Journal of Internet and Software Engineering Vol: 1, No 3, 2024, Page: 1-15

Implementasi *Domain Driven Design* dan *Clean Architecture* dalam Pengembangan *Web Service* Aplikasi Alifarm Digital

Rama Sakti Hafidz Fadhilah Aziz*, Irwan A. Kautsar, Sumarno
Informatika, Universitas Muhammadiyah Sidoarjo

Abstrak: Implementasi *Domain Driven Design* (DDD) dan *Clean Architecture* dalam pengembangan aplikasi *web service* Alifarm Digital telah menunjukkan manfaat yang signifikan. Proses pengembangan yang melibatkan analisis kebutuhan, desain berbasis DDD, dan implementasi *Clean Architecture* memastikan efisiensi dalam pengembangan aplikasi. DDD membantu dalam pemodelan subdomain seperti *invest the project*, detail proyek, dan pembagian hasil proyek, sementara *Clean Architecture* memudahkan pemisahan presentasi, logika bisnis, dan sumber data untuk pemeliharaan dan pengembangan yang lebih mudah. Meskipun memberikan keuntungan, tantangan juga diidentifikasi, termasuk lonjakan waktu respons tinggi dan fluktuasi dalam permintaan yang gagal, menunjukkan kesulitan sistem dalam menangani beban tinggi. Pemantauan terus-menerus dan pemeliharaan sistem sangat penting untuk memastikan kualitas layanan yang optimal. Penelitian ini memberikan wawasan berharga tentang pentingnya menerapkan DDD dan *Clean Architecture* untuk aplikasi *web service* yang efisien, dapat diskalakan, dan berkualitas tinggi.

Kata kunci: *Domain Driven Design, Clean Architecture, Web Service, Crowdfunding, Locust*

DOI: https://doi.org/10.47134/pjise.v1i3.2511
*Correspondence: Rama Sakti Hafidz Fadhilah Aziz
Email: ramasakti1337@gmail.com

Received: 21-04-2024
Accepted: 01-05-2024
Published: 31-06-2024

Abstract: *The implementation of Domain Driven Design (DDD) and Clean Architecture in developing the Alifarm Digital web service application has shown significant benefits. The development process involving requirements analysis, DDD-based design, and Clean Architecture implementation ensures efficiency in application development. DDD aids in modeling subdomains such as invest the project, project details, and project profit distribution, while Clean Architecture facilitates the separation of presentation, business logic, and data sources for easier maintenance and development. Despite the advantages, challenges were identified, including high response time spikes and fluctuations in failed requests, indicating the system's struggle with high loads. Continuous monitoring and system maintenance are crucial to ensure optimal service quality. This research provides valuable insights into the importance of applying DDD and Clean Architecture for efficient, scalable, and high-quality web service applications.*

Keywords: *Domain Driven Design, Clean Architecture, Web Service, Crowdfunding, Locust*

Pendahuluan

Dalam era digital yang terus berkembang, aplikasi *web service* telah menjadi bagian integral dari hampir setiap aspek kehidupan kita (Deljouyi, 2022; Koblitz, 2023; Li, 2022; Ling, 2023; Miao, 2023; Ngo, 2022; Stefanovska, 2022). Salah satu fungsi *web service* pada sistem adalah untuk berkomunikasi dengan database secara real time dan menghasilkan data yang dapat diakses dengan mudah oleh berbagai platform (Ahmad, 2022; Alulema, 2023; Blond, 2023; Chamari, 2023; Chessa, 2023). *Web service* juga digunakan untuk mendapatkan, menyimpan, memperbarui, dan menghapus data dengan metode yang telah ditentukan (Prasetyo et al., 2022).

Penerapan *web service* telah menjadi tren saat ini, salah satu contoh penerapan teknologi ini adalah AliFarm Digital. Aplikasi AliFarm Digital merupakan sebuah web aplikasi yang memberikan layanan dalam bidang investasi peternakan digital. Ide pengembangan web aplikasi ini muncul karena maraknya kasus penipuan berkedok investasi, seperti kasus Indra Kenz, yang telah merugikan investor dengan nilai mencapai Rp3,8 miliar (Mufidah & Setiawan, 2022). Selain itu, Aplikasi AliFarm Digital dikembangkan berbasis syariah dengan model *crowdfunding*, yang diharapkan dapat memberikan alternatif investasi yang lebih aman dan sesuai dengan prinsip-prinsip syariah (*Crowdfunding Pada Teknologi Keuangan Islam | SOSMANIORA: Jurnal Ilmu Sosial Dan Humaniora*, n.d.). Namun, dalam pengembangan *web service*, masalah mulai muncul ketika fungsionalitas pada aplikasi semakin banyak, dan *traffic* yang terus meningkat. Sehingga saat suatu fungsional mengalami eror maka akan berdampak pada keseluruhan *web services* (Budi & Bachtiar, n.d.). Tidak terkecuali dalam pengembangan *web service* aplikasi alifarm digital.

