai Bangunan objek pajak	
Pre Condition : 1. User telah memasuki aplikasi	
Basic Course : 1. User menekan tombol “Input PBB” 2. User memasukkan nilai-nilai sesuai variabel dan satuan yang ada 3. User menekan tombol “Halaman Selanjutnya” untuk memulai proses perhitungan	
Alternate Course : 2.1. User belum memasukkan nilai variabel yang dibutuhkan 2.2. User salah memasukkan satuan nilai variabel yang dibutuhkan	

Tabel 4.6 Use Case Description Hitung Nilai Tanah

U-02	*Use Case Name :* Hitung Nilai Tanah
Primary actor : User	
Brief Description : Use case ini digunakan user yang telah masuk kedalam aplikasi dan telah membuat tiket untuk menghitung nilai Tanah objek pajak	
Pre Condition : 1. User telah memasuki aplikasi 2. User telah menyelesaikan perhitungan nilai tanah	
Basic Course :	

1. User memasukkan nilai-nilai sesuai variabel dan satuan yang ada 2. User menekan tombol “Simpan Data” untuk memulai proses perhitungan
Alternate Course : 1.1. User belum memasukkan nilai variabel yang dibutuhkan 1.2. User salah memasukkan satuan nilai variabel yang dibutuhkan

Tabel 4.7 Use Case Description Lihat Database

U-03 *Use Case Name :* Lihat Database
Primary actor : User
Brief Description : Use case ini digunakan user untuk melihat informasi terkait data objek pajak yang telah dimasukkan kedalam database
Pre Condition : 1. User telah melakukan penyimpanan data kedalam database
Basic Course : 1. User memasuki halaman utama aplikasi 2. User menekan tombol “Lihat Database”
Alternate Course : 2.1 Tidak terkoneksi internet

Tabel 4.8 Use Case Description Ambil Gambar

U-04 *Use Case Name :* Ambil Gambar
Primary actor : User

Brief Description : Use case ini digunakan user untuk mengambil gambar objek pajak yang nantinya akan digunakan untuk keperluan perhitungan pajak
Pre Condition : 1. User sedang melakukan perhitungan bangunan
Basic Course : 1. User menekan tombol “Ambil Gambar”
Alternate Course : 1.1. Tidak terkoneksi dengan kamera

Tabel 4.9 Use Case Description Ambil Lokasi

U-05 *Use Case Name :* Ambil Lokasi
Primary actor : User
Brief Description : Use case ini digunakan user untuk mengambil lokasi dan alamat objek pajak
Pre Condition : 1. User sedang dalam proses melakukan perhitungan tanah
Basic Course : 1. User menekan tombol “Ambil Lokasi”
Alternate Course : -

Tabel 4.10 Use Case Description Lihat Database

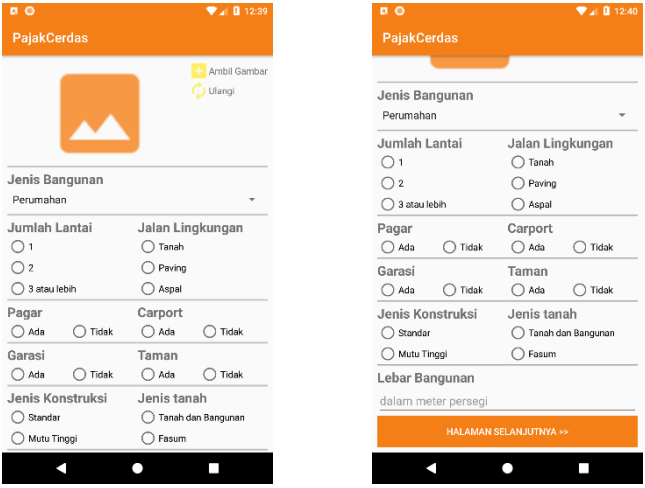
U-06 *Use Case Name :* Simpan Database
Primary actor : User

Brief Description : Use case ini digunakan user untuk menyimpan data objek pajak yang sudah diolah
Pre Condition : 1. User telah menyelesaikan input data pada Hitung nilai tanah 2. User telah menyelesaikan input data pada Hitung nilai bangunan
Basic Course : 1. User menekan tombol “Simpan ke Database”
Alternate Course : -

4.2.2 Pembuatan Desain Aplikasi

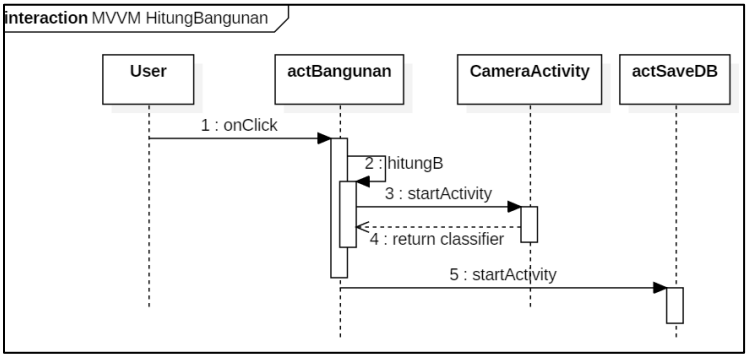
Dari use case dan peta *service* yang telah dibuat, maka langkah selanjutnya adalah pembuatan desain *user interface* aplikasi beserta *sequence diagram* masing-masing *design pattern* sebagai berikut.

4.2.2.1 Use Case U-01 (MVVM Hitung nilai Bangunan)



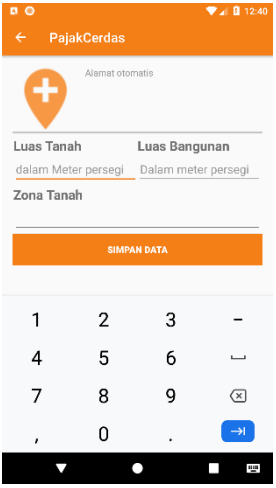
Gambar 4.4 *User interface* MVVM Hitung nilai Bangunan

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.4.



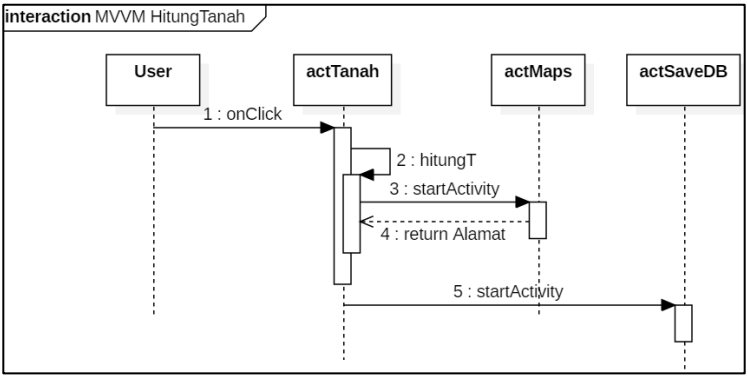
Gambar 4.5 Sequence Diagram Use Case MVVM Hitung nilai Bangunan

4.2.2.2 Use Case U-02 (MVVM Hitung nilai Tanah)



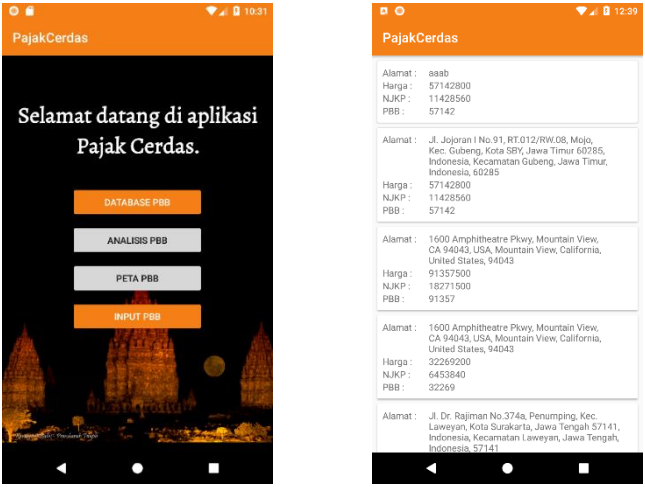
Gambar 4.6 *User interface* MVVM Hitung nilai Tanah

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.6.



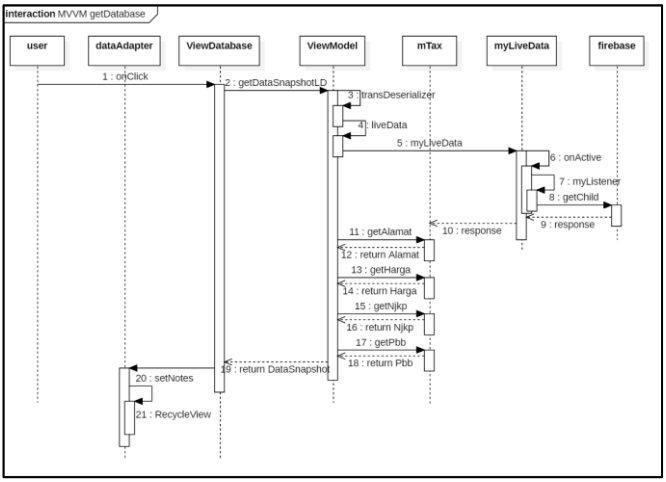
Gambar 4.7 Sequence Diagram Hitung Bangunan

4.2.2.3 Use Case U-03 (MVVM Lihat Database)



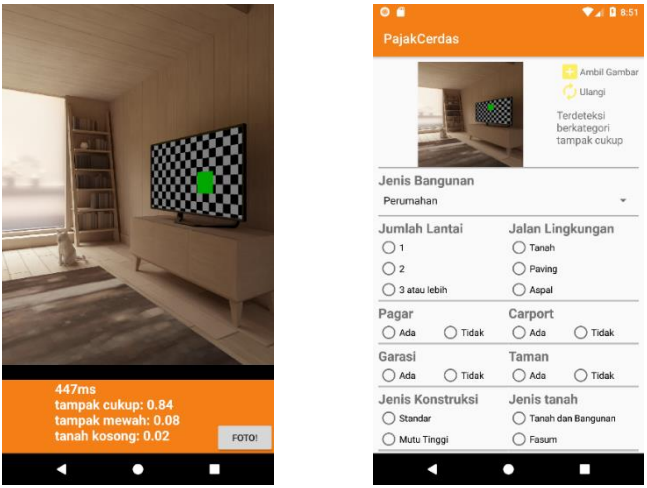
Gambar 4.8 *User interface* MVVM Lihat Notifikasi

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.8.



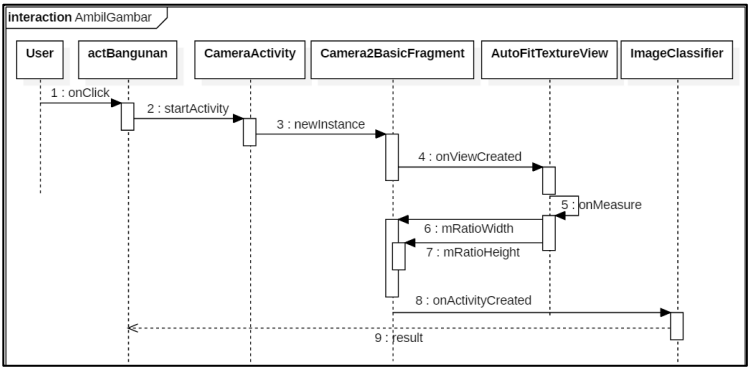
Gambar 4.9 Sequence Diagram MVVM Lihat Database

4.2.2.4 Use Case U-04 (MVVM Ambil Gambar)



Gambar 4.10 *User interface* MVVM Ambil Gambar

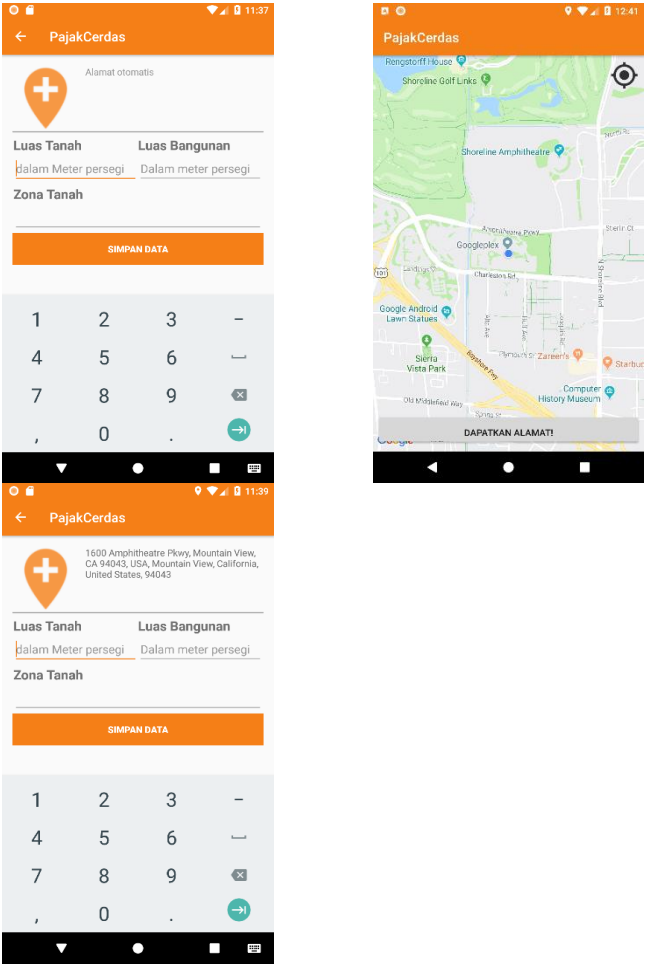
Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.10.



