Penerapan Prinsip Desain Solid Dalam Pengembangan Aplikasi Happy Thrifting Berbasis Web

Dini Amy Narti[1], Depandi Enda[2]

[1,2] *Program Studi Teknik Informatika - Politeknik Negeri Bengkalis*

Jl. Bathin Alam – Bengkalis Riau - Indonesia

dinidiniamynarti@gmail.com, depandienda@polbeng.ac.id

Abstrak— **Pengembangan perangkat lunak modern menuntut kualitas kode yang tinggi, fleksibel, dan mudah dipelihara. Namun, dalam praktiknya masih banyak aplikasi yang memiliki struktur kode tidak terorganisir dan tanggung jawab ganda dalam satu komponen, sehingga menyulitkan proses pemeliharaan. Penelitian ini bertujuan untuk menerapkan prinsip desain SOLID, yaitu *Single Responsibility, Open Closed, Liskov Substitution, Interface Segregation*, dan *Dependency Inversion,* pada aplikasi *Happy Thrifting* sebagai platform jual beli pakaian bekas berbasis web guna meningkatkan kualitas desain perangkat lunak. Metodologi penelitian yang digunakan meliputi identifikasi permasalahan pada sistem awal, pengumpulan data, *refaktorisasi* kode menggunakan *framework* Laravel, serta evaluasi hasil implementasi melalui metode *static code analysis* dengan bantuan *tools PHP Metrics.* Hasil penelitian menunjukkan bahwa penerapan prinsip SOLID menghasilkan struktur aplikasi yang lebih terorganisir dengan pemisahan tanggung jawab yang jelas antar komponen. Evaluasi kualitas kode menunjukkan penurunan tingkat ketergantungan antar kelas dengan nilai *Average Efferent Coupling* sebesar 3,45 serta peningkatan kohesi metode dengan nilai LCOM sebesar 2,28. Selain itu, nilai *Average Cyclomatic Complexity* sebesar 5,45 menunjukkan bahwa sistem berada pada kategori risiko rendah dan memiliki prosedur yang stabil. Dengan demikian, penerapan prinsip desain SOLID mampu meningkatkan kualitas desain perangkat lunak dan kemudahan pemeliharaan sistem**

Kata Kunci— **SOLID, Refaktorisasi, Laravel, Maintainability, *PHP Metrics*.**

Abstract— **Modern software development demands high code quality, flexibility, and easy to maintain. However, in practice there are still many applications that have an unorganized code structure and dual responsibilities in a single component, making the maintenance process difficult. This research aims to apply the SOLID design principles, namely *Single Responsibility, Open Closed, Liskov Substitution, Interface Segregation*, and *Dependency Inversion,* to the *Happy Thrifting* application as a web-based used clothing buying and selling platform to improve the quality of software design. The research methodology used includes identifying problems in the initial system, collecting data, *refactoring* code using the Laravel framework, and evaluating the implementation results through *the static code analysis* method with the help of *PHP Metrics tools.* The results show that the application of the SOLID principle results in a more organized application structure with a clear separation of responsibilities between components. The code quality evaluation showed a decrease in the level of dependency between classes with *an Average Efferent Coupling* value of 3.45 and an increase in method cohesion with an LCOM value of 2.28. In addition, an *Average Cyclomatic Complexity* value of 5.45 indicates that the system is in the low-risk category and has stable procedures. Thus, the application of SOLID design principles is able to improve the quality of software design and ease of system maintenance**

Keywords— **SOLID, Refactorization, Laravel, Maintainability, *PHP Metrics*.**

I. PENDAHULUAN

Seiring dengan berkembangnya era digital ini, perkembangan sistem perangkat lunak dituntut untuk memiliki kualitas yang baik, fleksibel dalam melakukan pengembangan, serta mudah dalam melakukan pemeliharaan. Namun, pada kenyataannya, masih banyak pengembang atau developer perangkat lunak yang meghasilkan kode program yang tidak terstruktur, sulit dibaca, serta menyulitkan proses pengembangan lebih lanjut. Hal ini disebabkan oleh kurangnya pemahaman dan penerapan prinsip desain yang tepat dalam proses pengembangan perangkat lunak.

