TUGAS AKHIR

MODEL ANALISIS APLIKASI ABSENSI KARYAWAN DI UNIVERSITAS MULTI DATA PALEMBANG MENGGUNAKAN *CLEAN ARCHITECTURE* BERDASARKAN PRINSIP DESAIN *SOLID*

Oleh:

Calvin Saputra 2024240056

PROGRAM STUDI SISTEM INFORMASI
FAKULTAS ILMU KOMPUTER DAN REKAYASA
UNIVERSITAS MULTI DATA PALEMBANG
PALEMBANG

2024

Fakultas Ilmu Komputer dan Rekayasa
Universitas Multi Data Palembang

Program Studi Sistem Informasi
Tugas Akhir Sarjana Komputer
Semester Gasal Tahun 2023/2024

Model Analisis Aplikasi Absensi Karyawan di Universitas Multi Data Palembang Menggunakan *Clean Architecture* Berdasarkan Prinsip Desain *SOLID*

Calvin Saputra 2024240056

Abstrak

Aplikasi Pencatatan Absen Daring atau yang biasa dikenal dengan PANDA merupakan aplikasi presensi yang digunakan oleh karyawan dan dosen di Universitas Multi Data Palembang. Aplikasi tersebut merupakan aplikasi *android* yang dikembagkan dengan menggunakan bahasa pemrograman *Java*. Tujuan penelitian adalah untuk mengetahui seberapa jauh aplikasi PANDA menerapkan kaidah *Clean Architecture* dan prinsip desain *SOLID*. Pada tahap penelitian awal ditemukan bahwa *source code* aplikasi belum menerapkan beberapa *layer* yang sesuai dengan kaidah *Clean Architecture* sehingga menyebabkan setiap komponen tidak tergantung terhadap abstraksi, melainkan dengan implementasi. *So*urce *code* juga belum menerapkan prinsip desain *SOLID* di berbagai *class,* hal ini menyebabkan aplikasi lebih sulit untuk dilakukan *maintenance* karena diperlukan waktu yang lebih untuk memahami struktur kode. Proses *refactor* dilakukan terhadap tujuh *class* yang menyalahi aturan *Single Responsibility Principle* dengan cara memisahkannya menjadi *class* baru yang sesuai dengan tanggung jawab, satu *class* yang menyalahi aturan *Interface Segregation Principle* dengan cara menghapus *function* implementasi yang tidak dipakai, dan dua *class* yang menyalahi aturan *Dependency Inversion Principle* dengan cara di *inject* dengan menggunakan *Hilt*.

Clean Architecture juga dicapai dengan memodifikasi *layer entities* berupa *model* agar tidak terpengaruh dengan pengaruh *framework* luar. *Use Case* layer juga ditambahkan agar dependensi dari *presentation layer* tidak langsung terikat dengan *data layer*, melainkan melalui *domain layer* yang berupa *use case* dan *entities*. Rekomendasi *source code* yang dihasilkan oleh penelitian ini diharapkan dapat mempermudah *developer* selanjutnya dalam melakukan proses pemeliharaaan aplikasi PANDA.

Kata kunci: Aplikasi PANDA, *Clean Architecure*, *SOLID*, *Java*, *Android*, *Android Studio*, *Hilt*.


UNIVERSITAS
MDP

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Clean Architecture merupakan pendekatan desain desain *software* yang bertujuan untuk memisahkan logika bisnis dari suatu aplikasi beserta *framework* dan teknologi luarannya. *Clean Architecture* bertujuan untuk membuat aplikasi menjadi lebih independen dari perubahan di lingkungan luar, sehingga lebih mudah untuk dilakukan pemeliharaan. Menurut Martin (2017), *Clean Architecture* merupakan arsitektur sistem yang mengikuti dependency *rule* yang hanya dapat mengarah ke dalam. *Clean Architecture* merepresentasikan integrasi dari berbagai lapisan arsitektur yang memiliki karakteristik independen dari *framework*, *user interface*, dan *database*, serta dapat diuji (Dumbravan, 2022). *Clean Architecture* dapat diibaratkan sebagai arsitektur dari suatu bangunan, jika batu bata yang membangunnya memiliki kualitas yang buruk maka arsitektur dari bangunan tersebut akan gagal. Tetapi, jika batu bata yang membangunnya telah dibuat dengan baik maka fondasi dari bangunan tersebut akan kokoh dan tahan lama. Analogi fondasi rumah tersebut jika dihubungkan dengan pengembangan *software* sangat berkaitan dengan prinsip desain *SOLID*.