Gambar 4.11 Sequence Diagram MVVM Ambil Gambar

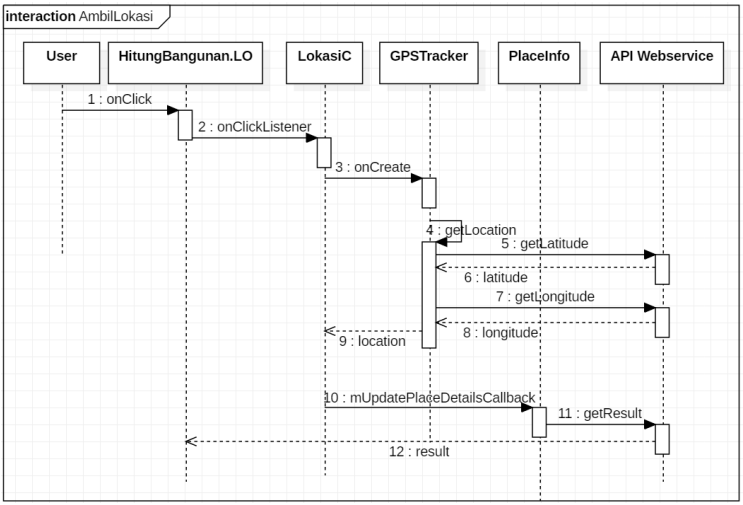
Sebagai catatan, usecase AmbilGambar ini menggunakan API yang disediakan oleh Tensorflow untuk keperluan pengklasifikasian gambar objek pajak, sehingga dari segi desain aplikasi tidak ada perubahan antara MVVM dengan MVC.

4.2.2.5 Use Case U-05 (MVVM Ambil Lokasi)



Gambar 4.12 *User interface* MVVM Ambil Lokasi

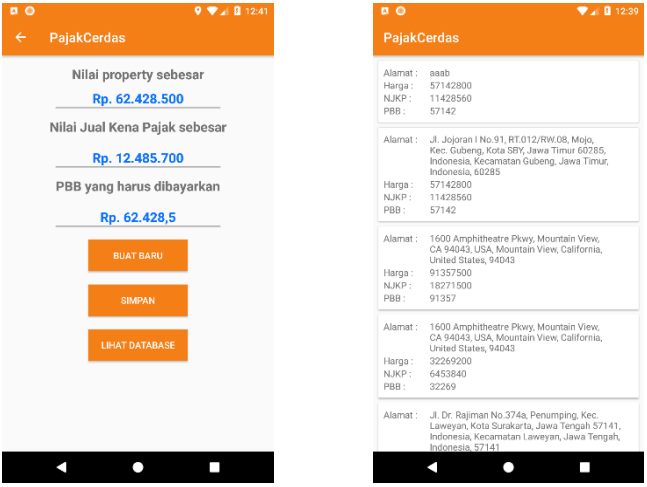
Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.12.



Gambar 4.13 Sequence Diagram MVVM Ambil Lokasi

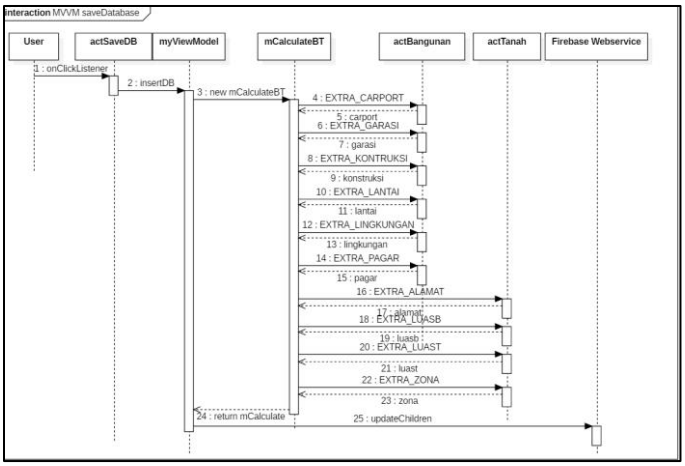
Sebagai catatan, usecase Ambil Lokasi ini menggunakan API yang disediakan oleh Google Maps untuk keperluan pengambilan data alamat dan lokasi objek pajak, sehingga dari segi desain aplikasi tidak ada perubahan antara MVVM dengan MVC.

4.2.2.6 Use Case U-06 (MVVM Simpan Database)



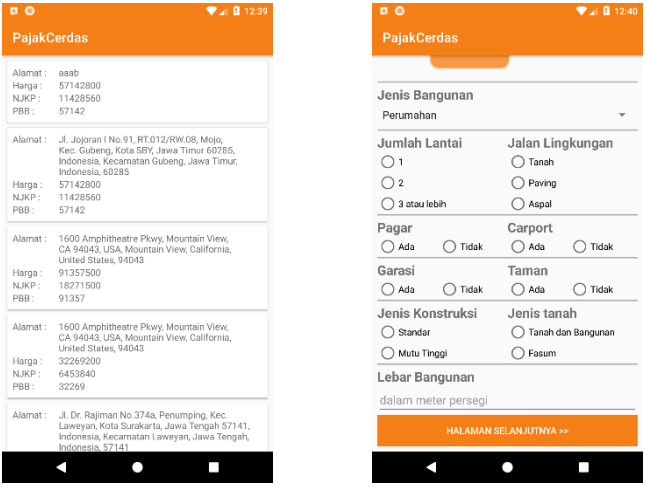
Gambar 4.14 MVVM Simpan Database

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.14 dibawah.



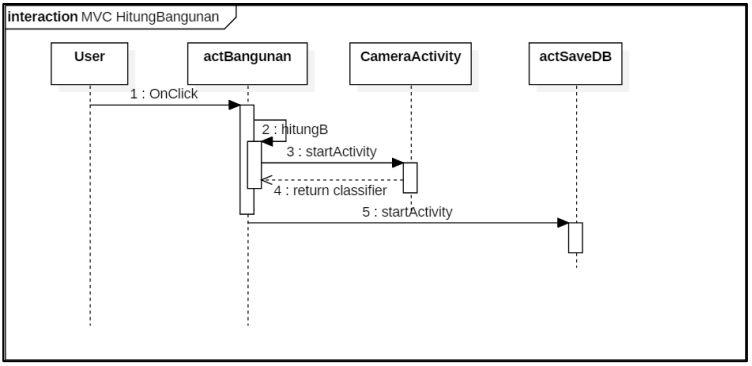
Gambar 4.15 Sequence Diagram MVVM Simpan Database

4.2.2.7 Use Case U-01 (MVC Hitung nilai Bangunan)



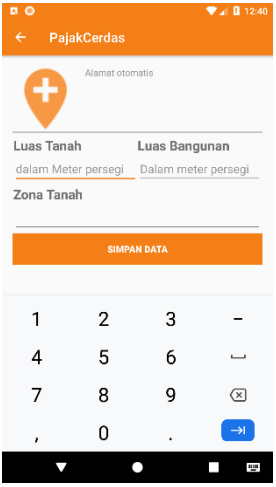
Gambar 4.16 MVC Hitung nilai Bangunan

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.16.



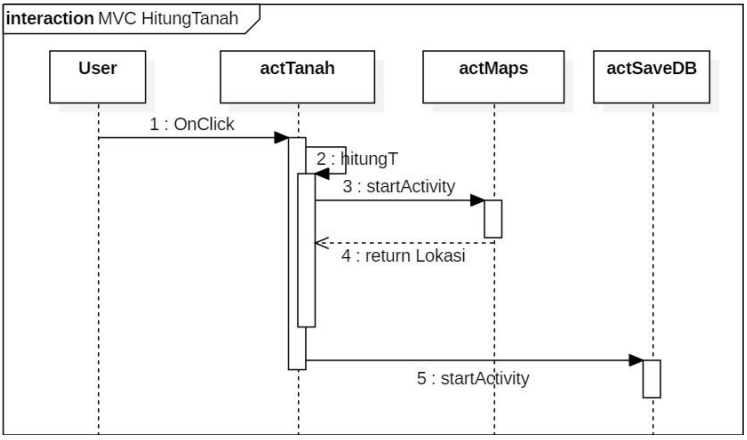
Gambar 4.17 Sequence Diagram MVC Hitung nilai Bangunan

4.2.2.8 Use Case U-02 (MVC Hitung nilai Tanah)



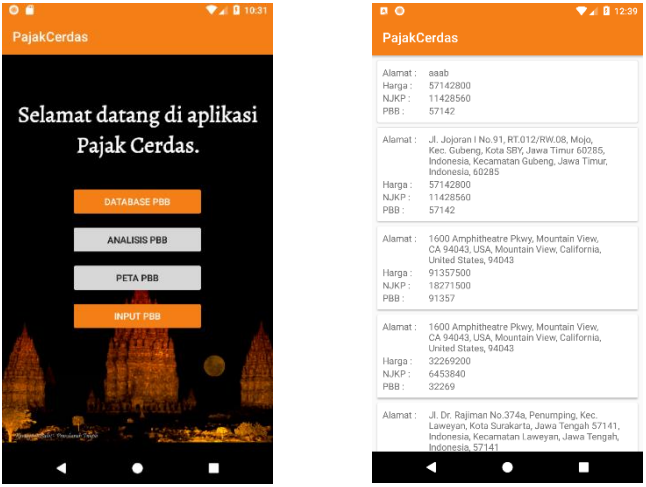
Gambar 4.18 MVC Hitung nilai Tanah

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.18.



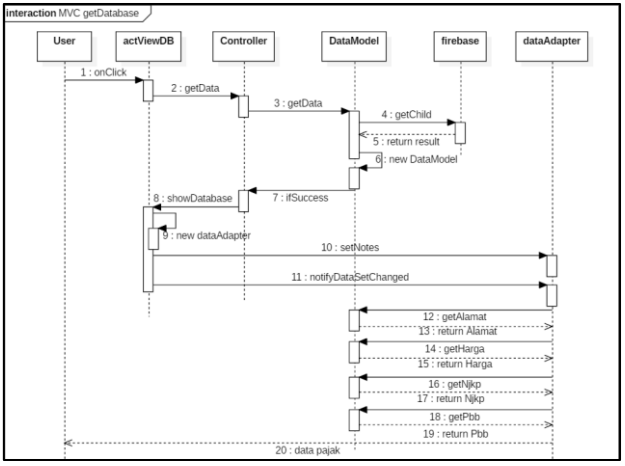
Gambar 4.19 Sequence Diagram MVC Hitung nilai Tanah

4.2.2.9 Use Case U-03 (MVC Lihat Database)



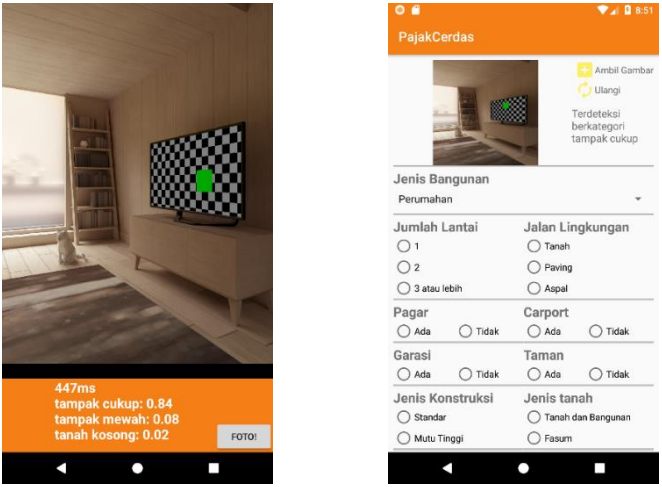
Gambar 4.20 MVC Lihat Database

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.20.



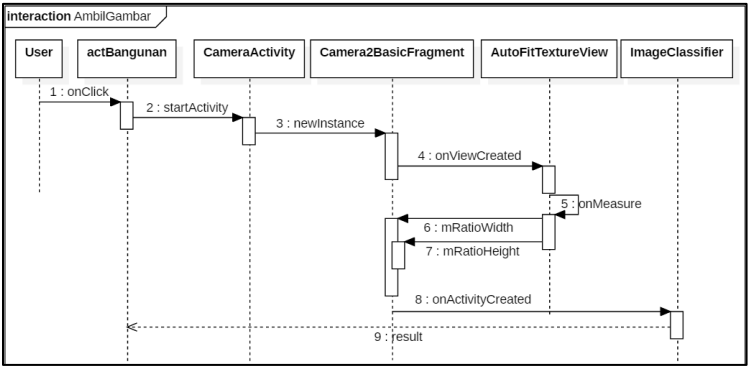
Gambar 4.21 Sequence Diagram MVC Lihat Database

4.2.2.10 Use Case U-04 (MVC Ambil Gambar)



Gambar 4.22 *User interface* MVC Ambil Gambar

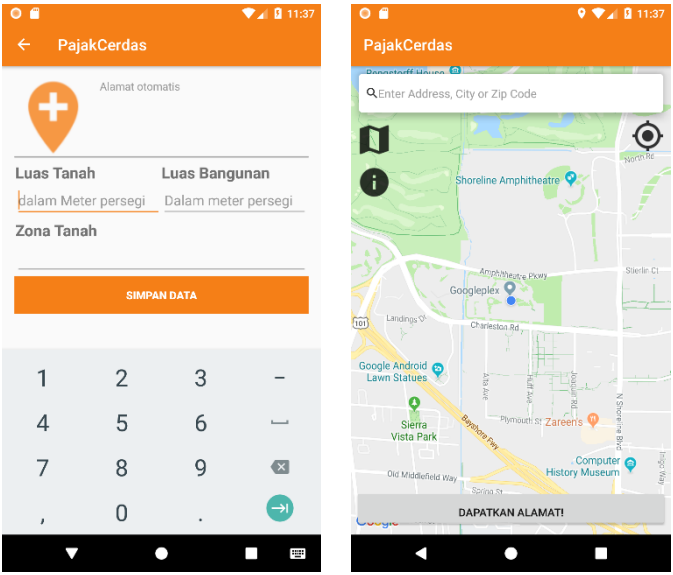
Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.22.