Salah satu solusi untuk mengatasi permasalahan tersebut adalah dengan menerapkan prinsip desain SOLID. Prinsip SOLID merupakan seperangkat pemodan dalam pemograman beriorientasi objek. Maka dengan menerapkan prinsip SOLID ini pengembang dapat membangun struktur kode yang bersih, terorganisir, dan mudah untuk diuji. Sehingga mampu meningkatkan efisiensi pemeliharaan serta kualitas perangkat lunak secara keseluruhan [1].

SOLID *Principle* pertama kali diperkenalkan oleh *Robert C. Martin (Uncle Bob)* sebagai panduan desain dalam pemrograman berorientasi objek. SOLID Principle ini memiliki lima prinsip utama yaitu *Single Responsibility, Open/Closed, Liskov Subsitution, Interface Segregation, dan Depedency Inversion* bertujuan untuk membat asrsitekture perangkat lunak yang lebih tangguh terhadap perubahan, serta memudahkan kalaborasi dalam melakukan pengambangan. Penerapan prinsip ini terbukti mampu meningkatkan keterbacaan, pengujian, dan pemeliharaan perangkat lunak, terutama pada project bersekala menengah hingga besar.

Prinsip desain SOLID telah banyak digunakan untuk pengembangan perangkat lunak modren untuk menciptakan

sistem yang modular dan fleksibel. Pendekatan ini memberikan panduan dalam penyusunan kode yang lebih terstruktur, sehingga memudahkan proses perawatan dan dan pengembangan lebih lanjut. Dengan menerapkan prinsip SOLID, perubahan pada sistem dapat diakomodasi tanpa menimbulkan ketergantungan kompleks antar komponen.[2]

Penelitian ini dilakukan dengan studi kasus pada aplikasi *Happy Thrifting*, yaitu sebuah platform berbasis web untuk jual beli baju bekas secara online. Pada sistem yang ada sebelumnya, struktur kode yang digunakan belum tertata dengan baik, belum modular, dan masih sulit dipahami. Pada skripsi ini dilakukan pengembangan ulang struktur kode dengan menerapkan prinsip desain SOLID, sehingga menghasilkan kode yang lebih bersih, modular, dan mudah diuji. Penerapan ini juga meningkatkan keteraturan struktur kode, mempermudah proses pemeliharaan, serta mendukung pengembangan sistem secara berkelanjutan.

II. KAJIAN PUSTAKA

Penelitian Naveen Chikkanayakanahalli membahas penerapan prinsip desain SOLID (karya Robert C. Martin) untuk mengatasi kompleksitas dalam pengembangan perangkat lunak. Penelitian ini memberikan panduan praktis menggunakan contoh kode C# untuk membantu *developer* membangun sistem yang lebih jelas, fleksibel, dan mudah dipelihara di skenario dunia nyata [1].

Penelitian Usman di Dinas Kominfo Musi Banyuasin menerapkan prinsip SOLID dan optimalisasi *query* pada aplikasi pencatatan kemiskinan. Hasil pengujian menunjukkan bahwa refaktorisasi ini menghasilkan baris kode yang lebih sedikit, sehingga aplikasi menjadi lebih modular, efisien, dan mudah dikelola [2].

Penelitian Manalu Josua menganalisis dan merefaktor aplikasi absensi PANDA berbasis Android (Java) menggunakan *Clean Architecture* dan prinsip SOLID. Dengan memisahkan tanggung jawab *class*, menghapus fungsi yang tidak perlu, dan menggunakan injeksi dependensi Hilt, struktur kode menjadi lebih rapi dan mempermudah *developer* dalam pengembangan ke depan. [3].

Senada dengan sebelumnya, penelitian Manalu juga merefaktor aplikasi absensi Universitas Multi Data Palembang menggunakan *Clean Architecture* dan prinsip SOLID. Pemecahan fungsi ke dalam *class* yang tepat dan pemisahan *layer* domain membuat *presentation layer* tidak bergantung langsung pada data, sehingga menghasilkan kode yang sedikit dan mudah dipelihara [4].

Penelitian Ivan Yanakiev di Belanda berfokus pada pengurangan utang teknis pada sistem perangkat lunak yang telah dibangun selama dua dekade. Dengan menggunakan analisis prioritas berbasis data dan prinsip SOLID (terutama *Dependency Inversion*), tim berhasil menerapkan pola refaktorisasi secara sistematis pada lebih dari 5.000 *file* [5].

