ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

ANALISIS SKALABILITAS APLIKASI SMART PARKING MELALUI PENERAPAN CLEAN ARCHITECTURE

Fauzan Abdillah✉, Achmad Sirojudin, M. Yusril Amin, Ery Setiawan Jullev Atmadji
Teknologi Informasi, Politeknik Negeri Jember, Indonesia
Email: fauzan.abdillah2705@gmail.com

DOI: https://doi.org/10.46880/jmika.Vol8No1.pp96-104

ABSTRACT

This research aims to enhance the capabilities of smart parking applications with a focus on scalability, organized structure, and ease of maintenance. We adopt the principles of Clean Architecture to separate functional layers and ensure clear isolation between business logic and specific implementations. The implementation of Clean Architecture in smart parking demonstrates a structured approach with high cohesion and component independence. Tools such as freezed, Dio, and Bloc pattern contribute to the development that is maintainable and extensible. This study provides practical insights into the application of Clean Architecture in real-life scenarios, emphasizing structured features' core and efficient state management with the Bloc pattern. This approach illustrates how Clean Architecture can be effectively applied to achieve scalability, good organization, and ease of maintenance in developing smart parking applications.

Keyword: Smart Parking, Clean Architecture, Scalability, Maintenance, Bloc Pattern.

ABSTRAK

Penelitian ini bertujuan untuk meningkatkan kemampuan aplikasi smart parking dengan fokus pada skalabilitas, struktur yang terorganisir, dan kemudahan pemeliharaan. Kami mengadopsi prinsip Clean Architecture untuk memisahkan lapisan fungsional dan memastikan isolasi yang jelas antara logika bisnis dan implementasi khusus. Implementasi Clean Architecture di smart parking menunjukkan pendekatan terstruktur dengan kohesi tinggi dan independensi komponen. Penggunaan alat seperti freezed, Dio, dan pola Bloc berkontribusi pada pengembangan yang dapat dipelihara dan diperluas. Penelitian ini memberikan wawasan praktis tentang penerapan Clean Architecture dalam kehidupan nyata, menekankan struktur yang teratur pada fitur inti dan manajemen state yang efisien dengan pola Bloc. Pendekatan ini menggambarkan bagaimana Clean Architecture dapat diterapkan secara efektif untuk mencapai skalabilitas, organisasi yang baik, dan kemudahan pemeliharaan dalam pengembangan aplikasi smart parking.

Kata Kunci: Smart Parking, Clean Architecture, Skalabilitas, Pemeliharaan, Pola Bloc.

PENDAHULUAN

Dampak dari pertumbuhan populasi yang cepat juga dapat diamati dalam peningkatan jumlah kendaraan yang digunakan dalam sistem transportasi (Nurhikmah, 2022). Transportasi sendiri memiliki peran yang sangat penting sebagai alat utama dalam mendukung mobilitas manusia dan barang dalam masyarakat modern (Tama & Madani, 2021; Paminto, 2020). Oleh karena itu, perkembangan dalam sektor transportasi menjadi elemen yang tidak terpisahkan dari evolusi kehidupan manusia. Tahun demi tahun, terdapat perkembangan yang signifikan dalam kemajuan sektor transportasi (Parmana & Prihatini, 2017). Data yang dikeluarkan oleh Badan Pusat Statistik Nasional menunjukkan bahwa jumlah mobil penumpang yang terdaftar mengalami peningkatan dari 15.797.746 unit pada tahun 2020 menjadi 16.413.348 unit pada tahun 2021. Fakta ini mencerminkan adanya pertumbuhan yang berkelanjutan dalam jumlah mobil penumpang, dengan peningkatan hampir sebanyak 1 juta unit setiap tahunnya. Tingginya jumlah kendaraan bermotor, terutama mobil roda empat, mengakibatkan meningkatnya permintaan akan lahan parkir yang tersedia, yang pada gilirannya menimbulkan permasalahan ketersediaan tempat parkir yang memadai. Kesulitan dalam menemukan tempat parkir yang tersedia menjadi masalah yang umum dihadapi, terutama di kota-kota besar di Indonesia. Karena itu, keberadaan sistem *smart parking* diharapkan mampu mengatasi tantangan ini (Arfianto, 2022; Andrean, 2020; Putra, 2020).