Gambar 4.23 Sequence Diagram MVC Ambil Gambar

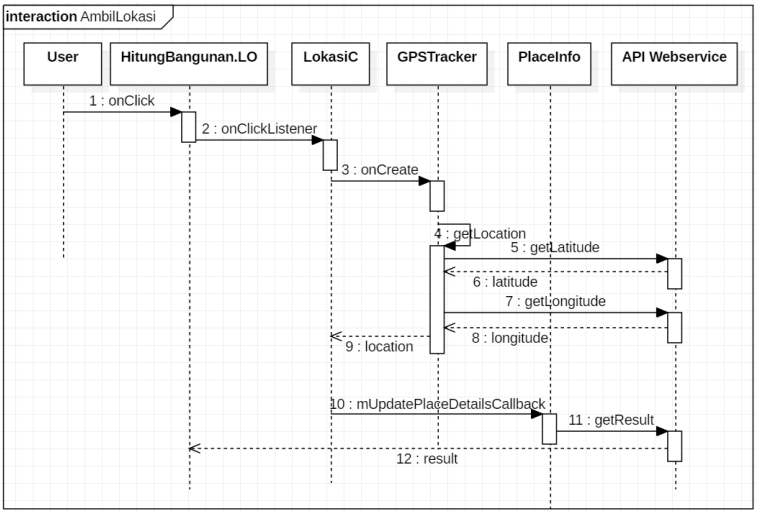
Sebagai catatan, usecase AmbilGambar ini menggunakan API yang disediakan oleh Tensorflow untuk keperluan pengklasifikasian gambar objek pajak, sehingga dari segi desain aplikasi tidak ada perubahan antara MVVM dengan MVC.

4.2.2.11 Use Case U-05 (MVC Ambil Lokasi)



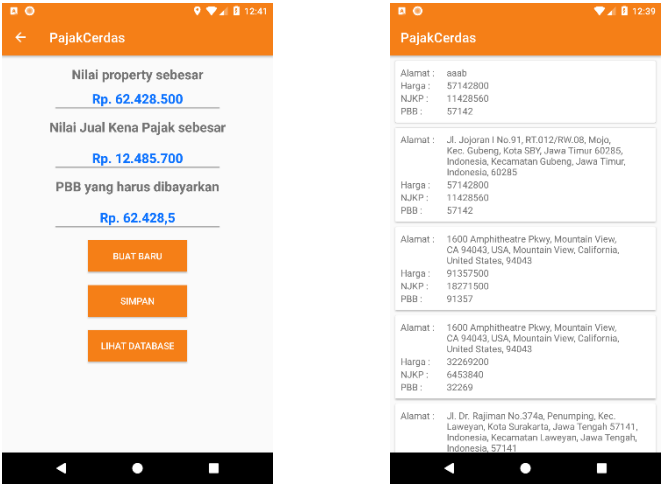
Gambar 4.24 *User interface* MVC Ambil Lokasi

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.24.



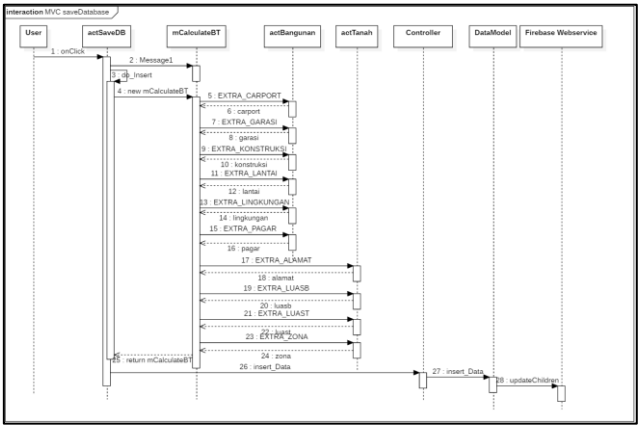
Gambar 4.25 Sequence Diagram MVC Ambil Lokasi

4.2.2.12 Use Case U-06 (MVC Simpan Database)



Gambar 4.26 *User interface* MVC Simpan Database

Adapun *sequence diagram* dari desain *user interface* diatas dapat dilihat pada gambar 4.26.



Gambar 4.27 Sequence Diagram MVC Simpan Database

4.2.3 Perancangan Test Case

Tahap ini merupakan tahapan dimana skenario untuk melakukan pengujian pada setiap use case yang ada dibuat. Tujuan dari pembuatan *test case* adalah untuk menghasilkan berbagai kemungkinan interaksi guna dilakukan pengujian apakah sistem telah berjalan sesuai dengan desain yang telah dibuat. *Test case* dibuat berdasarkan metode pengujian *Instrumented Unit testing*. Setiap *test case* mempunyai nomor pengujian dan skenario pengujian tertentu. Detail mengenai perancangan *test case* terdapat pada lampiran A halaman 98.

4.3 Validasi Desain

Untuk mendukung terciptanya sebuah *design pattern* yang baik, dibutuhkan sebuah variabel yang mampu untuk memvalidasi kesesuaian yang ada pada aspek-aspek pendukung *design pattern* dengan kesesuaiannya pada implementasi kode program maupun desain programnya. Pada subbab ini, penulis melampirkan beberapa kriteria yang dikutip dari sumber-sumber yang kredibel mengenai bagaimana seharusnya sebuah

design pattern dibuat dalam bentuk checklist. Detail mengenai perancangan validasi desain terdapat pada lampiran B halaman 103.

Halaman ini sengaja dikosongkan

BAB V
IMPLEMENTASI

Dalam bab ini akan dijelaskan implementasi dari perancangan yang telah dilakukan sesuai dengan metode pengembangan yang dibuat.

5.1 Lingkungan Implementasi

Pengembangan aplikasi *mobile* PajakCerdas menggunakan komputer dengan spesifikasi yang tertera pada tabel 5.1.

Tabel 5.1 Spesifikasi Komputer Pengembang

Processor	1.6 GHz Intel Core i5
Memory	16 GB RAM DDR4
Sistem Operasi	*Microsoft Windows 10 Home SL*
Graphic Card	*Intel UHD Graphics 620*

Aplikasi PajakCerdas dikembangkan dengan menggunakan beberapa teknologi seperti editor, database, bahasa pemrograman, dan *API* yang terdapat pada tabel 5.2.

Tabel 5.2 Teknologi Pengembangan yang Digunakan

Bahasa Pemrograman	Java, XML
Database	Firebase Realtime Database
Editor (IDE)	Android Studio

API	• Firebase Realtime Database • Tensorflow • Keras • Google Maps

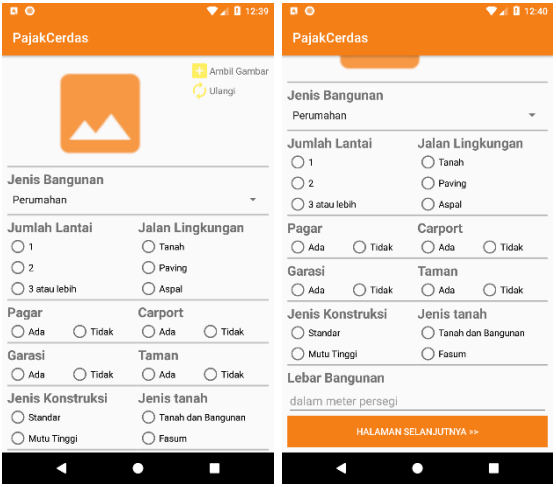
5.2 Implementasi Desain Aplikasi

Aplikasi *mobile* PajakCerdas dibangun dengan fokus pada pengembangan menggunakan pendekatan *design pattern* MVVM. Untuk menunjang justifikasi penggunaan *design pattern* MVVM, disertakan juga implementasi kode program aplikasi dengan pendekatan *design pattern* lain, dalam kasus ini *design pattern* MVC. Setiap implementasi yang menggunakan *design pattern* MVVM maupun MVC akan menghasilkan aplikasi yang sama secara tampilan maupun secara fungsional. Yang menjadi perbedaan dari kedua pendekatan tersebut hanyalah kode programnya saja.

Pada pembahasan dibawah, setiap implementasi dari masing-masing *design pattern* akan mempunyai kode tersendiri untuk membedakan kedua pendekatan masing-masing *design pattern*.

5.2.1 Implementasi Hitung nilai Bangunan

Fungsi pertama yang diimplementasikan sesuai desain adalah Hitung nilai Bangunan dengan tampilan seperti pada gambar 5.1 dibawah ini.



Gambar 5.1 *User interface* Hitung nilai Bangunan

Adapun potongan kode *usecase* Hitung nilai Bangunan adalah sebagai berikut.

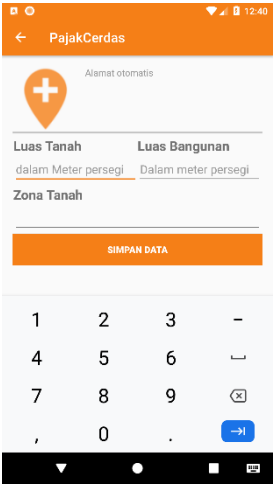
```
01.  public void hitungB(){
02.      Intent intent = new Intent("com.example.ryzen.pajakcerdas.actTanah");
03.      if(tanah.equals("") || lantai.equals("") || konstruksi.equals("") || pagar.equals("") ||
04.              carport.equals("") || garasi.equals("") || taman.equals("") ||
05.              lingkungan.equals("") || tampakTensor.equals("") ||
06.              String.valueOf(lebarrb.getText()).equals("")){
07.          Toast.makeText(this, R.string.toast_text, Toast.LENGTH_SHORT).show();
08.      } else {
09.          int lebarr = Integer.parseInt(String.valueOf(lebarrb.getText()));
10.          Log.d("hitungB", "hasil lebarr : " + lebarr);
11.          if (lebarr < 7) {
12.              lebarbeneran = "1";
13.          } else if(7 <= lebarr && lebarr <= 10){
14.              lebarbeneran = "2";
15.          } else {
16.              lebarbeneran = "3";
17.          }
18.          String hasilneural = neural(lantai, konstruksi, pagar, carport, garasi, taman,
19.                  lingkungan, tampakTensor, lebarbeneran);
20.          intent.putExtra(EXTRA_JENIS, posisi);
21.          intent.putExtra(EXTRA_LANTAI, lantai);
22.          intent.putExtra(EXTRA_TANAH, tanah);
23.          intent.putExtra(EXTRA_PAGAR, pagar);
24.          intent.putExtra(EXTRA_CARPORT, carport);
25.          intent.putExtra(EXTRA_GARASI, garasi);
26.          intent.putExtra(EXTRA_TAMAN, taman);
27.          intent.putExtra(EXTRA_TAMPAK, tampakTensor);
28.          intent.putExtra(EXTRA_LINGKUNGAN, lingkungan);
29.          intent.putExtra(EXTRA_KONSTRUKSI, konstruksi);
30.          intent.putExtra(EXTRA_NILAI, hasilneural);
31.          intent.putExtra(EXTRA_LEBARBANGUNAN, lebarr);
32.          startActivity(intent);
33.      }
34.  }
```

Gambar 5.2 Potongan Kode Program Fungsi hitungB MVVM & MVC

Pada usecase ini, tidak banyak fungsi yang melibatkan database, sehingga kode program hanya berinteraksi antar komponen *view* saja yang menyebabkan kode program antara MVVM dengan MVC sama persis.

5.2.2 Implementasi Hitung nilai Tanah

Fungsi kedua yang diimplementasikan sesuai desain adalah Hitung nilai Tanah dengan tampilan seperti pada gambar 5.3 dibawah ini.



Gambar 5.3 *User interface* Hitung nilai Tanah

Adapun potongan kode *usecase* Hitung nilai Tanah adalah sebagai berikut.

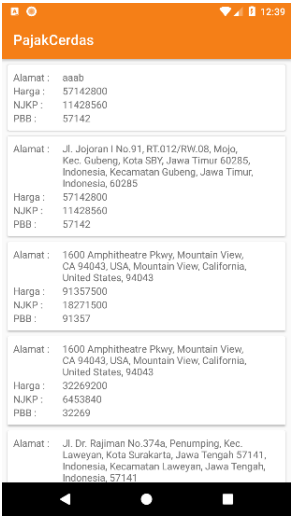
```
01.  public void hitungTanah(){
02.          Intent x = new Intent("com.example.ryzen.pajakcerdas.actSaveDB");
03.          if (luass.getText().toString().matches("") || zona.getText().toString().matches("")) {
04.              Toast.makeText(this, R.string.toast_text, Toast.LENGTH_SHORT).show();
05.          } else {
06.              hasiltanah = Integer.parseInt(luass.getText().toString());
07.              hasilbangunan = Integer.parseInt(luasb.getText().toString());
08.              x.putExtra(EXTRA_ALAMAT, result.getText());
09.              x.putExtra(EXTRA_LUAST, hasiltanah);
10.              x.putExtra(EXTRA_LUASB, hasilbangunan);
11.              x.putExtra(EXTRA_ZONA, zona.getText().toString().toUpperCase());
12.              x.putExtra(EXTRA_EXTRA, hasilnya1);
13.              x.putExtra(EXTRA_NILAI2, hasilnya2);
14.              x.putExtra(EXTRA_TANAH2, EXTRA_TANAH);
15.              x.putExtras(getIntent());
16.              startActivity(x);
17.          }
18.      }
```

Gambar 5.4 Potongan Kode Program Fungsi hitungTanah MVVM & MVC

Kode program diatas berfungsi untuk menyalurkan nilai variabel tanah yang diperlukan untuk menyimpan data kedalam database. Namun tetap pada usecase ini sebagian besar data hanya berpindah antar activity, sehingga tidak ada perbedaan antara *design pattern* MVVM dengan MVC karena tidak membutuhkan koneksi kepada database.