Penelitian Bhaumik Tyagi mengkaji pentingnya prinsip desain SOLID untuk menjaga pemeliharaan, penggunaan ulang, dan skalabilitas perangkat lunak di era *Agile*. Eksperimen ini membuktikan keandalan prinsip SOLID dengan menerapkannya pada sebuah prototipe fungsional yang kemudian dievaluasi kualitasnya menggunakan analisis metrik CKJM [6].

Penelitian Laela merancang sistem informasi penjualan *thrifting* berbasis *web* untuk platform Squad 1994 menggunakan metode *Agile* (DSDM). Sistem ini dibuat untuk menggantikan pemasaran manual via Instagram guna mencegah kerugian, penipuan, dan kesalahan rekapitulasi data, sehingga transaksi menjadi jauh lebih efisien [7].

Penelitian Faqih mengeksplorasi perancangan UI/UX aplikasi *mobile thrifting* menggunakan metode *Design Thinking* untuk mendorong pengurangan limbah pakaian bekas. Hasil uji coba menunjukkan bahwa pengguna merespons positif tampilan yang bersih, navigasi intuitif, serta fitur *flash sale* dan donasi pakaian [8].

Penelitian Sinatria menerapkan Clean Architecture dan prinsip SOLID pada pengembangan aplikasi mobile dengan framework Flutter dan metode Agile Scrum. Penggunaan arsitektur ini sangat krusial dalam struktur Flutter untuk mencegah bug yang tidak disengaja dan meminimalkan pengeluaran biaya pemeliharaan sistem [9].

Penelitian Nugroho mengevaluasi refaktorisasi aplikasi XYZ (Android/Kotlin) menggunakan *Clean Architecture* dan prinsip SOLID. Hasilnya menunjukkan penurunan rata-rata kompleksitas kode dari 1,5 menjadi 1,4 serta menjaga stabilitas penggunaan CPU dan memori, membuktikan bahwa pendekatan ini membuat kinerja perangkat lunak lebih efisien[10].

III. METODE PENELITIAN

A. Data dan Alat Penelitian

Penelitian ini menggunakan data primer berupa dokumentasi kode program dan arsitektur dari sistem lama aplikasi *Happy Thrifting* yang belum menerapkan prinsip *Solid Object-Oriented Design*. Lingkungan pengembangan dan pengujian yang digunakan meliputi perangkat keras dengan spesifikasi prosesor Intel Core i5, RAM 8GB, dan sistem operasi Windows 10 Pro. Perangkat lunak utama yang digunakan mencakup *framework* Laravel dengan bahasa pemrograman PHP, basis data MySQL (XAMPP), serta *Visual Studio Code* sebagai *text editor*.

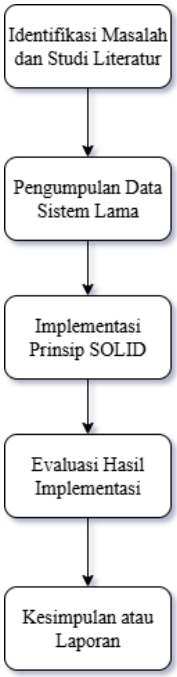
B. Tahapan Penelitian

Metodologi dalam penelitian ini dirancang dalam lima tahapan sistematis untuk mencapai tujuan refaktorisasi perangkat lunak:

1. Identifikasi Masalah dan Studi Literatur: Menganalisis kelemahan pada struktur kode lama (sistem monolitik pada *controller*) dan mengkaji literatur terkait prinsip desain SOLID.
2. Pengumpulan Data Sistem Lama: Mengumpulkan berkas *source code* aplikasi awal yang menunjukkan tingginya tingkat ketergantungan antar komponen (*high coupling*).
3. Implementasi Prinsip SOLID: Melakukan refaktorisasi kode program. Proses ini memisahkan

logika bisnis dari *controller* ke dalam lapisan *Service*, *Repository*, dan *Contracts* (*Interface*) untuk memenuhi prinsip *Single Responsibility*, *Open/Closed*, *Liskov Substitution*, *Interface Segregation*, dan *Dependency Inversion*.

4. Evaluasi Hasil Implementasi: Melakukan *static code analysis* menggunakan perangkat bantu *PHP Metrics* untuk membandingkan kualitas kode sebelum dan sesudah refaktorisasi.
5. Kesimpulan: Menyusun pelaporan hasil metrik secara kuantitatif.