Aplikasi *smart parking* adalah sebuah platform berbasis android yang didesain dengan tujuan utama untuk menyederhanakan pengalaman pengguna dalam

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

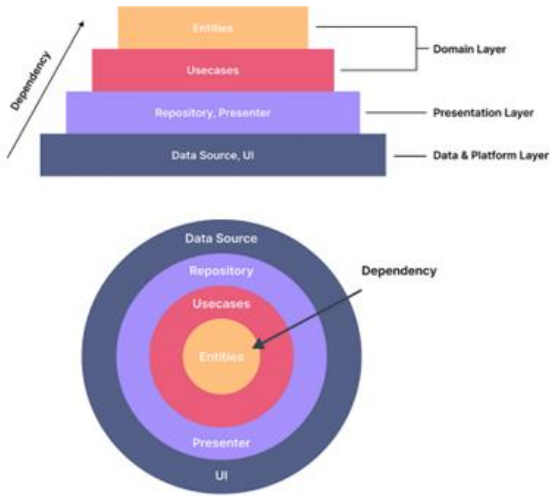
mengelola seluruh proses transaksi parkir, mulai dari pemesanan tempat parkir hingga pembayaran. Keberhasilan aplikasi ini sangat tergantung pada kemampuannya untuk menyesuaikan diri dengan berbagai lingkungan yang berbeda, serta pada pemilihan arsitektur yang tepat yang dapat mengurangi biaya pemeliharaan dan risiko kerusakan yang tidak diinginkan (Sinatria, 2023; Badrudduja & Putra, 2022). Oleh karena itu, aplikasi *smart parking* tidak hanya bertujuan untuk memberikan kenyamanan kepada pengguna dalam melakukan transaksi parkir, tetapi juga diharapkan mampu beroperasi secara efisien tanpa memerlukan investasi besar dalam hal pemeliharaan.

Penelitian ini bertujuan untuk memperkuat kemampuan aplikasi *smart parking* dalam menghadapi tantangan pengembangan fitur baru, menangani pertumbuhan sistem yang berkelanjutan, dan mempermudah proses pemeliharaan. Fokus penelitian terutama tertuju pada optimalisasi skalabilitas setelah implementasi *smart parking* dengan menerapkan konsep *Clean Architecture*. Dengan pendekatan ini, diharapkan aplikasi dapat tetap efisien sejak tahap awal dan menjaga kinerja serta keandalan optimalnya saat menghadapi situasi yang semakin kompleks serta pertumbuhan pengguna yang signifikan. *Clean Architecture* memberikan kerangka kerja yang terstruktur dan terorganisir dengan baik, memungkinkan pemisahan yang jelas antara logika bisnis aplikasi dan aspek-aspek teknis seperti antarmuka pengguna (UI), manajemen *state*, dan akses ke sumber data eksternal. Hal ini membantu pengembang untuk mengelola dan memahami kode dengan lebih baik, serta membuat perubahan dengan lebih mudah tanpa mengganggu keseluruhan sistem.

METODE PENELITIAN

Fokus analisis terhadap skalabilitas aplikasi ini terletak pada kemampuannya untuk menyesuaikan diri dengan lingkungan setelah menerapkan konsep *Clean Architecture*. Skalabilitas sendiri mengacu pada sifat suatu sistem yang dapat mengelola peningkatan jumlah pekerjaan dengan menambahkan sumber daya ke dalam sistem (Blinowski, 2022; Prawira, 2022). Analisis ini bertujuan untuk mengevaluasi seberapa efektif aplikasi dalam menghadapi perubahan, terutama dalam hal pemeliharaan, penyesuaian fitur, dan penambahan fitur baru. Selain itu, penemuan dan identifikasi potensi bug juga menjadi fokus utama, mengingat bahwa keberadaan bug dapat berdampak serius, termasuk menimbulkan biaya tambahan dan risiko terjadinya kesalahan yang tidak diinginkan. Dengan demikian, analisis ini bertujuan untuk mengukur kemampuan aplikasi dalam menanggapi perubahan dan menjamin kehandalan serta kinerja optimal dalam jangka waktu yang panjang.

Fokus utama dalam analisis ini terarah pada *Clean Architecture*, terutama dengan penekanan pada prinsip pemisahan fungsional dan skalabilitas yang menjadi dasar dari *Clean Architecture*. Pendekatan ini memberikan suatu struktur kerja untuk perancangan arsitektur aplikasi, memastikan bahwa logika bisnis dipisahkan dengan jelas dari implementasi yang lebih spesifik (Laksono, 2024). Keberhasilan dalam menerapkan pemisahan ini dapat menghasilkan pengurangan biaya dalam pemeliharaan aplikasi dan mengurangi risiko kemungkinan kerusakan yang tidak diinginkan. Gambaran dari *Clean Architecture* dapat dilihat pada Gambar 1.



Gambar 1. Diagram *Clean Architecture*

Clean Architecture, dalam analogi piramida atau irisan bawang, memberikan kerangka kerja yang kokoh untuk memisahkan kode aplikasi menjadi lapisan-lapisan yang terdefinisi dengan jelas. Dalam konteks ini, *Clean Architecture* membantu dalam pemisahan yang tajam antara kode yang berkaitan dengan logika bisnis dari kode yang terkait dengan platform, seperti antarmuka pengguna (UI), manajemen *state*, dan akses ke sumber data eksternal. Dengan memiliki struktur yang terorganisir seperti ini, pengembang dapat lebih mudah mengelola dan memahami kode, serta membuat perubahan tanpa mengganggu bagian lain dari sistem.