5.2.3 Implementasi Lihat Database

Fungsi ketiga yang diimplementasikan sesuai desain adalah Lihat Database dengan tampilan seperti pada gambar 5.5 dibawah ini.



Gambar 5.5 *User interface* Lihat Notifikasi Aplikasi

Adapun potongan kode untuk usecase ini dalam *design pattern* MVVM adalah sebagai berikut.

```
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.recycle_viewdatabase);
    rc = findViewById(R.id.recycler_view);
    vm = ViewModelProviders.of(this).get(myViewModel.class);
    LiveData<mTax> liveData = vm.getDataSnapshotLiveData();
    checkConn();
    liveData.observe(this, new Observer<mTax>() {
        @Override
        public void onChanged(@Nullable mTax mTax) {
            Log.d("observer", "onChanged: VM active");
            if (mTax != null) {
                String alamat = mTax.getAlamat();
                long harga = mTax.getHarga();
                long njkp = mTax.getNjkp();
                Integer pbb = mTax.getPbb();
                mUsername.add(new mTax(alamat, harga, njkp, pbb));
                adapter.setNotes(mUsername);
                adapter.notifyDataSetChanged();
            }
        }
    });
    initRecyclerView();
}
```

Gambar 5.6 Potongan Kode Program *View* Fungsi Lihat Database

```
01. public class myViewModel extends AndroidViewModel {
02.     @Override
03.     protected void onCleared(){
04.         Log.d("onCleared ", "ViewModel Destroyed");
05.     }
06.     private static final DatabaseReference DATA_OBJEK =
07.             FirebaseDatabase.getInstance().getReference("dataObjek");
08.     private final myLiveData liveData = new myLiveData(DATA_OBJEK);
09.     private final LiveData<mTax> transDeserializer = Transformations.map
10.             (liveData, new Deserializer());
11.     private class Deserializer implements Function<DataSnapshot, mTax> {
12.         @Override
13.         public mTax apply(DataSnapshot hohoho) {
14.             return hohoho.getValue(mTax.class);
15.         }
16.     }
17.     public myViewModel(@NonNull Application application) {
18.         super(application);
19.     }
20.     @NonNull
21.     public LiveData<mTax> getDataSnapshotLiveData() {
22.         return transDeserializer;
23.     }
24.     public void insert(mCalculateBT ct){..}
```

Gambar 5.7 Potongan Kode Program *ViewModel* Fungsi Lihat Database

```
01. public class myLiveData extends LiveData<DataSnapshot> {
02.     private static final String LOG_TAG = "myLiveData";
03.     private final Query query;
04.     private final myListener listener = new myListener();
05.     public myLiveData(Query query) {
06.         this.query = query;
07.     }
08.     public myLiveData(DatabaseReference ref) {
09.         this.query = ref;
10.     }
11.     @Override
12.     protected void onActive() {
13.         Log.d(LOG_TAG, "onActive");
14.         actViewDB.clearData();
15.         query.addChildEventListener(listener);
16.     }
17.     @Override
18.     protected void onInactive() {
19.         Log.d(LOG_TAG, "onInactive");
20.         query.removeEventListener(listener);
21.         actViewDB.clearData();
22.     }
23.     private class myListener implements ChildEventListener {..}
24. }
```

Gambar 5.8 Potongan Kode Program *LiveData* Fungsi Lihat Database MVVM

Sedangkan untuk kode program *design pattern* MVC pada fungsi Lihat Database adalah sebagai berikut.

```
01. public class actViewDB extends AppCompatActivity {
02.     private List<DataModel> modelList;
03.     private RecyclerView rc;
04.     private dataAdapter adapter;
05.     @Override
06.     protected void onCreate(@Nullable Bundle savedInstanceState) {
07.         super.onCreate(savedInstanceState);
08.
09.         setContentView(R.layout.recycle_viewdatabase);
10.
11.         rc = findViewById(R.id.recycler_view);
12.         rc.setLayoutManager(new LinearLayoutManager(this));
13.         rc.setHasFixedSize(true);
14.         Controller controller = new Controller(this);
15.         controller.getData();
16.         checkConn();
17.     }
18.
19.     public void showDatabase(List<DataModel> mUsername){
20.         modelList = new ArrayList<>();
21.         this.modelList = mUsername;
22.
23.         adapter = new dataAdapter();
24.         adapter.setNotes(modelList);
25.         rc.setAdapter(adapter);
26.     }
27.
28.     public void checkConn(){..}
29. }
```

Gambar 5.9 Potongan Kode Program *View* Fungsi Lihat Database MVC

```
01. public class Controller {
02.     private static actViewDB activity;
03.     public Controller(actViewDB activity) {
04.         this.activity = activity;
05.     }
06.     public static void getData(){
07.         DataModel.getData();
08.     }
09.     public static void ifSuccess(List<DataModel> mUsername){
10.         activity.showDatabase(mUsername);
11.     }
12.     public static void ifFail(){
13.         Log.d("Error", "Error");
14.     }
```

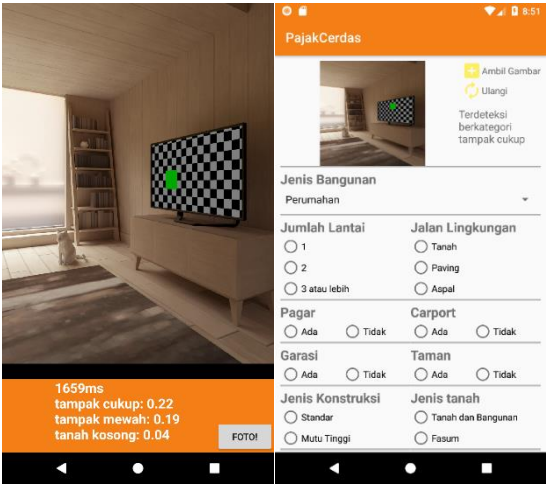
Gambar 5.10 Potongan Kode Program *Controller* Fungsi Lihat Database MVC

```
01. public static void getData() {
02.     FirebaseDatabase database = FirebaseDatabase.getInstance();
03.     DatabaseReference myRef = database.getReference("dataObjek");
04.     List<DataModel> mUsername = new ArrayList<>();
05.
06.     myRef.addValueEventListener(new ValueEventListener() {
07.         @Override
08.         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
09.             mUsername.clear();
10.             for(DataSnapshot ds : dataSnapshot.getChildren()) {
11.                 String alamat = ds.child("alamat").getValue(String.class);
12.                 long harga = (long) ds.child("harga").getValue();
13.                 long njkp = (long) ds.child("njkp").getValue();
14.                 Integer pbb = ds.child("pbb").getValue(Integer.class);
15.                 Log.d("TAG", alamat + " / " + harga);
16.                 mUsername.add(new DataModel(alamat, harga, njkp, pbb));
17.                 Controller.ifSuccess(mUsername);
18.             }
19.         }
20.
21.         @Override
22.         public void onCancelled(@NonNull DatabaseError databaseError) {
23.             Controller.ifFail();
24.         }
25.     });
26. }
```

Gambar 5.11 Potongan Kode Program *getData* Fungsi Lihat Database MVC

5.2.4 Implementasi Ambil Gambar

Fungsi keempat yang diimplementasikan sesuai desain adalah Ambil Gambar dengan tampilan seperti pada gambar 5.12 dibawah ini.



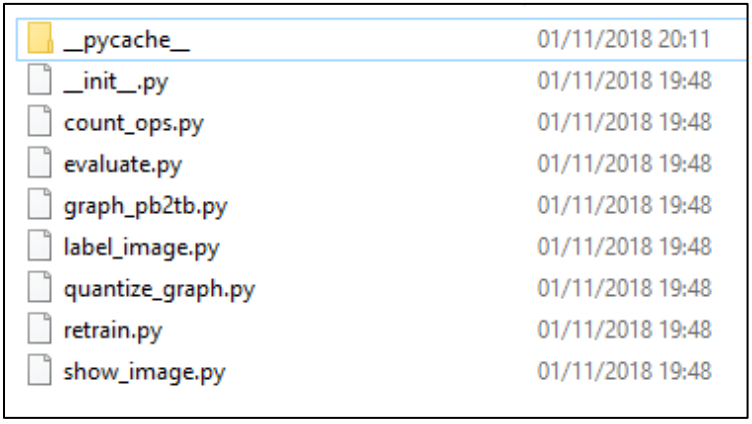
Gambar 5.12 *User interface* Ambil Gambar

Fungsi Ambil Gambar merupakan fungsi yang dijalankan menggunakan API dan Library dari google dan tensorflow, sehingga dalam kasus penelitian ini tidak akan dibedakan antara MVVM dengan MVC. Sebelum mengimplementasikan kode program pada aplikasi, terdapat tahapan untuk membuat data model berbagai macam jenis rumah yang nantinya akan digunakan sebagai rujukan untuk menentukan nilai jual objek pajak. Untuk membuat model tersebut, sebelumnya penulis harus menyiapkan data gambar yang dibutuhkan seperti pada contoh dibawah ini :

Gambar 5.13 Mempersiapkan pembuatan model

Data dibagi menjadi 4 kelompok sesuai dengan spesifikasi aplikasi yang dibutuhkan, yakni gambar rumah dengan klasifikasi "tampak cukup", "tampak_mewah", "tampak_standar" dan gambar tanah kosong yang berlabel "tanah_kosong". Masing-masing kelompok mempunyai jumlah gambar sebanyak 500 buah gambar, idealnya semakin banyak jumlah gambar, semakin baik.

Setelah menyiapkan data yang dibutuhkan, tahap selanjutnya adalah pembuatan model. Untuk tahap ini, diperlukan algoritma yang mampu mengubah gambar menjadi matriks data. Pada studi kasus ini, penulis memutuskan untuk menggunakan algoritma yang telah disediakan oleh Tensorflow.org[34].



Gambar 5.14 Kumpulan script algoritma Tensorflow

Selain karena kemampuannya yang dikenal baik dalam memproduksi model untuk machine learning, tensorflow juga diketahui sangat populer dikalangan pengembang Artificial Intelligence, sehingga secara tidak langsung memudahkan penulis dalam mengembangkan model machine learning dalam studi kasus ini.

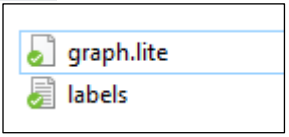
Dalam proses pembangunan model, untuk dapat menggunakan algoritma dengan baik, diperlukan script untuk

memberdayagunakan algoritma yang sudah kita punya dari tensorflow diatas. Script yang bisa digunakan untuk membangun model adalah sebagai berikut :

```
1. python -m scripts.retrain
2. --
   bottleneck_dir=tf_files/bottlenecks
   #lokasi menyimpan file bottleneck
3. --
   how_many_training_steps 500
   #berapa banyak proses training dilakukan
4. --model_dir=tf_files/models/mobilenet_0.50_224
   #lokasi penyimpanan algoritma mobilenet
5. --architecture=mobilenet_0.50_224
   #versi mobilenet
6. --
   summaries_dir=tf_files/training_summaries/mobile
   net_0.50_224
   #lokasi menyimpan file model setengah jadi
7. --
   output_graph=tf_files/retrained_graph.pb
   #lokasi menyimpan file grafik model
8. --output_labels=tf_files/retrained_labels.txt
   #lokasi menyimpan file label
9. --
   image_dir=tf_files/DataSet/
   #lokasi data gambar
```

Setelah script selesai dijalankan, maka model yang diperlukan untuk mengimplementasikan machine learning pada aplikasi telah berhasil dibuat pada direktori yang telah ditentukan pada script diatas. Perlu diketahui bahwa script yang dijalankan diatas bisa memakan waktu pemrosesan yang cukup lama, tergantung kepada kemampuan komputasi yang digunakan ketika menjalankan script tersebut. Sebagai referensi, dengan Lingkungan Implementasi yang telah tercantum pada bab 5.1, dibutuhkan waktu kurang lebih 2 jam 30 menit untuk menyelesaikan script diatas.

Model yang telah berhasil dibangun nantinya akan dimasukkan kedalam folder aplikasi pada direktori khusus yang sudah dirancang untuk menampung file model, yakni pada android/tf*mobile*/assets .



Gambar 5.15 File model didalam direktori *assets*

Setelah model berhasil dimasukkan kedalam aplikasi, maka tahap selanjutnya adalah menyiapkan kode program pada aplikasi untuk membaca model dan menggunakannya untuk proses klasifikasi gambar obyek pajak. Kode program dibawah adalah inisialisasi dan penggunaan graph.lite :

```
1.  private static final String MODEL_PATH = "graph.
    lite";
2.  private static final String LABEL_PATH = "labels
    .txt";
3.  private MappedByteBuffer loadModelFile(Activity
    activity) throws IOException {
4.    AssetFileDescriptor fileDescriptor = activity.
    getAssets().openFd(MODEL_PATH);
5.    FileInputStream inputStream = new FileInputStr
    eam(fileDescriptor.getFileDescriptor());
6.    FileChannel fileChannel = inputStream.getChann
    el();
7.    long startOffset = fileDescriptor.getStartOffs
    et();
8.    long declaredLength = fileDescriptor.getDeclar
    edLength();
9.    return fileChannel.map(FileChannel.MapMode.REA
    D_ONLY, startOffset, declaredLength);
10. }
```

Kemudian untuk membuat tampilan aplikasi, kode program Ambil Gambar pada View adalah sebagai berikut.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  this.requestWindowFeature(Window.FEATURE_NO_TITLE);
  this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
       WindowManager.LayoutParams.FLAG_FULLSCREEN);
  this.setContentView(R.layout.activity_camera);
  if (null == savedInstanceState) {
    getFragmentManager()
        .beginTransaction()
        .replace(R.id.container, Camera2BasicFragment.newInstance())
        .commit();
  }
}
```

Gambar 5.16 Potongan Kode Program *View* Fungsi Ambil Gambar

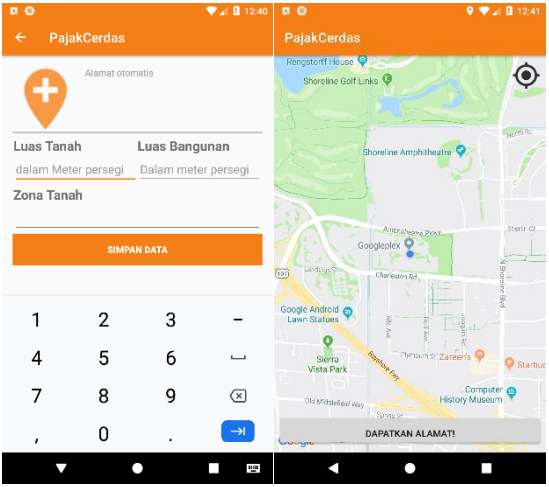
Berikut potongan kode program lanjutan fungsi Ambil Gambar untuk mengirimkan nilai klasifikasi ke *view*.