Gbr. 1 Prosedur Penelitian

C. Perancangan Sistem dan Arsitektur

Pada aplikasi *Happy Thrifting* versi awal, sistem belum menerapkan prinsip perancangan perangkat lunak yang baik. Sebagian besar logika aplikasi terpusat secara langsung di dalam *controller* (*fat controller*), yang menyebabkan satu kelas memiliki tanggung jawab ganda, mulai dari validasi data, pengolahan logika bisnis, hingga akses langsung ke basis data (*model*).

Untuk mengatasi masalah tersebut, sistem diusulkan untuk dirancang ulang dengan menerapkan prinsip SOLID. Arsitektur aplikasi dipecah menjadi beberapa lapisan (*layer*):

1. Layer Http/Controllers: Bertanggung jawab secara eksklusif dalam menangani *request* dan *response* dari pengguna (Penerapan *Single Responsibility Principle*).
2. Layer Services: Berperan dalam mengeksekusi logika bisnis inti aplikasi (Penerapan *Dependency Inversion Principle*).
3. Layer Repositories & Contracts: Difokuskan pada proses akses dan manipulasi data melalui *interface* yang spesifik, sehingga sistem tidak bergantung pada implementasi kelas konkret (Penerapan *Interface Segregation* dan *Open/Closed Principle*).

D. Skenario Pengujian

Evaluasi pada penelitian ini dilakukan dengan metode analisis statis (*static code analysis*) tanpa menjalankan aplikasi (secara *runtime*). Pengujian berfokus pada perbandingan struktur hierarki kode sebelum dan sesudah penerapan SOLID dengan memanfaatkan alat bantu *PHP Metrics*. Parameter metrik yang dievaluasi mencakup:

1. *Lines of Code* (LOC): Menilai efisiensi dan perubahan jumlah baris kode.
2. *Cyclomatic Complexity* (CC): Mengukur tingkat kompleksitas alur logika program.
3. *Class Coupling*: Menghitung *Efferent* dan *Afferent Coupling* untuk melihat tingkat ketergantungan antar kelas.
4. *Maintainability Index* (MI) & *Lack of Cohesion of Methods* (LCOM): Menilai kemudahan pemeliharaan dan keterpaduan fungsional di dalam setiap kelas.

IV. HASIL DAN PEMBAHASAN

A. Implementasi Prinsip Desain SOLID

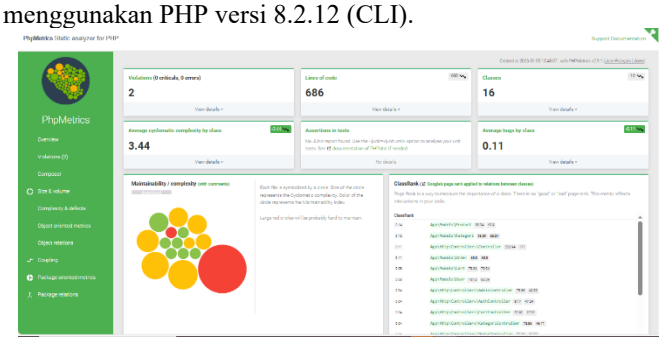
Implementasi pada penelitian ini difokuskan pada perbaikan struktur kode aplikasi *Happy Thrifting* melalui proses refaktorisasi pada modul-modul utama (produk, keranjang, pesanan, dan transaksi) tanpa mengubah fungsionalitas utama sistem. Struktur kode yang sebelumnya terpusat pada *controller* dipisahkan menjadi beberapa lapisan, yaitu *Service* untuk logika bisnis, *Repository* untuk pengelolaan data, dan *Contracts* sebagai *interface*. Penerapan kelima prinsip desain tersebut adalah sebagai berikut:

1. Single Responsibility Principle (SRP): Tanggung jawab *controller* dibatasi hanya untuk menangani *request* dan *response* HTTP. Logika bisnis yang kompleks, seperti manipulasi data dan manajemen stok yang sebelumnya berada di ProductController, dipindahkan ke dalam kelas khusus seperti ProductService. Hal ini juga diterapkan pada SolidCartController dan SolidOrderController.
2. Open/Closed Principle (OCP): Diterapkan melalui penggunaan abstraksi pada lapisan *Repository*. Sebagai contoh, penambahan fitur mekanisme *cache* dilakukan dengan membuat kelas baru CachedProductRepository yang mengimplementasikan *interface* yang sama, tanpa harus memodifikasi kelas ProductRepository yang sudah stabil.
3. Liskov Substitution Principle (LSP): Diterapkan melalui penggunaan ReadProductRepositoryInterface. Setiap kelas turunan (seperti repositori produk umum maupun repositori berbasis *cache*) dapat saling menggantikan tanpa menyebabkan kegagalan fungsional pada lapisan *service*.