Prinsip desain *SOLID* menjelaskan cara pembagian struktur data dan *function* ke dalam suatu *class* beserta bagaimana *class* tersebut dapat terhubung satu

sama lain, *class* merujuk kepada pengelompokkan *function* dan *data* (Martin, 2017). Dengan menerapkan prinsip desain *SOLID*, aplikasi dapat lebih menoleransi perubahan, dan mudah untuk dipahami.

Penerapan kaidah *Clean Architecture* dan prinsip desain *SOLID* saling melengkapi satu sama lain. Aplikasi yang dikembangkan dengan mengikuti kaidah dan prinsip desain tersebut dapat menciptakan aplikasi yang mudah dipelihara, lebih toleran terhadap perubahan, dan mudah untuk dipahami. Aplikasi yang akan dianalisis merupakan aplikasi milik Universitas Multi Data Palembang.

Universitas Multi Data Palembang pada awalnya merupakan sebuah lembaga kursus terhadap program komputer yang berlokasi di kota Palembang. Untuk memenuhi kebutuhan dan permintaan dari masyarakat yang ingin memiliki suatu sistem pendidikan dengan tekad yang kuat, berdirilah sebuah perguruan tinggi dengan program studi setara dengan Diploma 1 atas dasar Yayasan Multi Data Palembang. Pada 9 April 2021, berdasarkan Surat Keputusan Menteri Pendidikan dan Kebudayaan Republik Indonesia No. 125/E/O/2021 tentang penggabungan AMIK MDP, STMIK GI MDP, dan STIE MDP menjadi Universitas Multi Data Palembang.

Universitas Multi Data Palembang memiliki salah satu aplikasi berbasis *mobile* yang digunakan untuk absensi karyawan yang bernama Aplikasi Pencatatan Abensi Daring (PANDA). Aplikasi tersebut dibangun dengan bahasa pemrograman *Java* dan ditujukan untuk pengguna *smartphone Android*. Penggunaan aplikasi dilakukan dengan melakukan *login* sesuai dengan *username* dan *password* yang telah dibuatkan oleh tim Unit Pelaksana Teknis Sistem Informasi (UPT-SI).

Karyawan perlu berada di daerah gedung Universitas MDP dengan radius minimal enam puluh meter, lalu memilih opsi absensi datang dengan melakukan swafoto wajah. Absensi dilakukan saat jam kedatangan dan jam pulang karyawan.

Aplikasi PANDA dipilih menjadi objek analisis dikarenakan aplikasi ini digunakan sebagai penunjang presensi karyawan dan dosen di Universitas Multi Data Palembang, selain itu berdasarkan observasi awal yang dilakukan terhadap Aplikasi PANDA yang menjadi studi kasus, ditemukan bahwa *source code* aplikasi belum menerapkan *layering* sesuai kaidah *Clean Architecture* yang menyebabkan setiap komponen tidak bergantung terhadap abstraksi, melainkan dengan implementasi dari *class*. *Source code* juga belum menerapkan prinsip desain *SOLID* di berbagai *class.*

Berdasarkan uraian yang telah dijelaskan sebelumnya, dibutuhkan proses analisis berdasarkan *Clean Architecture* dengan menerapkan prinsip desain *SOLID* untuk menghasilkan aplikasi yang telah menerapkan arsitektur dan prinsip desain tersebut. Judul yang diangkat adalah "Model Analisis Aplikasi Absensi Karyawan di Universitas Multi Data Palembang menggunakan *Clean Architecture* berdasarkan Prinsip Desain *SOLID*".

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan sebelumnya, didapat permasalahan sebagai berikut.

1. Aplikasi absensi karyawan belum menerapkan kaidah *Clean Architecture* dan prinsip desain *SOLID*.

1.3 Ruang Lingkup

Dalam penelitian ini ditentukan ruang lingkup atau batasan masalah sebagai berikut.