```
public void doTensor(){
  Intent intent = new Intent("com.example.ryzen.pajakcerdas.actBangunan");

  String aa = ImageClassifier.tampilkan;
  intent.putExtra(EXTRA_TENSOR, aa);

  Bitmap b = textureView.getBitmap(ImageClassifier.DIM_IMG_SIZE_X,
ImageClassifier.DIM_IMG_SIZE_Y); // your bitmap
  ByteArrayOutputStream bs = new ByteArrayOutputStream();
  b.compress(Bitmap.CompressFormat.PNG, 50, bs);
  intent.putExtra("byteArray", bs.toByteArray());
  startActivity(intent);
}
```

Gambar 5.17 Potongan Kode Program *doTensor* Fungsi Ambil Gambar

5.2.5 Implementasi Ambil Lokasi

Fungsi kelima yang diimplementasikan sesuai desain adalah Ambil Lokasi dengan tampilan seperti pada gambar 5.15 dibawah ini.



Gambar 5.18 *User interface* Ambil Lokasi

Potongan kode *view* method onCreateView() di dalam *class* actMaps adalah sebagai berikut.

```
01.  protected void onCreate(@Nullable Bundle savedInstanceState) {
02.      super.onCreate(savedInstanceState);
03.      setContentView(R.layout.activity_lokasi);
04.
05.      mGps = (ImageView) findViewById(R.id.ic_gps);
06.
07.      getLocationPermission();
08.      mContext = this;
09.
10.      gps = new GPSTracker(mContext, actMaps.this);
11.
12.      if (gps.canGetLocation()) {
13.          latitude = gps.getLatitude();
14.          longitude = gps.getLongitude();
15.      } else {
16.          // Can't get location.
17.          // GPS or network is not enabled.
18.          // Ask user to enable GPS/network in settings.
19.          gps.showSettingsAlert();
20.      }
21.      getlocation = findViewById(R.id.getlocation);
22.      getlocation.setOnClickListener(new View.OnClickListener() {..}
23.  }
```

Gambar 5.19 Potongan Kode Program *View* Fungsi Ambil Lokasi

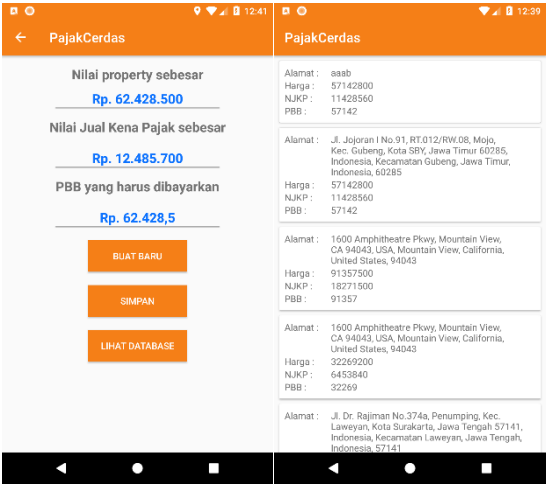
Berikut potongan kode program untuk mengirim nilai hasil pengambilan lokasi dan alamat objek pajak.

```
getlocation.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            addressList = geocoder.getFromLocation(latitude,longitude,1);

            if (addressList !=null) {
                String addressStr = addressList.get(0).getAddressLine(0);
                String areaStr = addressList.get(0).getLocality();
                String cityStr = addressList.get(0).getAdminArea();
                String countryStr = addressList.get(0).getCountryName();
                String postalcodeStr = addressList.get(0).getPostalCode();

                final String fullAddress = addressStr+", "+areaStr+", "+cityStr+",
                      "+countryStr+", "+postalcodeStr;
                Intent myintent = new Intent("com.example.ryzen.pajakcerdas.actTanah");
                myintent.putExtra("alamat", fullAddress.toString());
                myintent.putExtras(getIntent());
                Log.d(TAG, "Isi full address: " + fullAddress.toString());
                startActivity(myintent);

                /*Controller.mapsSuccess(fullAddress);*/
            } else {
                Toast.makeText(mContext, "GPS gagal terbuka", Toast.LENGTH_SHORT).show();
                return;
            }
        } catch (IOException e) {
            Log.e(TAG, "onClick: NullPointerException: " + e.getMessage() );
        }
    }
});
```

Gambar 5.20 Potongan Kode Program Fungsi *getLocation*

5.2.6 Implementasi Simpan Database

Fungsi selanjutnya yang diimplementasikan sesuai desain adalah lihat info buat tiket dengan tampilan seperti pada gambar 5.18 dibawah ini.



Gambar 5.21 *User interface* Simpan Database

Setelah mendapatkan seluruh nilai yang diperlukan, fungsi simpan database ini akan mengirimkan nilai-nilai pajak ke database yang digunakan yakni Firebase. Berikut beberapa potongan kode program untuk *design pattern* MVVM.

```
01.  public void do_Insert(double hitungan, double njkp, double pbb){
02.      Intent x = getIntent();
03.      mCalculateBT ct = new mCalculateBT(x.getStringExtra(EXTRA_LANTAI),
04.              x.getStringExtra(EXTRA_KONSTRUKSI),
05.              x.getStringExtra(EXTRA_PAGAR), x.getStringExtra(EXTRA_CARPORT),
06.              x.getStringExtra(EXTRA_GARASI), x.getStringExtra(EXTRA_TAMAN),
07.              x.getStringExtra(EXTRA_LINGKUNGAN), x.getStringExtra(EXTRA_TAMPAK),
08.              x.getStringExtra(EXTRA_LEBARBANGUNAN), x.getStringExtra(EXTRA_TANAH2),
09.              x.getStringExtra(EXTRA_NILAI), x.getStringExtra(EXTRA_LUAST),
10.              x.getStringExtra(EXTRA_LUASB), x.getStringExtra(EXTRA_ZONA),
11.              x.getStringExtra(EXTRA_ALAMAT),
12.              hitungan, njkp, pbb);
13.      vm.insert(ct);
14.      Toast.makeText(this,"Berhasil disimpan",Toast.LENGTH_SHORT).show();
15.      Intent cc = new Intent("com.example.ryzen.pajakcerdas.actViewDB");
16.      startActivity(cc);
17.  }
```

Gambar 5.22 Potongan Kode Program *doInsert MVVM* Simpan Database

```
01.  public void insert(mCalculateBT ct){
02.      String key = DATA_OBJEK.push().getKey();
03.      HashMap<String, Object> obyekpajak = new HashMap<>();
04.      obyekpajak.put("alamat", ct.getAlamat());
05.      obyekpajak.put("harga", ct.getHitungan());
06.      obyekpajak.put("njkp", ct.getNjkp());
07.      obyekpajak.put("pbb", ct.getPbb());
08.      obyekpajak.put("lantai", ct.getLantai());
09.      obyekpajak.put("konstruksi", ct.getKonstruksi());
10.      obyekpajak.put("pagar", ct.getPagar());
11.      obyekpajak.put("carport", ct.getCarport());
12.      obyekpajak.put("garasi", ct.getGarasi());
13.      obyekpajak.put("taman", ct.getTaman());
14.      obyekpajak.put("lingkungan", ct.getLingkungan());
15.      obyekpajak.put("tensor", ct.getTensor());
16.      obyekpajak.put("lebarbangunan", ct.getLebarbangunan());
17.      obyekpajak.put("tanah", ct.getTanah());
18.      obyekpajak.put("golonganrumah", ct.getNeural());
19.      obyekpajak.put("luastanah", ct.getLuast());
20.      obyekpajak.put("luasbangunan", ct.getLuasb());
21.      obyekpajak.put("zona", ct.getZona());
22.
23.      Map<String, Object> wadah = new HashMap<>();
24.      wadah.put(key, obyekpajak);
25.      DATA_OBJEK.updateChildren(wadah);
26.  }
```

Gambar 5.23 Potongan Kode Program *insert MVVM* Simpan Database

Sedangkan untuk kode program yang mengimplementasikan MVC bisa dilihat pada gambar 5.21 dibawah ini

```
01.  public void do_Insert(double hitungan, double njkp, double pbb){
02.      Intent x = getIntent();
03.      mCalculateBT ct = new mCalculateBT(x.getStringExtra(EXTRA_LANTAI),
04.              x.getStringExtra(EXTRA_KONSTRUKSI),
05.              x.getStringExtra(EXTRA_PAGAR), x.getStringExtra(EXTRA_CARPORT),
06.              x.getStringExtra(EXTRA_GARASI), x.getStringExtra(EXTRA_TAMAN),
07.              x.getStringExtra(EXTRA_LINGKUNGAN), x.getStringExtra(EXTRA_TAMPAK),
08.              x.getStringExtra(EXTRA_LEBARBANGUNAN), x.getStringExtra(EXTRA_TANAH),
09.              x.getStringExtra(EXTRA_NILAI), x.getStringExtra(EXTRA_LUAST),
10.              x.getStringExtra(EXTRA_LUASB), x.getStringExtra(EXTRA_ZONA),
11.              x.getStringExtra(EXTRA_ALAMAT),
12.              hitungan, njkp, pbb);
13.      Controller.insert_Data(ct);
14.      Intent aa = new Intent("com.example.ryzen.pajakcerdas.actViewDB");
15.      startActivity(aa);
16.  }
```

Gambar 5.24 Potongan Kode Program *doInsert MVC* Simpan Database

```
01.  public static void insert_Data(mCalculateBT ct){
02.      DataModel.insert_Data(ct);
03.  }
```

Gambar 5.25 Potongan Kode Program *insert_Data MVC* pada *Controller* Simpan Database