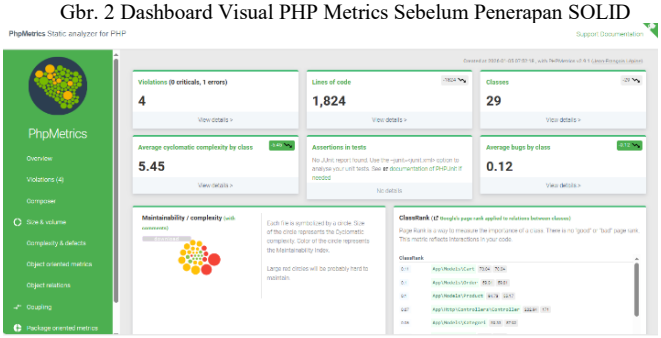
4. Interface Segregation Principle (ISP): Dilakukan dengan memecah *interface* yang besar menjadi antarmuka yang lebih spesifik. Contohnya, pemisahan fungsi pembacaan data ke dalam ReadProductRepositoryInterface dan pemisahan operasi manipulasi data (seperti *create*, *update*, *delete*, *updateStock*) ke dalam ProductRepositoryInterface. Hal ini mencegah terjadinya *fat interface*.
5. Dependency Inversion Principle (DIP): Modul tingkat tinggi seperti CartService tidak lagi bergantung secara langsung pada modul tingkat rendah (kelas *repository* konkret), melainkan pada abstraksi CartRepositoryInterface melalui *Dependency Injection* pada konstruktor. Pengikatan (*binding*) antara *interface* dan implementasi konkretnya diatur melalui RepositoryServiceProvider.

B. *Evaluasi Hasil Penerapan SOLID Menggunakan PHP Metrics*

Evaluasi dilakukan dengan membandingkan kualitas struktur kode sebelum dan sesudah refaktorisasi menggunakan alat *PHP Metrics*. Pengujian dieksekusi melalui terminal menggunakan PHP versi 8.2.12 (CLI).



Gbr. 2 Dashboard Visual PHP Metrics Sebelum Penerapan SOLID



Gbr. 3 Dashboard Visual PHP Metrics Sesudah Penerapan SOLID

Hasil perbandingan metrik kualitas kode dapat dilihat secara rinci pada tabel berikut:

TABEL I
PERBANDINGAN METRIK SEBELUM DAN SESUDAH PENERAPAN SOLID

Metrik Evaluasi	Sebelum	Sesudah	Indikator Ideal
Lines of Code (LOC)	686	1,824	-
Jumlah Class	16	29	-
Avg. Cyclomatic Complexity	3.44	5.45	1 - 10 (Risiko Rendah)
Avg. Efferent Coupling	3.86	3.45	Semakin Rendah Baik
LCOM	2.54	2.28	Semakin Rendah Baik
Avg. Bugs by Class	N/A	0.12	Semakin Rendah Baik
Jumlah Violations	N/A	4	Semakin Rendah Baik

TABEL II
ANALISIS PERUBAHAN STRUKTUR KODE

Metrik	Kondisi Awal (Sebelum SOLID)	Dampak Perubahan (Sesudah SOLID)
LOC & Class	Kode menumpuk (Fat Class) dan menangani banyak tanggung jawab.	Peningkatan jumlah akibat pemisahan fungsi yang spesifik (efek SRP).
Complexity	Logika berpotensi menjadi sangat rumit jika fitur ditambah.	Tetap stabil pada kategori risiko rendah, prosedur lebih terstruktur.
Coupling	Ketergantungan antar class tinggi, rentan rusak jika dimodifikasi.	Ketergantungan menurun, kode menjadi jauh lebih fleksibel.
LCOM	Metode dalam satu class kurang berkaitan (kohesi rendah).	Kohesi meningkat, metode dalam class lebih relevan dan spesifik.
Bugs & Violations	Risiko sulit diprediksi karena tidak ada standarisasi sistematis.	Potensi kesalahan menurun; pelanggaran bersifat ringan dan aman.