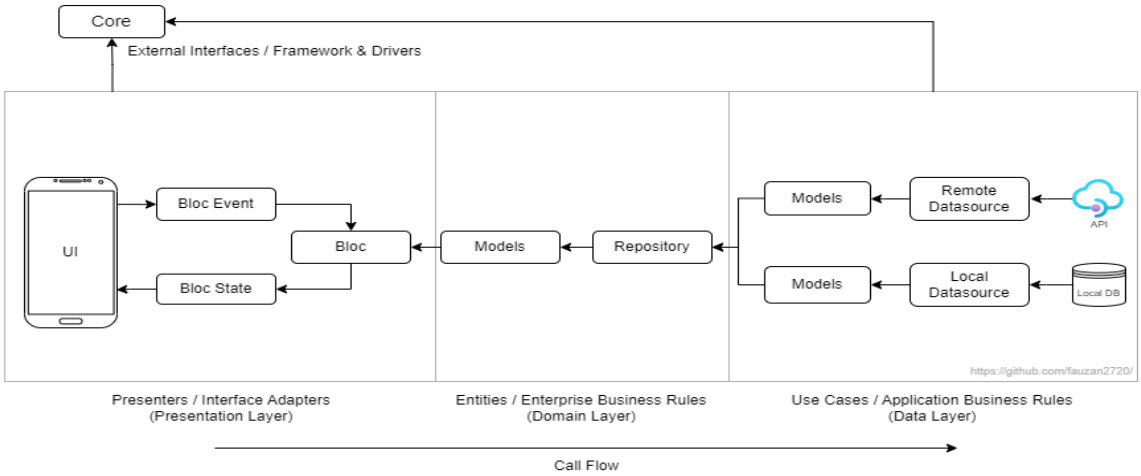
Selain itu, penerapan *Clean Architecture* juga memberikan keuntungan dalam hal pengujian independen. Dengan logika bisnis terisolasi di dalam lapisan inti, pengujian unit dapat dilakukan secara terpisah dari aspek teknis lainnya, sehingga memungkinkan untuk menguji fungsi-fungsi inti

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

dengan lebih efisien dan efektif. Analisis ini juga menyoroti peran *Clean Architecture* dalam menangani manajemen *state* aplikasi. Fokus pada pemisahan lapisan-lapisan sistem membantu dalam menciptakan manajemen *state* yang lebih terstruktur dan efisien. Dengan memisahkan logika bisnis dari tugas-tugas teknis seperti manajemen *state*, pengembang dapat mengelola *state* aplikasi dengan lebih baik, mencegah terjadinya bloatware/pembekakan, dan meningkatkan skalabilitas aplikasi.

Secara keseluruhan, analisis ini memberikan pemahaman holistik tentang sejauh mana *Clean Architecture* dapat memberikan solusi untuk berbagai tantangan yang dihadapi oleh aplikasi, baik dari segi adaptabilitas, manajemen *state*, maupun efisiensi pemeliharaan. Dengan memisahkan logika bisnis dari detail implementasi teknis, *Clean Architecture* menciptakan landasan yang kokoh untuk pengembangan dan pemeliharaan aplikasi yang *scalable* dan mudah dimengerti.

HASIL DAN PEMBAHASAN



Gambar 2. Arsitektur Sistem

Gambar 2 menunjukkan implementasi *Clean Architecture* yang diusung oleh Uncle Bob, struktur folder yang diadopsi pada aplikasi *smart parking* ini memperlihatkan pendekatan yang sistematis dan terstruktur. Direktori *"lib"* sebagai *root* dari proyek terbagi menjadi dua bagian utama, yaitu *"core"* dan *"features"*. Pada bagian *"core"*, elemen-elemen inti seperti komponen-komponen, konstanta-konstanta, basis data, ekstensi, serta utilitas umum tersusun secara rapi. Di sisi lain, direktori *"features"* menjadi wadah bagi fungsionalitas utama aplikasi, seperti fitur *parking* yang dijelaskan di sini.

Struktur *"parking"* di dalam *"features"* pada Gambar 3 dibawah, memiliki lapisan-lapisan yang mewakili konsep *Clean Architecture* secara jelas. Di dalam *"core"*, terdapat *"bloc_providers"* dan *"dependency"* yang menunjukkan penyedia BLOC dan *dependency injection* dengan baik. Bagian "data" berisikan segala yang terkait dengan pemrosesan data, seperti sumber data, model, dan *repository*. Lapisan *"domain"* menjelaskan entitas-entitas yang mendefinisikan inti bisnis aplikasi, dengan sub-direktori *"entities"* yang merinci struktur tubuh dan responnya.

```
├─ lib
   ├─ core
   └─ features
      ├─ parking
      │  ├─ core
      │  │  ├─ bloc_providers
      │  │  └─ dependency
      │  ├─ data
      │  │  ├─ datasources
      │  │  ├─ models
      │  │  └─ repositories
      │  ├─ domain
      │  │  ├─ models
      │  │  └─ repositories
      │  └─ presentation
      │     ├─ managers
      │     ├─ pages
      │     └─ widgets
      └─ other_feature…
```

Gambar 3. Struktur Folder Proyek

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

Tingkat presentasi didefinisikan di dalam direktori *"presentation"* dengan manajer-manajer yang mengkoordinasikan logika aplikasi (*state management*), halaman-halaman yang mewakili antarmuka pengguna, dan widget-widget untuk tampilan yang lebih spesifik. Dengan struktur ini, aplikasi *smart parking* menunjukkan kemandirian komponen dan tingkat kohesi yang tinggi, memberikan fondasi yang kokoh untuk pengujian skalabilitas.