1. *Refactoring* fitur dilakukan terhadap *Presentation Layer* atau *UI Layer* yang merupakan lapisan yang menampilkan elemen antarmuka pengguna berupa *Activity* atau *Fragment* (Developer, 2023). Proses *refactor* juga dilakukan terhadap *Data Layer* yang merupakan lapisan pemanggilan data, selain itu dilakukan juga *refactor* terhadap *class* RiwayatFragment. Fitur dari aplikasi absensi karyawan di Universitas Multi Data Palembang adalah sebagai berikut.
   a. Fitur tambah absen yang mencakup *function cameraCaptureLauncher*, *startCaptureCamera*, *takePicture*, *takePictureN*, *resizePhoto*, *findFrontFacingCamera*, *decodeBitmapUri*, *getOutputMediaFileUri*, *createImageFile*, *rotateImageIfRequired* pada *class* AbsenActivity.
   b. Fitur *login* yang mencakup *function* updateNomorHpUser, dan *startFirebaseLogin* di *class LoginActivity*.
   c. Fitur pembaruan token yang mencakup *function updateToken* di *class APIService* beserta *function updateTokenUser* di *class LoginActivity*.
   d. Fitur pengambilan absen terlambat di bulan ini yang mencakup *function* getAbsenTerlambatBulanIni di *class APIService* beserta *function* getAbsenTerlambatBulanIni, dan *getUserData* di *class ServiceRepository*.
   e. Fitur pengambilan absen yang mencakup *function* getAbsen di *class APIService*, *function* getAbsenData di *class ServiceRepository*, beserta

function isLastItemDisplaying, *setupRecyclerView*, *onRiwayatItemClick*, *showLoading*, *loadMore*, *load*, *loadMoreOld* di *class* RiwayatFragment.

f. *Class APIService* yang mencakup *function getInboxMessageDetail*, *getInboxMessageDetailNotif*, *getInboxMessage*, *getInboxCount*, dan *getProfile*.

g. *Class Utilities* yang mencakup *function clearUser*, *setValue*, *getValue*, *checkValue*, *getVersionName*, *getApplicationName*, *isMockLocationOn*, *hideKeyboard*, *getListOfFakeLocationAppsFromAll*.

2. *Refactoring source code* akan dilakukan dengan mengikuti kaidah *Clean Architecture* dan menerapkan prinsip desain *SOLID*.
3. Alat bantu yang digunakan dalam proses analisis dan *refactoring* kode aplikasi adalah IDE Android Studio versi *Giraffe* dengan bahasa pemrograman *Java*.

1.4 Tujuan dan Manfaat

Tujuan dari skripsi ini adalah sebagai berikut.

1. Mengetahui sejauh mana penerapan kaidah *Clean Architecture* dan prinsip desain *SOLID* di aplikasi absensi karyawan.
2. Menghasilkan rekomendasi kode aplikasi absensi karyawan yang telah menerapkan *Clean Architecture* dan prinsip desain *SOLID.*

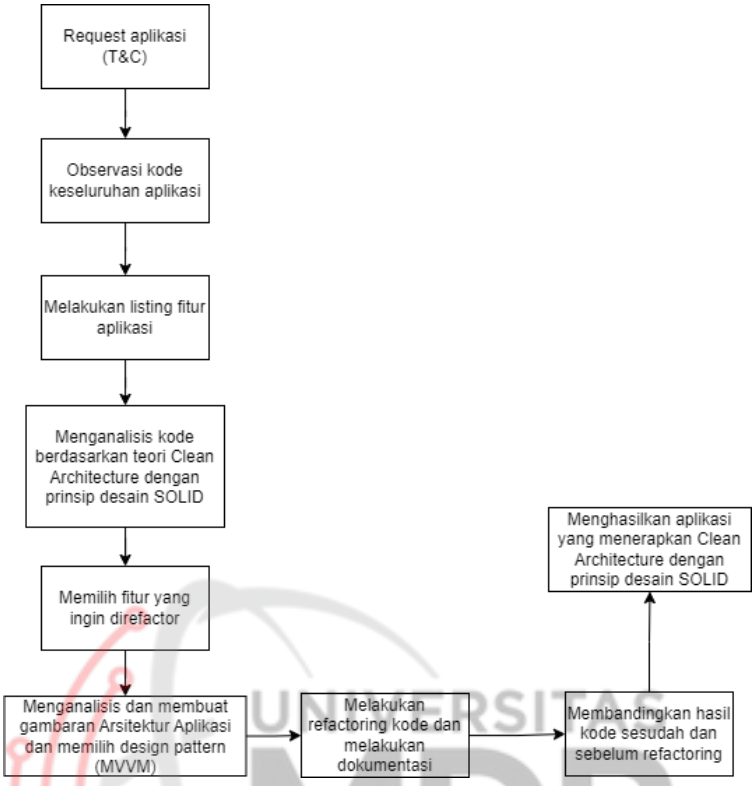
Manfaat dari skripsi ini adalah sebagai berikut.