C. *Pembahasan Hasil Evaluasi*

Berdasarkan hasil analisis *PHP Metrics*, penerapan arsitektur berlapis berbasis prinsip SOLID memberikan dampak nyata yang signifikan terhadap fleksibilitas dan keterkelolaan sistem.

Terdapat peningkatan pada kuantitas elemen kode, di mana *Lines of Code* (LOC) meningkat dari 686 menjadi 1.824 baris, serta jumlah *class* bertambah dari 16 menjadi 29 kelas. Peningkatan ini merupakan dampak langsung dari penerapan prinsip SRP dan ISP yang memecah fungsi-fungsi kompleks ke dalam kelas dan *interface* yang lebih spesifik. Peningkatan kuantitas kode demi mencapai modularitas yang lebih baik ini sejalan dengan teori bahwa teknik *refactoring* merupakan kunci perbaikan struktur kode internal, meskipun berakibat pada penambahan elemen kode secara kuantitatif.

Peningkatan kualitas struktur perangkat lunak dibuktikan melalui penurunan nilai *Average Efferent Coupling* dari 3,86 menjadi 3,45. Penurunan nilai ini mengindikasikan berkurangnya tingkat ketergantungan antar kelas, sehingga perubahan pada salah satu modul tidak lagi memicu efek berantai yang berpotensi merusak fungsi lainnya.

Selain itu, nilai *Lack of Cohesion of Methods* (LCOM) mengalami penurunan dari 2,54 menjadi 2,28. Angka ini menunjukkan bahwa keterpaduan fungsional di dalam setiap kelas meningkat; setiap metode di dalam sebuah kelas kini memiliki tanggung jawab yang lebih terfokus dan relevan satu sama lain.

Pada aspek kompleksitas, nilai rata-rata *Cyclomatic Complexity* mengalami peningkatan menjadi 5,45. Meskipun terjadi peningkatan, nilai tersebut tetap berada dalam rentang 1-10 yang menurut standar metrik dikategorikan sebagai tingkat risiko rendah, di mana prosedur program masih tergolong sederhana, terstruktur, dan stabil. Penggunaan *interface* untuk abstraksi (DIP) juga terbukti memudahkan perawatan (*maintainability*) aplikasi di masa mendatang secara terukur.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Penerapan prinsip desain SOLID (*Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation*, dan *Dependency Inversion*) pada pengembangan aplikasi *Happy Thrifting* berbasis web telah berhasil memperbaiki struktur kode secara signifikan. Sistem yang sebelumnya tidak terorganisir dan memiliki tanggung jawab ganda pada komponennya, kini menjadi jauh lebih modular, terstruktur, dan mematuhi kaidah rekayasa perangkat lunak yang baik. Berdasarkan hasil analisis *static code* menggunakan *PHP Metrics*, pemecahan fungsi ke dalam kelas yang spesifik telah meningkatkan kuantitas *Lines of Code* (LOC) dari 686 menjadi 1.824 baris dan jumlah kelas dari 16 menjadi 29 kelas. Meskipun demikian, kualitas kode terbukti meningkat yang ditunjukkan oleh penurunan *Average Efferent Coupling* dari 3,86 menjadi 3,45 (ketergantungan kelas menurun) serta penurunan *Lack of Cohesion of Methods* (LCOM) dari 2,54 menjadi 2,28 (kohesi kelas meningkat). Selain itu, nilai *Average Cyclomatic Complexity* berada di angka 5,45 yang mengindikasikan bahwa prosedur program tetap berada pada kategori risiko rendah dan terkontrol. Secara keseluruhan, penerapan SOLID menghasilkan kode dengan tingkat *maintainability* tinggi, sehingga sistem siap untuk dikembangkan secara berkelanjutan dan efisien.

Untuk pengembangan sistem dan penelitian di masa mendatang, aplikasi *Happy Thrifting* disarankan untuk berekspansi ke dalam bentuk aplikasi *mobile* guna meningkatkan aksesibilitas dan pengalaman pengguna (*user experience*). Ekspansi ini harus tetap konsisten mengadopsi arsitektur dan prinsip desain SOLID agar kemudahan pemeliharaan lintas *platform* tetap terjaga. Lebih lanjut, penelitian selanjutnya juga perlu memfokuskan pada peningkatan validasi data di sisi sistem registrasi untuk menjaga konsistensi, seperti penerapan pembatasan domain email khusus (misalnya hanya menerima @gmail.com) serta kewajiban penggunaan *username* yang bersifat unik bagi setiap pengguna.