Implementasi *Clean Architecture*

Entity / Enterprise Business Rules (Domain Layer)

Entity merupakan struktur terdalam dari *Clean Architecture*, fokus ini terletak pada definisi UI (*User Interface*) yang disajikan kepada pengguna dan aturan bisnis yang mendasari aplikasi *smart parking*. Aturan bisnis pada layer ini memandu pemahaman tentang data apa yang diperlukan untuk membangun tampilan UI yang optimal dan input apa yang dibutuhkan agar aplikasi *smart parking* berjalan sesuai kebutuhan pengguna. Dalam struktur folder proyek *smart parking*, terdapat dua komponen utama, yaitu models dan repositories. Struktur ini dirinci sebagai berikut:

```
├─ domain
│  ├─ models
│  │  ├─ explore_parking_model.dart
│  │  └─ parking_area.dart
│  └─ repositories
│     └─ parking_repository.dart
```

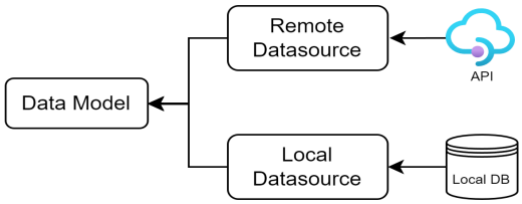
Gambar 4. Struktur Folder *Domain Layer*

Pada direktori *"domain"* seperti **Gambar 4** diatas, terdapat sub-direktori *"models"* dan *"repositories"*. Di dalam *"models"* terdapat tiga file yang memegang peran penting dalam mendefinisikan struktur data untuk tampilan antarmuka pengguna (UI): *"explore_parking_model.dart"* dan *"parking_area.dart"*. Setiap file ini fokus pada aspek tertentu, seperti informasi nama area parkir, dan alamat.

Disisi lain, pada direktori *"repositories"*, terdapat file *"parking_repository.dart"*. File ini memuat implementasi dari *Abstract Repository*, dimana logika bisnis terkait area parkir dipisahkan dari sumber data konkret. Metode ini dapat diimplementasikan oleh *repository* konkrit pada layer data untuk mengambil data terkini tentang area parkir. Sebagai hasilnya, pendekatan ini memfasilitasi fleksibilitas dan kemudahan pengujian.

Use Case / Application Business Rules (Data Layer)

Proses perolehan data dimulai dengan penyedia data, yang dapat berupa API eksternal atau database lokal. Penyedia data ini menyediakan akses ke informasi terkini mengenai area parkir. Selanjutnya, *datasource* bertindak sebagai perantara antara aplikasi dan penyedia data. Ketika aplikasi membutuhkan data, *datasource* akan meminta dan mengelola res dari penyedia data, kemudian mengonversi respon tersebut ke dalam bentuk model data yang sesuai, seperti pada Gambar 5.



Gambar 5. *Application Business Rules*

Proses perolehan data tersebut terintegrasi dengan tiga komponen utama dalam lapisan data, yakni *datasources*, models, dan *repositories*. Struktur ini dirinci sebagai berikut:

```
├─ data
│  ├─ datasources
│  │  └─ parking_remote_datasource.dart
│  ├─ models
│  │  └─ explore_response_model.dart
│  └─ repositories
│     └─ parking_repository_impl.dart
```

Gambar 6. Struktur Folder *Data Layer*

Pada *"datasources"* seperti Gambar 6, terdapat file *"parking_remote_datasource.dart"* yang berfungsi sebagai kelas penyedia data dari sumber eksternal. Komunikasi data menjadi kunci dalam mengelola akses ke sumber data, baik secara *remote* maupun lokal. Dalam konteks ini, datasource berfungsi sebagai gerbang masuk yang mengatur interaksi dengan data. Anotasi *@RestApi* menunjukkan keterlibatan dalam komunikasi dengan API menggunakan library Dio, dengan ketergantungan pada Build Runner untuk pengaturan dan menghasilkan kode. Dengan adanya *remote datasource* ini, aplikasi dapat berinteraksi dengan *backend server* dan mendapatkan informasi terkini mengenai area parkir.