```
01.  public static void insert_Data(mCalculateBT cbt){
02.      String key = DATA_OBJEK.push().getKey();
03.
04.      HashMap<String, Object> obyekpajak = new HashMap<>();
05.      obyekpajak.put("alamat", cbt.getAlamat());
06.      obyekpajak.put("harga", cbt.getHitungan());
07.      obyekpajak.put("njkp", cbt.getNjkp());
08.      obyekpajak.put("pbb", cbt.getPbb());
09.
10.      obyekpajak.put("lantai", cbt.getLantai());
11.      obyekpajak.put("konstruksi", cbt.getKonstruksi());
12.      obyekpajak.put("pagar", cbt.getPagar());
13.      obyekpajak.put("carport", cbt.getCarport());
14.      obyekpajak.put("garasi", cbt.getGarasi());
15.      obyekpajak.put("taman", cbt.getTaman());
16.      obyekpajak.put("lingkungan", cbt.getLingkungan());
17.      obyekpajak.put("tensor", cbt.getTensor());
18.      obyekpajak.put("lebarbangunan", cbt.getLebarbangunan());
19.      obyekpajak.put("tanah", cbt.getTanah());
20.      obyekpajak.put("golonganrumah", cbt.getNeural());
21.      obyekpajak.put("luastanah", cbt.getLuast());
22.      obyekpajak.put("luasbangunan", cbt.getLuasb());
23.      obyekpajak.put("zona", cbt.getZona());
24.
25.      Map<String, Object> wadah = new HashMap<>();
26.      wadah.put(key, obyekpajak);
27.
28.      DATA_OBJEK.updateChildren(wadah);
29.  }
```

Gambar 5.26 Potongan Kode Program *insert_Data* pada *DataModel* Simpan Database

5.3 Implementasi Test Case

Metode yang dilakukan pada pengujian aplikasi *mobile* PajakCerdas adalah metode *Instrumented Unit Testing*, yang artinya *testing* akan berfokus pada kesesuaian jalannya interaksi aplikasi yang telah dikembangkan dengan use case story yang telah dibuat. Dalam pelaksanaan *testing*, *testcase* yang dijalankan akan disesuaikan dengan use case aktor. Pengujian

dilakukan ketika aplikasi telah selesai dikembangkan sesuai kebutuhan fungsional. Dengan dilakukannya *Instrumented Unit Testing*, dapat dipastikan seluruh fungsional aplikasi berjalan dengan baik sesuai dengan kebutuhan fungsional yang telah disepakati. Berikut merupakan hasil *testing* yang dilakukan pada aplikasi *mobile* PajakCerdas yang ditampilkan pada tabel 5.3.

Tabel 5.3 Hasil Testing Role User

Use Case	Identifikasi Pengujian	Skenario	Hasil
U-01	Usecase1	Normal	Passed
	Alternative1.1	Alternatif 1	Passed
U-02	Usecase2	Normal	Passed
	Alternative2.1	Alternatif 1	Passed
U-03	Usecase3	Normal	Passed
	Alternative3.1	Alternatif 1	Passed
U-04	Usecase4	Normal	Passed
	Alternative4.1	Alternatif 1	Passed
U-05	Usecase5	Normal	Passed
U-06	Usecase6	Normal	Passed

Detail mengenai skenario test case terdapat pada lampiran A halaman 98.

5.4 Implementasi Checklist

Metode yang dilakukan pada proses validasi desain adalah dengan metode Checklist yang artinya setiap karakteristik *design pattern* yang telah berhasil diimplementasikan pada masing-masing kode program akan menghasilkan karakteristik yang telah di contreng. Berikut merupakan hasil checklist yang telah diimplementasikan pada masing-masing kode program untuk masing-masing *design pattern* pada aplikasi PajakCerdas yang ditampilkan pada tabel 5.4.Detail mengenai implementasi Checklist terdapat pada lampiran B halaman 103.

Tabel 5.4 Hasil Checklist MVVM

Checklist	Identifikasi Checklist	Hasil
Model	Checklist 1	Passed
	Checklist 2	Passed
	Checklist 3	Passed
	Checklist 4	Passed
View	Checklist 1	Passed
	Checklist 2	Passed
	Checklist 3	Passed
	Checklist 4	Passed
ViewModel	Checklist 1	Passed
	Checklist 2	Passed

Tabel 5.5 Hasil Checklist MVC

Checklist	Identifikasi Checklist	Hasil
Model	Checklist 1	Passed
	Checklist 2	Passed
	Checklist 3	Passed
	Checklist 4	Passed
View	Checklist 1	Passed
	Checklist 2	Passed
	Checklist 3	Passed
	Checklist 4	Passed
Controller	Checklist 1	Passed
	Checklist 2	Passed
	Checklist 3	Passed
	Checklist 4	Passed

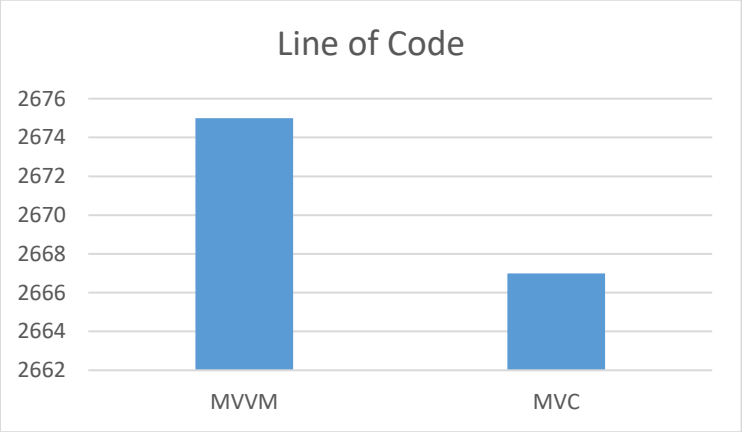
Halaman ini sengaja dikosongkan

BAB VI
HASIL DAN PEMBAHASAN

Bab ini berisikan hasil dan pembahasan penelitian setelah tahap perbandingan antara kedua *design pattern* dilakukan pada aplikasi PajakCerdas.

Seperti yang sudah dituliskan pada bab sebelumnya mengenai metodologi penelitian, pada bab ini akan dibahas hasil yang didapatkan ketika perbandingan menggunakan Software Metric Tools yang bernama JHawk 6.13. Beberapa metrik yang digunakan beserta hasilnya akan disampaikan pada ulasan dibawah ini beserta penjelasan kondisi ideal masing-masing.

6.1 Pengukuran Line of Code



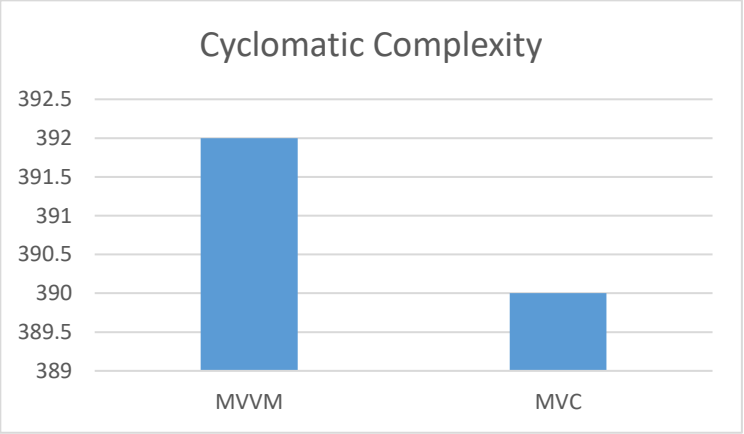
Gambar 6.1 Grafik hasil pengukuran baris kode program

Gambar 6.1 menampilkan data dan hasil dari pengukuran kode baris program dari dua aplikasi yang sama dengan menggunakan pendekatan *design pattern* yang berbeda, MVVM dengan MVC. Dapat dilihat dari gambar diatas bahwa

baris kode program pada aplikasi yang menggunakan *design pattern* MVVM terlihat mempunyai baris kode program yang lebih banyak daripada aplikasi yang menggunakan *design pattern* MVC, dengan jumlah baris kode aplikasi yang menggunakan *design pattern* MVVM sebanyak 2675 sedangkan jumlah baris kode aplikasi yang menggunakan *design pattern* MVC sebanyak 2667.

Hal ini bisa terjadi mengingat komponen-komponen yang ada pada *design pattern* MVVM sedikit lebih banyak daripada komponen yang terdapat pada *design pattern* MVC. Dengan bertambahnya komponen yang ada, maka akan menghasilkan baris kode program yang lebih banyak pula.

6.2 Pengukuran Cyclomatic Complexity

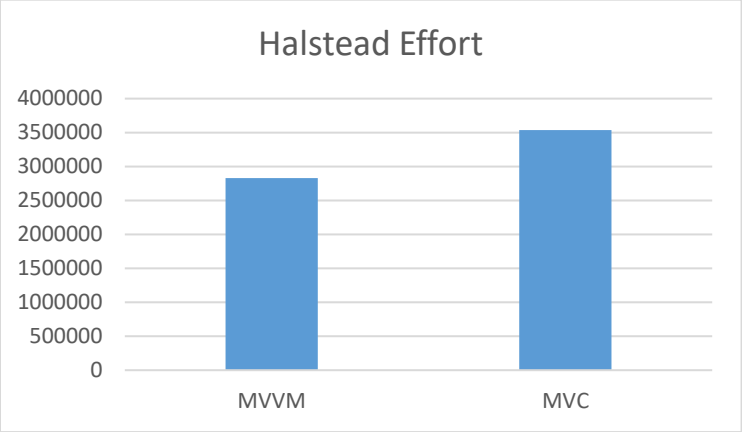


Gambar 6.2 Grafik hasil pengukuran Cyclomatic Complexity

Gambar 6.2 menunjukkan hasil dari pengukuran metrik Cyclomatic Complexity, sebuah metrik yang mengukur kompleksitas kode program. Dari gambar diatas dapat disimpulkan bahwa pada studi kasus ini, aplikasi dengan *design pattern* MVVM mempunyai tingkat kompleksitas kode

program yang lebih tinggi yang berarti berdampak negatif pada proses pembangunan aplikasi[35] dibandingkan aplikasi dengan *design pattern* MVC dengan perbandingan skor sebesar 392 berbanding 390.

6.3 Pengukuran Halstead Effort

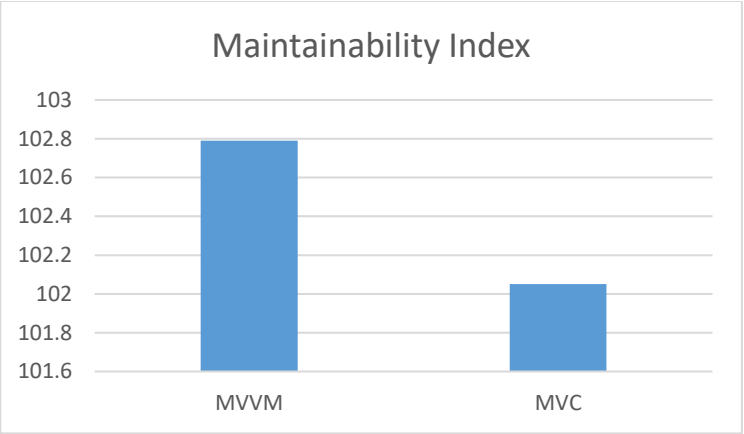


Gambar 6.3 Grafik hasil pengukuran Halstead Effort

Gambar 6.3 menunjukkan hasil pengukuran metrik Halstead Effort, atau metrik yang mengukur seberapa banyak usaha yang dibutuhkan untuk membangun kode program terkait. Semakin kecil nilai halstead effort, semakin mudah untuk sebuah kode program dimodifikasi[32].

Dari gambar diatas, dapat diambil kesimpulan bahwa kode program yang menggunakan pendekatan *design pattern* MVVM memiliki skor yang lebih rendah, dengan skor sebesar 2.831.921,25. Sedangkan aplikasi yang menggunakan pendekatan *design pattern* MVC mempunyai skor yang lebih tinggi, sebesar 3.538.854,15 yang berarti bahwa usaha yang diperlukan untuk membuat kode program aplikasi dengan *design pattern* MVC lebih tinggi dari MVVM.

6.4 Pengukuran Maintainability Index



Gambar 6.4 Grafik hasil pengukuran Maintainability Index

Gambar 6.4 menunjukkan hasil pengukuran dari metrik Maintainability Index. Maintainability Index sendiri mempunyai range nilai 0 sampai dengan 171. Maintainability Index didapat dari metrik-metrik yang sudah disebutkan sebelumnya. Semakin tinggi nilai MI, semakin baik[35]. Dari beberapa metrik yang ada, sebagian besar metrik mendukung bahwa MVVM merupakan *design pattern* yang tepat untuk digunakan pada studi kasus penelitian ini, sedangkan terdapat juga metrik yang mendukung penggunaan MVC. Dari gambar diatas dapat diambil kesimpulan bahwa MVVM mempunyai index maintainabilitas yang lebih tinggi dari MVC dengan perbandingan skor 102,79 berbanding 102,05.

Dengan munculnya nilai Maintainability Index, usai sudah tahapan komparasi dua *design pattern* antara MVVM dengan MVC. Untuk memudahkan visualisasi data komparasi, berikut data hasil komparasi empat metrik seperti yang tercantum pada tabel 6.1 :

Metriks	MVVM	MVC
Line of Code	2675	2667
Cyclomatic Complexity	392	390
Cumulative Halstead Effort	2.831.921,25	3.538.854,15
Maintainability Index	102,79	102,05

Tabel 6.1 Hasil Komparasi

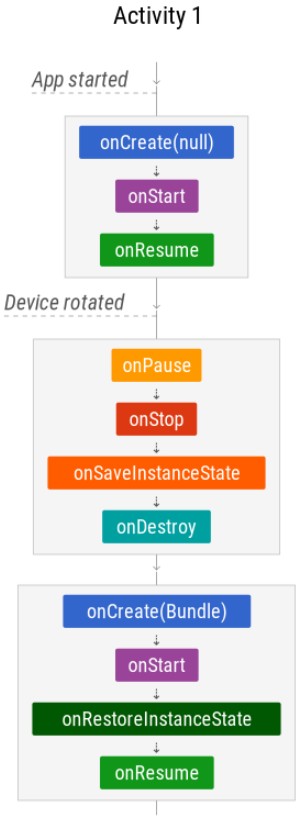
6.5 Pendapat dari Ahli

Selain dari hasil penelitian diatas, penulis merasa untuk perlu memasukkan berbagai pendapat yang kredibel mengenai penggunaan *design pattern* yang tepat pada sebuah aplikasi berbasis Android. Tidak bijak tentunya apabila penulis hanya mendasarkan sebuah justifikasi pada penelitian ini apabila hanya menggunakan satu buah penelitian yang penulis gunakan. Pada sub bab ini, penulis akan melampirkan pendapat dari pihak Google sebagai pengembang utama ekosistem opensource Android untuk mendapatkan informasi yang lebih baik sekaligus mendapatkan pengetahuan dari sisi *expert* tentang penggunaan *design pattern* yang tepat.

Dilansir laman resmi Android Jetpack[36], terdapat beberapa aspek dimana *design pattern* MVVM dinilai lebih cocok untuk diterapkan pada pengembangan aplikasi berbasis Android.

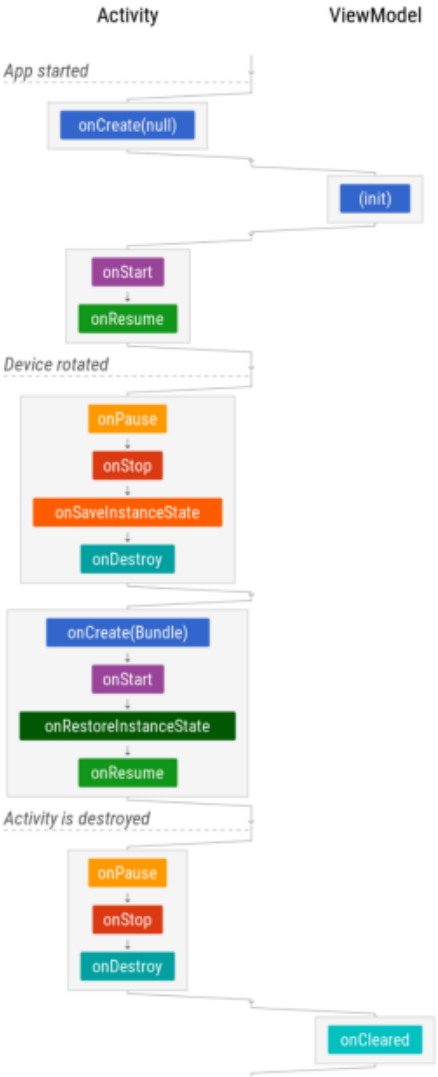
1. *Configuration Changes*

   Perangkat bergerak, termasuk didalamnya perangkat berbasis Android sering mengalami perubahan konfigurasi, entah itu dari internal maupun dari eksternal. Contoh paling sederhana ialah ketika perangkat mengalami perubahan orientasi dari portrait ke landscape atau sebaliknya.



Gambar 6.5 Tipikal lifecycle aplikasi android

Seperti yang dapat kita lihat pada gambar 6.5, setiap kali perangkat mengalami perubahan konfigurasi seperti rotasi, maka sebuah *activity* akan mengalami tahap *onDestroy* untuk kemudian di inisialisasi kembali pada tahap *onCreate*. Dari sini terlihat bahwa aplikasi harus memulai *lifecycle*-nya kembali dari awal.



Gambar 6.6 Tipikal lifecycle aplikasi android dengan MVVM

Gambar diatas adalah *lifecycle* aplikasi android yang telah mengimplementasikan ViewModel yang mana

merupakan komponen dari *design pattern* MVVM. Dengan mengimplementasikan ViewModel, seluruh aktifitas yang dirasa penting dan krusial dapat kita masukkan kedalam ViewModel dengan tujuan ketika *activity* mulai berjalan, ViewModel ikut diinisialisasi bersamaan. Pada saat perangkat mengalami perubahan konfigurasi, yang mengalami tahap *onDestroy* hanyalah bagian *activity* saja, sedangkan ViewModel tetap ada sembari *broadcast* data yang dia punya.

Gambar 6.7 Contoh pengaruh MVVM terhadap Activity Lifecycle (rotasi perangkat)

Pada gambar 6.8, penulis mencoba untuk menjelaskan bagaimana lifecycle aplikasi android berjalan ketika mengimplementasikan *design pattern* MVVM dengan *design pattern* MVC. Seperti yang telah disebutkan pada gambar 6.7, antara *class* Activity dengan *class* ViewModel mempunyai *protected method* khusus yang bisa diimplementasikan untuk meneliti *lifecycle* mereka. Pada kasus ini, method *onCreate* dan *onDestroy* akan digunakan pada *class* Activity, sedangkan method onCleared akan digunakan pada *class* ViewModel. Selanjutnya, akan diterapkan perubahan orientasi untuk membuktikan pengaruh implementasi ViewModel pada aplikasi.

Pada mulanya, ketika sebuah fungsi pada aplikasi membutuhkan *class* Activity dan *class* ViewModel, keduanya akan memulai lifecycle mereka masing-masing yang ditandai dengan dimulainya method onCreate pada *class* Activity, dan inisialisasi pada *class* ViewModel. Setelah kedua *class* tersebut berjalan, ketika terdapat perubahan orientasi seperti contoh dari portrait ke landscape, seperti pada gambar 6.7 pada *class* Activity akan memanggil *method onDestroy*, namun *class* ViewModel tetap berjalan seperti tidak terjadi apapun, ditandai dengan tidak terpanggilnya method *onCleared*. Ketika kedua *class* tersebut sudah tidak terpakai, seperti contoh user mematikan aplikasi, maka onDestroy pada *class* Activity dan onCleared pada *class* ViewModel akan terpanggil, seperti pada gambar 6.8.

ViewModel akan sangat berguna ketika perangkat sudah akan memulai kembali aktifitasnya. Aplikasi tidak perlu memulai ulang seluruh komponen-komponennya, cukup hanya memulai ulang komponen yang berhubungan dengan *view*, untuk kemudian mengambil data secara langsung ke ViewModel.

Akan tetapi, ViewModel akan tetap dapat terhapus dari *lifecycle* aplikasi android apabila sistem memutuskan untuk menutup seluruhnya aplikasi yang bersangkutan, sebagai contoh ketika sistem mendapati keadaan perangkat telah kehabisan sumber daya RAM, atau user menekan tombol “back”.

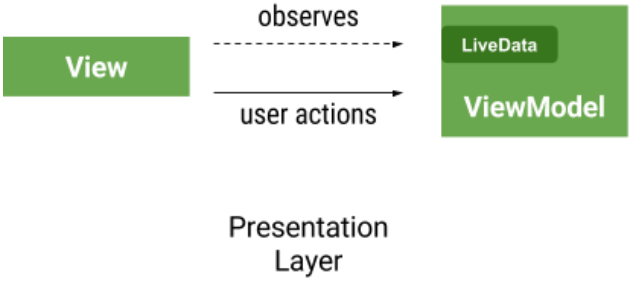
2. View yang *Independent*
   Sama seperti *design pattern* lainnya, pada MVVM komponen *View* dikondisikan sebisa mungkin tidak mengandung logika bisnis dari aplikasi. Lebih lanjut, pada MVVM komponen *View* hanya dikondisikan untuk menjadi penerima sinyal atau data yang di *broadcast* oleh ViewModel.