B. Saran

Berdasarkan hasil penelitian, terdapat beberapa saran untuk pengembangan sistem di masa mendatang:

1. Pengembangan Platform: Aplikasi *Happy Thrifting* disarankan untuk dikembangkan ke dalam bentuk aplikasi *mobile* untuk meningkatkan aksesibilitas dan pengalaman pengguna (*user experience*). Pengembangan ini harus tetap mengadopsi arsitektur dan prinsip desain SOLID agar kemudahan pemeliharaan tetap terjaga.
2. Peningkatan Validasi Data: Sistem registrasi saat ini perlu diperketat untuk menjaga konsistensi data. Disarankan untuk menerapkan aturan validasi yang lebih spesifik, seperti pembatasan domain email (misalnya, hanya menerima domain @gmail.com) serta kewajiban penggunaan *username* yang unik agar tidak ada pengguna dengan nama yang sama persis namun menggunakan email yang berbeda.

DAFTAR PUSTAKA

[1] N. Chikkanayakanahalli and R. Lead, “Prinsip Desain SOLID dalam Rekayasa Perangkat Lunak,” vol. 72, no. September, pp. 18–23, 2024.

[2] U. Usman, W. Widhiarso, and M. Rachmadi, “Implementasi SOLID dan Optimalisasi Query pada Aplikasi Pencatatan Kemiskinan,” *MDP Student Conf.*, vol. 4, no. 1, pp. 284–291, 2025, doi: 10.35957/mdp-sc.v4i1.11176.

[3] “Metode Clean Architecture Berdasarkan Prinsip Desain Solid Untuk Analisis Aplikasi Pencatatan Absensi Daring (Panda) Oleh : Josua Manalu Fakultas Ilmu Komputer Dan Rekayasa Universitas Multi Data Palembang Palembang Fakultas Ilmu Komputer Dan Rekayasa U,” 2024.

[4] J. Manalu *et al.*, “3 RD MDP STUDENT CONFERENCE (MSC) 2024 Refactor Aplikasi Absensi Berdasarkan Clean Architecture Dan Prinsip Desain SOLID,” pp. 14–21, 2024.

[5] I. Yanakiev, B. Lazar, and A. Capiluppi, “Jurnal Sistem dan Perangkat Lunak Menerapkan prinsip-prinsip SOLID untuk pemfaktoran ulang kode lama : Sebuah laporan,” vol. 220, pp. 1–15, 2025.

[6] D. Pro, “Volume : Penilaian Eksperimental tentang Pengaruh Prinsip Desain yang Solid terhadap kualitas Perangkat Lunak menggunakan Analisis Metrik CKJM,” no. September, 2022.

[7] M. Mazzari and D. Ayu Muthia, “Perancangan Sistem Informasi Penjualan Thrifting Berbasis Web,” *Akrab Juara J. Ilmu-ilmu Sos.*, vol. 7, no. 2, p. 202, 2022, doi: 10.58487/akrabjuara.v7i2.1805.

[8] M. Faqih, “Sekolah Tinggi Teknologi Terpadu Nurul Fikri Perancangan UI/UX Mobile Thrifting untuk Mengurangi Limbah Pakaian Bekas Menggunakan Metode Design Thinking Tugas Akhir,” *Https://Repository.Nurulfikri.Ac.Id/*, 2024.

[9] M. B. Sinatria, Oman Komarudin, and Kamal

Prihamdani, “Penerapan Clean Architecture Dalam Membangun Aplikasi Berbasis Mobile Dengan Framework Google Flutter,” *INFOTECH J.*, vol. 9, no. 1, pp. 132–146, 2023, doi: 10.31949/infotech.v9i1.5237.

[10] B. Nugroho, N. F. Azhar, and P. C. Subekti, “Refactoring Aplikasi Xyz Menggunakan Pendekatan Clean Architecture,” *Inf. Syst. J.*, vol. 7, no. 01, pp. 20–33, 2024, doi: 10.24076/infosjournal.2024v7i01.1579.