1. Merekomendasikan *source code* yang menerapkan kaidah *Clean Architecture* dan menerapkan prinsip desain *SOLID.*

2. Menghasilkan *prototype* kode yang telah menerapkan kaidah *Clean Architecture* dan menerapkan prinsip desain *SOLID*.
3. Menghasilkan aplikasi absensi karyawan yang telah menerapkan kaidah *Clean Architecture* dan prinsip desain *SOLID* sehingga proses *maintenance* mudah dilakukan oleh *developer* dari Univeristas Multi Data Palembang.

1.5 Metodologi

Metodologi yang dalam menganalisis aplikasi bersifat linear dari tahap awal sampai dengan akhir. Metodologi ini disesuaikan dengan kebutuhan analisis dari tahap penerimaan aplikasi sampai dengan tahap keluaran yang menghasilkan aplikasi sesuai dengan prinsip *SOLID*. Berikut adalah penjelasan dari setiap tahapan metodologi yang digunakan dalam penelitian ini.



Gambar 1.1 Metodologi Penelitian

1. *Request* aplikasi (T&C)

   Tahap ini bertujuan untuk mengambil aplikasi dari *stakeholder* untuk dilakukan analisis. Pengambilan aplikasi juga disertai dengan prasyarat yang disetujui oleh kedua pihak untuk tidak menyebarluaskan aplikasi ke pihak umum saat dilakukan proses analisis.

2. Observasi kode keseluruhan aplikasi

   Tahap ini berfokus pada mengobservasi keseluruhan kode dari aplikasi yang akan dianalisis dengan tujuan memahami bagaimana gambaran besar aplikasi tersebut berjalan, beserta tujuan dari aplikasi tersebut.

3. Melakukan *listing* fitur aplikasi

Tahap ini bertujuan untuk menghasilkan daftar-daftar fitur yang dimiliki dari aplikasi tersebut. Setiap fitur yang ada di aplikasi akan dikelompokkan berdasarkan kategori tertentu beserta fungsi dari fitur tersebut.

4. Menganalisis kode berdasarkan teori *Clean Architecture* dengan prinsip desain *SOLID*

   Tahap ini berfokus pada proses analisis kode dari aplikasi dengan tujuan memastikan aplikasi tersebut telah menerapkan teori *Clean Architecture* berdasarkan buku *Clean Architecture* buatan Robert C. Martin dan buku *Clean Android Architecture* buatan Alexandru Dumbravan, serta menerapkan prinsip desain *SOLID*.

5. Memilih fitur yang ingin di *refactor*

   Tahap ini bertujuan untuk menghasilkan daftar dari fitur yang ingin di *refactor* berdasarkan analisis kode sebelumnya. Fitur yang dipilih untuk dilakukan *refactor* didasarkan terhadap cakupan kode yang ada pada *class* beserta kesalahan yang ditemui setelah dilakukan identifikasi kode di aplikasi absensi karyawan. Fitur yang dipilih akan di *refactor* berdasarkan teori *Clean Architecture* dengan menerapkan prinsip desain *SOLID*.

6. Menganalisis dan membuat gambaran arsitektur aplikasi dan memilih *design pattern (MVVM)*

   Tahap ini berfokus pada perancangan arsitektur aplikasi beserta memilih *design pattern* yang akan digunakan dalam proses *refactor* yaitu *MVVM* (*Model*, *View*, dan *View Model*)

7. Melakukan *refactoring* kode dan melakukan dokumentasi

Tahap ini berfokus pada proses *refactor* kode dari fitur aplikasi yang telah dipilih sebelumnya. Proses ini akan dilakukan dengan mengikuti kaidah *Clean Architecture* dan menerapkan prinsip desain *SOLID*. Selama proses *refactoring* akan dilakukan dokumentasi terhadap perkembangan kode tersebut.

8. Membandingkan hasil kode sesudah dan sebelum *refactoring*

   Tahap ini bertujuan untuk membandingkan kode yang telah dilakukan proses *refactor* berdasarkan kaidah *Clean Architecture* dan penerapan prinsip desain *SOLID* terhadap kode yang belum dilakukan proses *refactor*.

9. Menghasilkan aplikasi yang menerapkan *Clean Architecture* berdasarkan prinsip desain *SOLID*

   Tahap ini bertujuan untuk menghasilkan keluaran akhir berupa aplikasi yang telah di *refactor* dengan menerapkan kaidah *Clean Architecture* berdasarkan prinsip desain *SOLID* guna menghasilkan kode yang baik dan mudah dipelihara.

1.6 Sistematika Penulisan

Proposal tugas akhir ini memiliki tiga bab yang tersusun secara sistematis. Berikut adalah sistematika penulisan dalam proposoal tugas akhir.