```
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.recycle_viewdatabase);
    rc = findViewById(R.id.recycler_view);
    vm = ViewModelProviders.of(this).get(myViewModel.class);
    LiveData<mTax> liveData = vm.getDataSnapshotLiveData();
    checkConn();
    liveData.observe(this, new Observer<mTax>() {
        @Override
        public void onChanged(@Nullable mTax mTax) {
            Log.d("observer", "onChanged: VM active");
            if (mTax != null) {
                String alamat = mTax.getAlamat();
                long harga = mTax.getHarga();
                long njkp = mTax.getNjkp();
                Integer pbb = mTax.getPbb();
                mUsername.add(new mTax(alamat, harga, njkp, pbb));
                adapter.setNotes(mUsername);
                adapter.notifyDataSetChanged();
            }
        }
    });
    initRecyclerView();
}
```

Gambar 6.8 View yang bebas dari logika bisnis

3. *Observer Pattern*
   Pada MVVM, komponen *Model* mendapat sedikit keuntungan dari design patten MVVM dikarenakan tugasnya untuk membroadcast data akan dibantu oleh komponen ViewModel. ViewModel sendiri memang memiliki tugas utama sebagai data, yang nantinya akan

ditangkap oleh komponen view manapun yang membutuhkan. Bersamaan dengan implementasi LiveData, ViewModel akan mampu untuk membroadcast data secara *realtime* dan sekaligus mendukung penggunaan database yang bersifat *realtime* seperti Firebase.



Gambar 6.9 MVVM dengan Broadcaster dan Receivernya

```
liveData.observe(this, new Observer<mTax>() {
    @Override
    public void onChanged(@Nullable mTax mTax) {
        Log.d("observer", "onChanged: VM active");
        if (mTax != null) {
            String alamat = mTax.getAlamat();
            long harga = mTax.getHarga();
            long njkp = mTax.getNjkp();
            Integer pbb = mTax.getPbb();
            mUsername.add(new mTax(alamat, harga, njkp, pbb));
            adapter.setNotes(mUsername);
            adapter.notifyDataSetChanged();
        }
    }
});
```