Dalam *"models"*, terdapat file *"explore_response_model.dart"* yang mendefinisikan model respons dari API dan berperan dalam mengubah

data dari format API ke dalam model domain yang bersih dan sesuai dengan kebutuhan aplikasi *smart parking*. Proses pengelompokkan ini terdapat dalam metode *toDomain()*, yang berfungsi untuk mentransformasikan objek dari kelas respons API ke dalam bentuk model domain yang lebih bersih dan sesuai dengan logika bisnis aplikasi *smart parking*.

Pada *"repositories"*, terdapat file *"parking_repository_impl.dart"* yang mengimplementasikan logika bisnis terkait area parkir dan mengemas data dengan format yang optimal untuk UI. *Repository* ini juga menangani potensi kesalahan saat mengambil data, sehingga UI dapat merespon dengan tepat. *Repository* mengemas data dengan format yang optimal, menyembunyikan kerumitan dan detail implementasi dari lapisan data. Jika terjadi kesalahan saat mengambil data, *repository* akan mengembalikan objek Left yang berisi informasi kesalahan, sehingga UI dapat merespons dengan tepat. Sebaliknya, jika pengambilan data berhasil, *repository* akan mengembalikan objek *Right* yang berisi data yang dibutuhkan oleh UI.

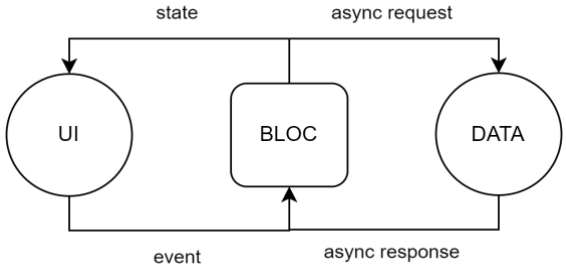
Presenters / Interfaces Adapters (Presentation Layer)

Dalam struktur proyek *smart parking* pada lapisan Presenters atau *Interfaces Adapters (Presentation Layer)*, terdapat tiga sub-direktori utama: *"managers"*, *"pages"*, dan *"widgets"*. Lapisan ini bertanggung jawab untuk mengelola tampilan UI dan berkomunikasi dengan lapisan logika bisnis melalui Bloc *pattern* (Zulistiyan & Wibowo, 2024; Azis, 2023; Szczepanik & Kedziora, 2020). Setiap sub-direktori ini memiliki peran yang spesifik dalam mengelola presentasi dan interaksi pengguna. Struktur folder dapat dijelaskan sebagai berikut:

```
├─ presentation
│  ├─ managers
│  │  ├─ explore_parking_Bloc.dart
│  │  ├─ explore_parking_event .dart
│  │  └─ explore_parking_state.dart
│  ├─ pages
│  │  ├─ explore_parking_page.dart
│  │  └─ parking_area.dart
│  └─ widgets
│     ├─ explore_parking_card.dart
│     └─ parking_area_card.dart
```

Gambar 7. Struktur Folder *Presentation Layer*

Dalam direktori *"managers"* pada Gambar 7, terdapat file *"explore_parking_*Bloc.*dart"* yang mengimplementasikan Bloc pattern untuk mengelola *state* terkait data area parkir. File *"explore_parking_event .dart"* dan *"explore_parking_state.dart"* digunakan untuk mendefinisikan *event* dan *state* yang terkait dengan manajemen *state* di dalam *ExploreParking*Bloc. Arsitektur dari Bloc *Pattern* dapat dilihat pada Gambar 8.



Gambar 8. BLOC *Pattern*

Fokus Bloc adalah menampilkan *state* ke halaman UI agar pengguna bisa berinteraksi dengan data area parkir. Saat menginisialisasi, Bloc dipanggil untuk memulai data area parkir dengan memicu *event ExploreParkingEvent .getData()*. Dalam proses memantau perubahan *state*, digunakan kelas BlocConsumer yang dapat memanfaatkan listener dan builder saat terjadi perubahan *state*. *Listener* bertanggung jawab merespons perubahan *state*, sedangkan builder merancang UI sesuai dengan *state* yang diterima.

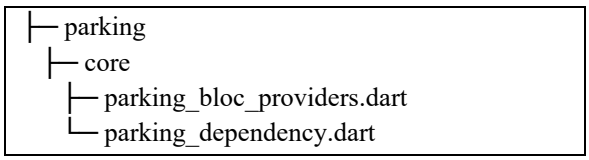
Selanjutnya, di dalam direktori *"pages"* terdapat file *"explore_parking_page.dart"* yang merupakan halaman utama untuk menampilkan data area parkir. Halaman ini menggunakan *BlocConsumer* untuk mendengarkan perubahan *state* dari *ExploreParkingBloc*. Selain itu, terdapat file *"parking_area.dart"* yang mungkin digunakan sebagai widget khusus untuk menampilkan detail area parkir. Dalam direktori *"widgets"*, terdapat file *"explore_parking_card.dart"* dan *"parking_area_card.dart"* yang mungkin digunakan sebagai widget reusable untuk menampilkan data area parkir dalam bentuk kartu di berbagai bagian aplikasi.