BAB 1 PENDAHULUAN

Bab ini menjelaskan latar belakang, permasalahan yang terjadi pada aplikasi, ruang lingkup, tujuan dan manfaat, metodologi, dan sistematika penulisan.

BAB 2 LANDASAN TEORI

Bab ini menjelaskan uraian teori-teori yang digunakan untuk mendukung laporan proposal tugas akhir ini, dan penelitian terdahulu yang berkaitan.

BAB 3 ANALISIS

Bab ini menjelaskan tentang hasil analisis dari aplikasi. Hasil analisis berupa pengamatan dan masalah yang ditemui dari hasil pengamatan *source code*.

BAB 4 *REFACTOR CODE*

Bab ini menjelaskan tentang hasil *refactor code* dari aplikasi. Hasil *refactor code* dibuat berdasarkan prinsip desain *SOLID* dan kaidah *Clean Architecture*.

BAB 5 PENUTUP

Bab ini menjelaskan kesimpulan dan saran penulis terkait laporan Tugas Akhir di Universitas Multi Data Palembang.


UNIVERSITAS
MDP

DAFTAR PUSTAKA

Abdelhalim, E. A., & El Khayat, G. A. (2016). A utilization-based genetic algorithm for solving the university timetabling problem (UGA). *Alexandria Engineering Journal*, *55*(2), 1395–1409. https://doi.org/10.1016/j.aej.2016.02.017

Akkan, C., & Gülcü, A. (2018). A bi-criteria hybrid genetic algorithm with robustness objective for the course timetabling problem. *Computers and Operations Research*, *90*, 22–32. https://doi.org/10.1016/j.cor.2017.09.007

de Oliveira, L. L., Freitas, A. A., & Tinós, R. (2018). Multi-objective genetic algorithms in the study of the genetic code's adaptability. *Information Sciences*, *425*, 48–61. https://doi.org/10.1016/j.ins.2017.10.022

Febrita, R. E., & Mahmudy, W. F. (2017). Modified genetic algorithm for high school time-table scheduling with fuzzy time window. *International Conference on Sustainable Information Engineering and Technology*, 88–92.

Gao, S., & Silva, C. W. De. (2016). A modified estimation distribution algorithm based on extreme elitism. *BioSystems*, *150*, 149–166. https://doi.org/10.1016/j.biosystems.2016.10.001

Ghasemi, E., Moradi, P., & Fathi, M. (2015). Integrating ABC with genetic grouping for university course timetabling problem. *2015 5th International Conference on Computer and Knowledge Engineering (lCCKE)*, 24–29.

Jafari-Marandi, R., & Smith, B. K. (2017). Fluid genetic algorithm (FGA). *Journal of Computational Design and Engineering*, *4*(2), 158–167. https://doi.org/10.1016/j.jcde.2017.03.001

Lei, Y., Shi, J., & Yan, Z. (2018). A memetic algorithm based on MOEA/D for the examination timetabling problem. *Soft Computing*, *22*(5), 1511–1523. https://doi.org/10.1007/s00500-017-2886-y

Lewis, R., & Thompson, J. (2014). Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. *European Journal of Operational Research*, *240*(3), 637–648. https://doi.org/10.1016/j.ejor.2014.07.041

Liu, W., Zhu, H., Wang, Y., Zhou, S., Bai, Y., & Zhao, C. (2013). Topology optimization of support structure of telescope skin based on bit-matrix representation NSGA-II. *Chinese Journal of Aeronautics*, *26*(6), 1422–1429.

https://doi.org/10.1016/j.cja.2013.07.046

Mahiba, A. A., & Durai, C. A. D. (2012). Genetic algorithm with search bank strategies for university course timetabling problem. *Procedia Engineering*, *38*, 253–263. https://doi.org/10.1016/j.proeng.2012.06.033

Parera, S., Sukmana, H. T., & Wardhani, L. K. (2016). Application of genetic algorithm for class scheduling (case study: faculty of science and technology UIN Jakarta). *2016 4th International Conference on Cyber and IT Service Management*, 1–5. https://doi.org/10.1109/CITSM.2016.7577525

Yousef, A. H., Salama, C., Jad, M. Y., El-gafy, T., Matar, M., & Habashi, S. S. (2016). A GPU based genetic algorithm solution for the timetabling problem. *2016 11th International Conference on Computer Engineering & Systems (ICCES)*, 103–109. https://doi.org/10.1109/ICCES.2016.7821982