```
01. public class myViewModel extends AndroidViewModel {
02.     @Override
03.     protected void onCleared(){
04.         Log.d("onCleared ", "ViewModel Destroyed");
05.     }
06.     private static final DatabaseReference DATA_OBJEK =
07.             FirebaseDatabase.getInstance().getReference("dataObjek");
08.     private final myLiveData liveData = new myLiveData(DATA_OBJEK);
09.     private final LiveData<mTax> transDeserializer = Transformations.map
10.             (liveData, new Deserializer());
11.     private class Deserializer implements Function<DataSnapshot, mTax> {
12.         @Override
13.         public mTax apply(DataSnapshot hohoho) {
14.             return hohoho.getValue(mTax.class);
15.         }
16.     }
17.     public myViewModel(@NonNull Application application) {
18.         super(application);
19.     }
20.     @NonNull
21.     public LiveData<mTax> getDataSnapshotLiveData() {
22.         return transDeserializer;
23.     }
24. }
```

Gambar 6.10 Implementasi Observer dengan Broadcasternya

4. Anti Null Pointer Exception

Dengan implementasi MVVM secara keseluruhan pada sistem, hampir dapat dipastikan keberadaan error Null Pointer Exception(NPE) akan berkurang drastis. Hal ini bisa terjadi karena akan sangat minim adanya keterikatan antara View dengan Model yang eksplisit. Dengan minimnya hal tersebut, otomatis ketika salah satu antara View atau Model tidak sengaja terhapus dari *lifecycle* aplikasi, kode program tidak akan mencari bagian yang terhapus tersebut secara langsung, melainkan menunggu iterasi *lifecycle* selanjutnya untuk menambal hilangnya data yang diperlukan.

BAB VII
KESIMPULAN DAN SARAN

Bab ini berisikan kesimpulan dari hasil penelitian dan juga saran perbaikan untuk penelitian kedepannya.

7.1 Kesimpulan

Berdasarkan seluruh proses yang telah dilakukan dalam pengerjaan tugas akhir terdapat beberapa kesimpulan yang dapat diambil, diantaranya sebagai berikut :

1. Penggunaan *design pattern* pada penelitian ini menciptakan separasi yang jelas terkait *role* tiap 3 kelas turunan yang dibuat, *class* model hanya bertugas untuk memodifikasi data, *class* view hanya bertugas untuk menampilkan data, dan *class* controller atau viewmodel bertugas untuk menghubungkan model dengan view.
2. Penggunaan MVVM akan banyak memberikan keuntungan apabila aplikasi yang dikembangkan mengandung dan memproses banyak database, sehingga untuk aplikasi dengan akses database yang rutin direkomendasikan untuk menggunakan *design pattern* MVVM.
3. Penggunaan MVC mengurangi banyaknya kode program yang harus dituliskan, sehingga cocok untuk diimplementasikan pada aplikasi dengan proses akses database yang minim.
4. Terdapat *trade-off* mengenai penggunaan salah satu *design pattern* pada studi kasus penelitian ini. Menggunakan *design pattern* MVVM yang notabene lebih baru, meningkatkan banyaknya jumlah kode baris yang harus dituliskan untuk mengimplementasikan *design pattern* tersebut. Namun demikian, dengan menggunakan MVVM, pengembang akan mendapatkan skor Maintainability index yang cukup tinggi yang berarti memudahkan pengembang untuk merawat aplikasinya.

5. Pada MVVM, kode program yang dituliskan lebih banyak dari kode program yang ditulis menggunakan *design pattern* MVC, namun pengukuran halstead effort menunjukkan MVVM lebih rendah daripada MVC, menunjukkan bahwa banyaknya kode program tidak selalu mencerminkan kesulitan yang berlebih bagi *pengembang*.
6. Pada MVC, skor maintainability index yang didapatkan lebih rendah daripada skor yang didapatkan oleh MVVM, menandakan bahwa MVVM mempunyai keunggulan pada aspek pemeliharaan perangkat lunak.
7. Pada aspek kompleksitas kode, MVVM memiliki kelemahan dibandingkan MVC karena skornya yang lebih tinggi, menandakan bahwa kompleksitas kode program pada MVVM lebih tinggi. Untuk itu pengembang yang memutuskan untuk menggunakan MVVM perlu untuk untuk memastikan bahwa kode program berjalan stabil dan tanpa masalah sebelum tahap publikasi, supaya pengembangan aplikasi tidak menjadi sulit kedepannya.

Jadi, implementasi *design pattern* tertentu pada studi kasus ini benar telah memberikan dampak pada proses implementasi kode program aplikasi. Dengan menggunakan design pattern, terdapat separasi yang jelas terhadap *class* turunan kode program yang dibuat.

Terkhusus pada *design pattern* MVVM, implementasi *design pattern* tersebut meningkatkan jumlah penulisan kode dan indeks kompleksitas kode. Namun, implementasi MVVM meningkatkan indeks maintainabilitas kode program, yang memudahkan pengembang untuk merawat aplikasi dan memodifikasi kode programnya. Selain itu, dikarenakan salah satu fokus utama MVVM adalah pada bagian database, untuk aplikasi yang sering mengakses database ataupun mempunyai database yang banyak disarankan untuk menggunakan *design pattern* MVVM, termasuk aplikasi pada studi kasus ini.

Sedangkan pada *design pattern* MVC, implementasinya mengurangi jumlah kode program yang harus dituliskan sekaligus menurunkan indeks kompleksitas kode, sehingga cocok untuk digunakan pada aplikasi yang sederhana.

7.2 Saran

Adapun saran yang diberikan untuk pengembangan selanjutnya adalah sebagai berikut :

1. Berdasarkan pengujian model pada proses *data testing*, akurasi yang didapatkan oleh model yang dibuat pada penelitian ini sudah mencapai lebih dari 80%, akan tetapi hal tersebut belum diuji coba secara langsung pada kasus nyata, perlu diadakan penelitian lebih lanjut untuk mengetahui akurasi model klasifikasi.
2. Data yang digunakan pada proses *training* belum melalui proses *cleaning*, perlu adanya proses tersebut dan penambahan data training yang lebih akurat terhadap lingkup masalah aplikasi.
3. Pengembangan aplikasi *mobile* telah mendapatkan momentumnya beberapa tahun belakangan, ada baiknya apabila pengembang aplikasi *mobile* kedepan tanggap atas perkembangan teknologi terbaru.
4. Untuk pengembangan aplikasi *mobile* berskala yang lebih besar, pengembang aplikasi *mobile* disarankan untuk mencari beberapa penelitian terkait pada ekosistem dan pengembangan aplikasi *mobile* yang akan dibentuk sebelum bergerak pada implementasi kode program.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] StatCounter, "Mobile Operating System Market Share Worldwide," 2019. [Daring]. Tersedia pada: http://gs.statcounter.com/os-market-share/mobile/worldwide. [Diakses: 09-Feb-2019].

[2] 42matters, "Store Stats for Mobile Apps," 2019. .

[3] CrispyCodes, "How Long Does It Take To Build An IOS or Android App?," 2014. [Daring]. Tersedia pada: https://visual.ly/community/infographic/technology/how-long-does-it-take-build-ios-or-android-app. [Diakses: 09-Feb-2019].

[4] K. Yarmosh, "How Often Should You Update Your App?," 2016. [Daring]. Tersedia pada: https://savvyapps.com/blog/how-often-should-you-update-your-app. [Diakses: 09-Feb-2019].

[5] Y. Ma, X. Liu, Y. Liu, Y. Liu, dan G. Huang, "A Tale of Two Fashions: An Empirical Study on the Performance of Native Apps and Web Apps on Android," *IEEE Trans. Mob. Comput.*, vol. 17, no. 5, hal. 990–1003, 2018.

[6] B. S. Panca, S. Mardiyanto, dan B. Hendradjaya, "Evaluation of Software Design Pattern on Mobile Application Based Service Development Related to The Value of Maintainability and Modularity," in *International Conference on Data and Software Engineering*, 2016.

[7] E. Kazan, M. Canturk, dan M. Bastan, "Performance analysis of a software developed with and without design patterns: A case study," *Proc. 2015 12th Int. Conf. Electron. Comput. Comput. ICECCO 2015*, 2016.

[8] Wikipedia, "Software design pattern," 2019. [Daring]. Tersedia pada: https://en.wikipedia.org/wiki/Software_design_pattern. [Diakses: 12-Feb-2019].

[9] C. Anderson, “The Model-View-ViewModel (MVVM) Design Pattern BT - Pro Business Applications with Silverlight 5,” C. Anderson, Ed. Berkeley, CA: Apress, 2012, hal. 461–499.

[10] S. A. Abdallah, R. Moawad, dan E. E. Fawzy, “An Optimization Approach for Automated Unit Test Generation Tools using Multi-Objective Evolutionary Algorithms,” *Futur. Comput. Informatics J.*, hal. 1–13, 2018.

[11] Tian Lou, “A Comparison of Android Native App Architecture – MVC , MVP and MVVM,” 2016.

[12] F. M. Alghamdi dan M. R. J. Qureshi, “Impact of Design Patterns on Software Maintainability,” *Int. J. Intell. Syst. Appl.*, vol. 10, hal. 41–46, 2014.

[13] I. Standards Board, “IEEE Standard Glossary of Software Engineering Terminology,” *IEEE Std 610.12-1990IEEE Std 610.12-1990*, 1990.

[14] N. Medvidovic dan R. N. Taylor, “Software Architecture : Foundations , Theory , and Practice,” *Africa (Lond).*, 2010.

[15] L. Chen, M. A. Babar, dan B. Nuseibeh, “Characterizing architecturally significant requirements,” *IEEE Softw.*, 2013.

[16] A. Shvets, “Design Pattern,” 2017. [Daring]. Tersedia pada: https://sourcemaking.com/design_patterns.

[17] Tung Bui Du, “Reactive Programming and Clean Architecture in Android Development,” no. April, hal. 47, 2017.

[18] G. Prabowo, “RANCANG BANGUN APLIKASI MOBILE CLIENT- SIDE SERVICE DESK ITS BERBASIS ANDROID DESIGN AND DEVELOPMENT OF CLIENT-SIDE ITS SERVICE DESK MOBILE APPLICATION BASED,” Institut Teknologi Sepuluh Nopember Surabaya, 2018.

[19] W. bahasa Indonesia, “Android (sistem operasi),” *Wikipedia*, 2018. [Daring]. Tersedia pada: https://id.wikipedia.org/w/index.php?title=Android_(si

stem_operasi)&stable=1. [Diakses: 16-Feb-2019].
[20] J. Grossman, “Introduction to Model/View/ViewModel pattern for building WPF apps,” 2005. .
[21] Developer.android, “LiveData.” .
[22] Tutorials Point, “SDLC - Overview.” [Daring]. Tersedia pada: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm. [Diakses: 24-Feb-2019].
[23] Tutorials Point, “SDLC - Waterfall Model.” [Daring]. Tersedia pada: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. [Diakses: 24-Feb-2019].
[24] Object Management Group, “What is UML,” 2005. [Daring]. Tersedia pada: http://www.uml.org/what-is-uml.htm. [Diakses: 24-Feb-2019].
[25] Lucid Software, “What is Unified Modeling Language.” [Daring]. Tersedia pada: https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language. [Diakses: 24-Feb-2019].
[26] Virtual Machinery, “JHawk - the Java Metrics tool,” 2017. [Daring]. Tersedia pada: http://www.virtualmachinery.com/jhawkmetrics.htm.
[27] R. Scandariato dan J. Walden, “Predicting vulnerable *class*es in an Android application,” hal. 11, 2012.
[28] J. Börstler, M. E. Caspersen, dan M. Nordström, “Beauty and the Beast: on the readability of object-oriented example programs,” *Softw. Qual. J.*, vol. 24, no. 2, hal. 231–246, 2016.
[29] Tutorials Point, “Software Testing - Overview.” [Daring]. Tersedia pada: https://www.tutorialspoint.com/software_testing/software_testing_overview.htm. [Diakses: 24-Feb-2019].
[30] M. Abadi *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th ${$USENIX$}$ Symposium on Operating Systems Design and Implementation (${$OSDI$}$ 16)*, 2016, hal. 265–283.
[31] V. N. Gudivada dan K. Arbabifard, *Open-Source*

Libraries , Application Frameworks , and Workflow Systems for NLP, 1 ed. Elsevier B.V.

[32] IBM Knowledge Center, "Statistics and metrics," *IBM*. [Daring]. Tersedia pada: https://www.ibm.com/support/knowledgecenter/en/SS3JHP_6.1.0/com.ibm.raa.analyze.doc/common/chalstd.html. [Diakses: 24-Feb-2019].

[33] K. D. Welker, "The Software Maintainability Index Revisited," *CrossTalk*, vol. 14, hal. 18–21, 2001.

[34] "Tensorflow." .

[35] G. Prabowo, "A Tale of Two Development Approach : Empirical Study on The Maintainability and Modularity of Android Mobile Application with Anti-Pattern and Model-View-Presenter Design Pattern," *2018 Int. Conf. Electr. Eng. Informatics*, hal. 149–154, 2018.

[36] "Android Developers." .

BIODATA PENULIS

Penulis bernama lengkap Achmad Farhan Mustaqim, lahir di Surakarta, 20 Januari 1997, merupakan anak pertama dari dua bersaudara. Penulis telah menempuh beberapa jenjang pendidikan formal di beberapa sekolah yaitu: SDIT Nur Hidayah Surakarta, SMP Al-Islam 1 Surakarta, dan SMA Negeri 1 Surabaya. Penulis meneruskan pendidikan di Jurusan Sistem Informasi Institut Teknologi Sepuluh Nopember (ITS) Surabaya pada tahun 2015 pasca lulus dari pendidikan SMA dan terdaftar sebagai mahasiswa dengan NRP 05211540000151. Selama menjadi mahasiswa, penulis aktif dan selalu tertarik mengikuti organisasi kemahasiswaan seperti menjadi staff di Biro Technology Development (TechDev) Himpunan Mahasiswa Sistem Informasi (HMSI) pada tahun 2017, Staff Competition Test Maker Information Systems Expo 2016. Berbagai kegiatan lain yang berkaitan dengan keprofesian juga pernah diikuti, salah satunya adalah kegiatan Indonesia Android Kejar pada tahun 2017 dan Beasiswa Digital Talent Scholarship yang diselenggarakan oleh Kominfo pada tahun 2019. Pada tahun keempat perkuliahan, penulis melaksanakan program magang pada Direktorat Pengembangan Teknologi dan Sistem Informasi (DPTSI) ITS dengan tugas utamanya adalah branding dan security pada web ITS. Penulis dapat dihubungi melalui email farhanjuve@gmail.com

Halaman ini sengaja dikosongkan

LAMPIRAN A : Perancangan Test Case

Pengujian yang digunakan pada penelitian ini adalah pengujian dengan menggunakan metode *Instrumented Unit Testing*. Berikut merupakan rancangan *testcase* pengujian yang dibuat berdasarkan use case yang telah disepakati berdasarkan kebutuhan fungsional dengan aktor bernama User

A. Test Case

Use Case	Identifikasi Pengujian	Skenario	Prosedur Pengujian	Masukan	Hasil Ekspektasi
U-01	Usecase1	Normal	1. User menekan tombol input PBB 2. User mengisi variabel yang diperlukan 3. User menekan tombol Halaman Selanjutnya	Input gambar obyek pajak, mengisi jenis bangunan, jumlah lantai, jenis jalan, ketersediaan pagar, carport, garasi dan	Muncul halaman Hitung Tanah

Use Case	Identifikasi Pengujian	Skenario	Prosedur Pengujian	Masukan	Hasil Ekspektasi
				taman, memasukkan jenis konstruksi dan tanah, dan mengisi lebar bangunan	
	Alternative1.1	Alternatif 1	1. User menekan tombol input PBB 2. User memasukkan variabel tidak lengkap / kosong salah satu 3. User menekan tombol Halaman Selanjutnya	Input variabel (Tidak lengkap / kosong salah satu isian)	Muncul peringatan untuk melengkapi data isian

Use Case	Identifikasi Pengujian	Skenario	Prosedur Pengujian	Masukan	Hasil Ekspektasi
U-02	Usecase2	Normal	1. User masuk pada halaman Hitung Tanah 2. User memasukkan variabel tanah yang dibutuhkan 3. User menekan tombol Simpan Data	Input alamat, Luas Tanah, Luas Bangunan dan Zona Tanah	User masuk kedalam halaman Review Pajak
	Alternative2.1	Alternatif 1	1. User masuk pada halaman Hitung Tanah 2. User memasukkan variabel tidak	Input variabel (Tidak lengkap / kosong salah satu isian)	Muncul peringatan untuk melengkapi data isian

Use Case	Identifikasi Pengujian	Skenario	Prosedur Pengujian	Masukan	Hasil Ekspektasi
			lengkap / kosong salah satu 3. User menekan tombol Simpan Data		
U-03	Usecase3	Normal	1. User menekan tombol Database PBB	-	User masuk ke halaman Database PBB
	Alternative3.1	Alternatif 1	1. User menekan tombol Database PBB 2. Sistem gagal mendapatkan data atau tidak terkoneksi dengan Internet	-	Muncul peringatan Loading

Use Case	Identifikasi Pengujian	Skenario	Prosedur Pengujian	Masukan	Hasil Ekspektasi
U-04	Usecase4	Normal	1. User menekan tombol + / ambil gambar 2. User memberi izin penggunaan kamera / klik tombol Allow 3. User menekan tombol Foto	-	User mendapatkan informasi Gambar dan klasifikasinya
	Alternative4.1	Alternatif 1	1. User menekan tombol + / ambil gambar 2. User tidak memberikan izin / klik tombol Deny	-	Muncul peringatan Kamera Eror

Use Case	Identifikasi Pengujian	Skenario	Prosedur Pengujian	Masukan	Hasil Ekspektasi
U-05	Usecase5	Normal	1. User menekan tombol GPS / ambil lokasi 2. User menekan tombol Allow pada permintaan izin lokasi 3. User menekan tombol Dapatkan Alamat	-	User mendapatkan alamat lokasi terkini
U-06	Usecase6	Normal	1. User menekan tombol Simpan Database		User masuk pada halaman Database PBB

LAMPIRAN B : Checklist

B.1. Checklist MVVM

MVVM			
General	Model	View	ViewModel
• MVVM adalah hasil perkembangan dari pattern MVC dan MVP • Tujuan utama dari MVVM adalah memisahkan pengembangan *User interface* dari pengembangan logika bisnis sebuah aplikasi • Inti dari MVVM ada pada ViewModelnya yang bertugas seperti	• Harus mempunyai Dependent(sama seperti pada MVC) • Harus mampu mem-Broadcast kepada ViewModel • Boleh mempunyai protocol atau method untuk mengatur datanya(sama seperti pada MVC)	• View bertugas sebagai Observer ViewModel • View hanya diperbolehkan mengandung kode presentasional • Tidak boleh mengandung kode query DB (kepunyaan Model) • Tidak boleh mengandung logika	• Harus mengandung kode yang bertugas untuk menyediakan data yang akan digunakan oleh View dan mengandung referensi kepada *class* Model • ViewModel tidak bertugas untuk menampilkan data pada View

MVVM			
General	Model	View	ViewModel
specialized controller yang merubah data dari model kepada view • Membutuhkan komponen tambahan yakni *Observable class*	• Tidak boleh memiliki kode yang bersifat presentasional	bisnis berat(kepunyaan ViewModel)	• ViewModel bertugas membroadcast perkembangan yang terjadi pada Model

B.2. Checklist MVC

MVC			
General	Model	View	Controller
• Implementasi MVC adalah mengenai separasi tugas dan tanggungjawab • Model menggambarkan data, View menggambarkan tampilan dan Controller mengatur komunikasi antar keduanya • View dan Controller harus mengetahui karakter Model, namun Model tidak boleh tau mengenai	• Harus mempunyai Dependent • Harus mampu mem-Broadcast kepada dependentnya • Boleh mempunyai protocol atau method untuk mengatur datanya • Satu Model dapat digunakan oleh beberapa View dan Controller • Tidak boleh menggunakan variabel yang berhubungan erat dengan request end-user (kepunyaan controller). • Tidak boleh memiliki kode yang bersifat	• View dapat memiliki Subview • View hanya diperbolehkan mengandung kode presentasional • Tidak boleh mengandung kode query DB (kepunyaan Model). • Tidak boleh mengandung kode request dari end-user (kepunyaan Controller).	• Harus mengandung kode yang bertugas untuk manipulasi tampilan atau data • Harus menjadi koordinator antara View dengan Model • Tidak boleh mengandung kode query DB (kepunyaan Model). • Tidak boleh mengandung kode presentasional (kepunyaan View).

MVC			
General	Model	View	Controller
View dan Controller-nya	presentasional(kepunyaan View).		