Dengan struktur proyek yang terorganisir seperti ini, lapisan presentasi dapat mengelola tampilan UI dan berinteraksi dengan lapisan logika bisnis dengan efisien, tanpa terlalu banyak kompleksitas. Widget-widget yang telah dibuat dapat digunakan kembali, memudahkan pengembangan selanjutnya.

External Interfaces (Core Feature)

Core feature yang terdapat pada *layer feature* adalah bagian penting dari manajemen aplikasi *Smart*

parking. Dua kelas utama yang mendefinisikan fungsi dan dependensi pada lapisan fitur ini adalah *ParkingBlocProviders* dan *ParkingDependency*. Dengan pemisahan logika bisnis dan dependensi khusus dari komponen-komponen lain dalam proyek, *core feature* ini mendukung kemudahan dalam pengelolaan dan pengembangan fitur-fitur spesifik dalam aplikasi. Struktur folder *core* dalam *feature*, dapat dilihat pada Gambar 9.

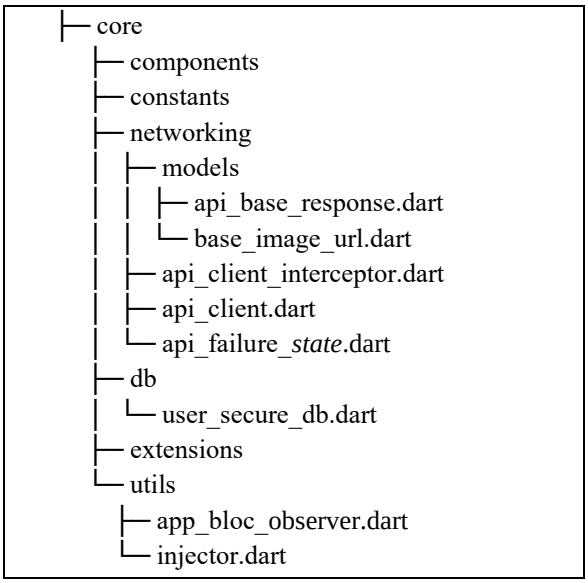


Gambar 9. Struktur *Folder Core Feature*

ParkingBlocProviders bertanggung jawab untuk mendaftarkan *Business Logic* Components (BLOC) ke dalam *main app*, mengelompokkannya sesuai dengan lapisan fitur. Kelas ini berperan sebagai penyedia Bloc yang dapat diakses oleh seluruh komponen UI di aplikasi. Disisi lain, *ParkingDependency* berfungsi sebagai pengatur dependensi dengan mendaftarkan *instance class* ke dalam main *app*, dan mengelompokkannya pada lapisan fitur. Implementasi *dependency injection* menggunakan library get_it untuk menyederhanakan manajemen dependensi dalam proyek *smart parking*.

External Interfaces (Core Project)

Core project yang bersifat global berfungsi sebagai penyimpan fungsi-fungsi atau komponen-komponen umum yang dapat digunakan secara luas di seluruh proyek *smart parking*. Tujuannya adalah menyediakan satu titik akses untuk elemen-elemen inti seperti fungsi bantu, koneksi jaringan, dan manajemen *state*. Dengan pendekatan ini, proyek dapat menghindari duplikasi kode, mempromosikan *reusability*, dan memastikan konsistensi serta efisiensi pengembangan di seluruh bagian aplikasi. Hasil dari implementasi ini dapat dilihat pada **Gambar 10** sebagai berikut:



Gambar 10. Struktur *Folder Core Project*

Hasil Pengujian Skalabilitas

Dalam menguji skalabilitas proyek *smart parking*, kami melibatkan dua validator berpengalaman. Validator pertama adalah *former Tech Lead* Gojek, mentor, pendiri platform belajar Jago Flutter serta komunitas FIC *(Flutter Intensive Club)*. Sementara itu, validator kedua adalah *senior mobile developer* di PT Koltiva, juga *former* di Telkomsel dan Detik.com. Mereka membawa pandangan berharga dari aspek teknis hingga praktis di dunia industri. Dengan keterlibatan keduanya, diharapkan proyek *smart parking* mendapatkan evaluasi menyeluruh mengenai skalabilitasnya. Keterlibatan kedua pihak akan memberikan penilaian menggunakan skala 1 hingga 5 untuk memberikan evaluasi yang lebih terperinci terkait dengan skalabilitas proyek *Smart Parking*. Dengan partisipasi dari keduanya, diharapkan proyek *Smart Parking* dapat menerima evaluasi yang komprehensif, yang akan membantu menentukan tingkat kesiapan skalabilitasnya secara keseluruhan. Dan hasil dari pengujian skalabilitas ini dapat dilihat pada **Tabel 1** dibawah.

Tabel 1. Hasil Pengujian Skalabilitas

	Pertanyaan	**Ahli**	
		1	**2**
Implementasi *Enterprise Business Rules (Entity)*			
1	Sejauh mana implementasi *entity* mempermudah pemetaan data antara UI dan *Rest API*?	5	5
2	Seberapa efektif penggunaan *freezed* pada *entity* model?	5	5

3	Apakah implementasi *abstract repository* membantu dalam pengembangan dan pemeliharaan *service Rest API*?	5	5
4	Apakah implementasi domain *layer* memenuhi kriteria fitur yang mudah diuji?	4	5
5	Sejauh mana abstraksi *entity* membantu dalam memisahkan logika bisnis dari logika presentasi dan persistensi data?	5	5
Implementasi *Application Business Rules (Use Case)*			
1	Seberapa efektif penggunaan *Retrofit* dalam *use case*?	5	4
2	Seberapa efektif pengelolaan *datasource* terhadap perubahan *Rest API*?	5	5
3	Seberapa efektif penggunaan *freezed* membantu dalam pengembangan fitur?	4	4
4	Apakah implementasi *repository* memenuhi kebutuhan pemeliharaan fitur?	4	5
5	Apakah pemisahan token dan *base* URL memudahkan manajemen developer?	5	5
6	Seberapa efektif penanganan *error response* pada *repository*?	5	5
7	Apakah implementasi data *layer* memenuhi kriteria fitur yang mudah diuji?	5	5
8	Seberapa baik pengelolaan *datasource* dapat menangani perkembangan *Rest* API pada *endpoint* atau struktur data?	5	5
Implementasi *Interface Adapter (Presenters)*			
1	Apakah implementasi Bloc memenuhi arsitektur yang *scalable*?	5	5
2	Apakah implementasi *freezed* pada Bloc membantu dalam pemeliharaan fitur?	5	5
3	Seberapa mudah adaptasi teknologi *state management* yang digunakan terhadap teknologi *state management* baru?	5	4
4	Apakah konsumsi Bloc pada *pages* sudah benar dan mudah dipelihara?	5	5
5	Seberapa membantu penulisan kode program yang disederhanakan ke dalam *layer widgets* untuk pemeliharaan fitur?	5	5
6	Apakah pemisahan *widget* memenuhi kriteria fitur yang mudah digunakan kembali (*reusable widget*) ?	5	5
7	Apakah implementasi Bloc memenuhi kriteria fitur yang mudah diuji?	5	5
Implementasi *External Interfaces*			
1	Apakah pemetaan Bloc *Providers* dalam *layer feature* memenuhi kriteria aplikasi yang mudah dipelihara?	5	5
2	Apakah *dependency injection* di *layer feature* mendorong struktur yang mudah dipelihara?	5	5
3	Sejauh mana pemisahan komponen *widget* mendukung pengembangan fitur baru dan penyesuaian antarmuka pengguna?	5	5
4	Apakah penggunaan *constants layer* memudahkan developer untuk mambangun fitur?	5	5

5	Apakah penggunaan *extensions layer* memudahkan developer untuk mambangun fitur?	5	5
6	Apakah penggunaan *networking layer* memudahkan developer untuk mambangun fitur?	5	5
7	Apakah penggunaan *utils layer* memudahkan developer untuk mambangun fitur?	5	5

Berdasarkan hasil pengujian skalabilitas *Clean Architecture* proyek *smart parking*, dapat disimpulkan bahwa proyek tersebut telah berhasil mencapai tingkat kesiapan yang tinggi dalam hal skalabilitas. Dengan penilaian sebagian besar mencapai skor 5, baik dari aspek teknis maupun praktis, proyek ini dinilai sangat efektif dalam implementasi *Enterprise Business Rules, Application Business Rules, Interface Adapter,* dan *External Interfaces*. Ditemukan bahwa penggunaan konsep *Clean Architecture* mampu memisahkan logika bisnis dengan jelas, mendukung pemeliharaan fitur, dan memudahkan pengembangan baru. Penerapan teknologi seperti freezed, Bloc, dan *dependency injection* juga memberikan kontribusi positif dalam memudahkan pengembangan, pemeliharaan, dan penyesuaian fitur. Dengan demikian, proyek *smart parking* dapat diandalkan dalam pengembangan lebih lanjut.

KESIMPULAN

Dalam pengembangan aplikasi *smart parking*, penerapan *Clean Architecture* dan prinsip-prinsip desain perangkat lunak modern, seperti pola Bloc, freezed, dan Dio, telah membawa dampak positif. Struktur proyek yang terorganisir dengan baik dan menciptakan lingkungan pengembangan yang efisien. Adopsi *Clean Architecture* membantu memisahkan lapisan fungsional, meningkatkan kohesi, dan membuat komponen-komponen menjadi mandiri, memberikan dasar yang kuat untuk pemeliharaan dan pengembangan fitur baru. Evaluasi skalabilitas oleh validator berpengalaman memberikan gambaran menyeluruh tentang kesiapan proyek dalam menghadapi pertumbuhan. Meskipun proyek telah mencapai tujuan dalam meningkatkan kemampuan aplikasi, beberapa saran perbaikan termasuk optimalisasi *dependency injection*, peningkatan penanganan *error*, uji coba yang lebih lanjut, penyediaan dokumentasi yang komprehensif, dan eksplorasi teknologi terkini dapat memperkuat posisi proyek dalam mengatasi tantangan perkembangan teknologi dan memenuhi kebutuhan pengguna.

DAFTAR PUSTAKA

Andrean, K., Armanto, H., & Pickerling, P. (2020). Sistem Tempat Parkir Terintegrasi yang Dilengkapi dengan Aplikasi Mobile dan Mikrokontroller. *Journal of Information System, Graphics, Hospitality and Technology, 2*(01), 22-29. https://doi.org/10.37823/insight.v2i01.79

Arfianto, M. R. (2022). Analisis Desain User Interface pada Aplikasi Pencari Parkir Mobil. *Desainpedia Journal of Urban Design, Lifestyle & Behaviour, 1*(1). https://doi.org/10.36262/dpj.v1i1.589

Azis, M. H. N., Pinandito, A., & Maghfiroh, I. S. E. (2023). Analisis Perbandingan Penggunaan State Management pada Aplikasi Ditonton menggunakan Framework Flutter. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 7*(1), 148-153.

Badrudduja, M. H., & Putra, R. E. (2022). Penerapan Clean Architecture pada Aplikasi Pemesanan Makanan menggunakan Metode Slope One Algorithm. *Journal of Informatics and Computer Science (JINACS), 3*(04), 506-514. https://doi.org/10.26740/jinacs.v3n04.p506-514

Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access, 10*, 20357-20374. https://doi.org/10.1109/ACCESS.2022.3152803

Laksono, W. P., Wicaksana, B. S. T., Wijasena, A. Y., & Sahria, Y. (2024). Implementasi Clean Architecture Dalam Membangun Aplikasi Mobile Menggunakan Flutter. *Nusantara Journal of Multidisciplinary Science, 1*(6), 173-180.

Nurhikmah, T., Fauzi, A., Putri, S. C. T., Asmarani, D., Damayanti, V., & Thalitha, R. F. (2022). Analisis faktor-faktor yang mempengaruhi loyalitas pelanggan layanan transportasi online (go-jek): kualitas pelayanan, harga dan kepuasan konsumen. *Jurnal Ilmu Manajemen Terapan, 3*(6), 646-656. https://doi.org/10.31933/jimt.v3i6.946

Paminto, A. K. (2020). Analisis dan Proyeksi Kebutuhan Energi Sektor Transportasi di Indonesia. *Jurnal Energi dan Lingkungan (Enerlink), 16*(2), 51-54. https://doi.org/10.29122/jel.v16i2.4801

Parmana, A. E., & Prihatini, A. E. (2017). Pengaruh Citra Merek Dan Kualitas Pelayanan Terhadap Keputusan Pengambilan Jasa Transportasi

ISSN: 2598-8565 (media cetak)
ISSN: 2620-4339 (media online)

(Studi Kasus Pada Po. Bejeu Jurusan Semarang–Jakarta). *Jurnal Ilmu Administrasi Bisnis*, *6*(3), 572-579. https://doi.org/10.14710/jiab.2017.16790

Prawira, A. F., Putra, W. H. N., & Purnomo, W. (2022). Pengembangan Aplikasi E-Commerce Anggrek berbasis Android menggunakan Clean Architecture (Studi Kasus: PT. Java Indo Arjuna). *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *6*(8), 3606-3612.

Putra, A. S. (2020). Analisa Dan Perancangan Sistem Tata Kelola Parkir Cerdas Di Kota Pintar Jakarta. *IKRA-ITH Informatika: Jurnal Komputer dan Informatika*, *4*(3), 13-21.

Sinatria, M. B., Komarudin, O., & Prihandani, K. (2023). Penerapan Clean Architecture Dalam Membangun Aplikasi Berbasis Mobile Dengan Framework Google Flutter. *Infotech Journal*, *9*(1), 132-146. https://doi.org/10.31949/infotech.v9i1.5237

Szczepanik, M., & Kedziora, M. (2020). State Management and Software Architecture Approaches in Cross-platform Flutter Applications. *In ENASE* (pp. 407-414). https://doi.org/10.5220/0009411604070414

Tama, Y. P., Putri, A. A., & Madani, M. W. (2021). Integrasi Sistem Transportasi Berkelanjutan Pada Kawasan Wisata Ubud-Bali. *Jurnal Transportasi Multimoda, 19(*1), 10-19. https://doi.org/10.25104/mtm.v19i1.1853

Zulistiyan, M., Adrian, M., & Wibowo, Y. F. A. (2024). Performance Analysis of Bloc and GetX State Management Library on Flutter. *Journal of Information System Research (JOSH), 5*(2), 583-591. https://doi.org/10.47065/josh.v5i2.4698