Dalam rangka untuk menjaga dan meningkatkan kualitas dan kecepatan pengembangan *web service* aplikasi Ali Farm Digital, perlu adanya pendekatan yang kuat dalam pengembangannya. *Domain Driven Design* (DDD) dan *Clean Architecture* adalah dua metode yang telah terbukti efektif dalam membangun aplikasi yang skalabel, mudah dikelola, dan mudah dipelihara (Ramadhan & Zukhri, 2023).

Domain Driven Design (DDD) adalah suatu metode pengembangan perangkat lunak yang memfokuskan implementasi sistem pada penyelesaian masalah-masalah yang dihadapi pada domain bisnis dalam bentuk domain model (Prayoga et al., 2021). Pendekatan *domain driven design* digunakan agar desain *web service* sistem dapat disesuaikan dengan ranah proses bisnis yang akan diselesaikan sehingga sistem menjadi lebih mudah untuk dimengerti (Ramadhan & Zukhri, 2023). Metode ini menekankan pada pemodelan domain yang kuat, sehingga aplikasi dapat lebih baik merepresentasikan dan mengatasi permasalahan bisnis yang ada.

Sedangkan, *Clean Architecture* merupakan sebuah arsitektur yang ditujukan agar kode pada suatu proyek menjadi lebih terstruktur dan rapi, dengan membagi konsentrasi kode menjadi beberapa layer (Fajri, 2022). *Clean Architecture* membantu memastikan bahwa perubahan dalam satu lapisan tidak mempengaruhi lapisan lainnya, sehingga aplikasi lebih mudah diuji, dipelihara, dan diperbarui (Giner-Miguelez, 2022; Mandal, 2022; Yang, 2022). Menerapkan konsep *Clean Architecture* sangat berpengaruh dalam mengembangkan aplikasi, untuk berkolaborasi dalam sebuah *project* yang akan dikerjakan tim, setiap

programmer harus mengikuti konsep *Clean Architecture*, sehingga dengan kode yang baik dan benar, antar *programmer* tidak mengalami kesulitan ketika melanjutkan kode dari *programmer* lain. Selain itu menerapkan *Clean Architecture* dapat menghemat waktu dan biaya saat harus dilakukan *maintenance* terhadap sistem (Anhar & Anggraeny, 2022).

Metode

A. Prosedur Pengembangan

Prosedur pengembangan *web service* AliFarm Digital melibatkan beberapa langkah penting, yang mencakup analisis kebutuhan, perancangan berdasarkan *Domain Driven Design* (DDD) dan implementasi dengan *Clean Architecture*. Berikut adalah langkah-langkahnya:

Gambar 1. Prosedur Pengembangan

Dengan mengikuti prosedur ini, pengembangan *web service* AliFarm Digital dapat dilakukan dengan lebih efisien. Selain itu, langkah ini akan memastikan bahwa sistem yang dibuat sesuai dengan kebutuhan dan persyaratan yang telah ditetapkan.

B. Model Pengembangan

1. *Domain Driven Design*

Metode *Domain Driven Design* (DDD) digunakan dalam perancangan *Web Service* AliFarm Digital dengan pendekatan berikut:

a. Pemodelan Subdomain: Dalam konteks Web Service AliFarm Digital, subdomain seperti *invest the project*, detail proyek, dan pembagian hasil proyek diidentifikasi dan dimodelkan.
b. Penggambaran *Class Diagram*: Domain model yang telah dibuat digunakan sebagai dasar untuk membuat *class diagram* dari domain model, *repository*, *events*, dan *query object*. Hal ini membantu dalam memvisualisasikan struktur dan hubungan antar komponen sistem.
c. Perancangan Web Service AliFarm Digital dengan DDD dapat difokuskan pada pembagian aplikasi menjadi domain yang lebih kecil, sehingga kompleksitas aplikasi dapat dikurangi dan fungsi bisnis dari aplikasi dapat diklasifikasikan dengan lebih baik.

2. *Clean Architecture*

Aplikasi yang dikembangkan dengan menggunakan *Clean Architecture* dapat menjadi kokoh dari segi bahasa dan ramah terhadap perubahan konfigurasi karena adanya pemisahan antara tampilan (View), logika bisnis (App), dan sumber data (DataStore) (Saifulloh et al., 2023). Dengan adanya pemisahan ini, perubahan pada salah

satu bagian tidak akan secara langsung memengaruhi bagian lainnya, sehingga aplikasi dapat lebih mudah untuk dipelihara dan dikembangkan.

C. Implementasi

Dalam implementasi *Domain Driven Design* (DDD) pada pengembangan *Web Service* AliFarm Digital, terdapat proses analisis kebutuhan sehingga muncullah kebutuhan fungsional dan non-fungsional seperti pada tabel di bawah ini:

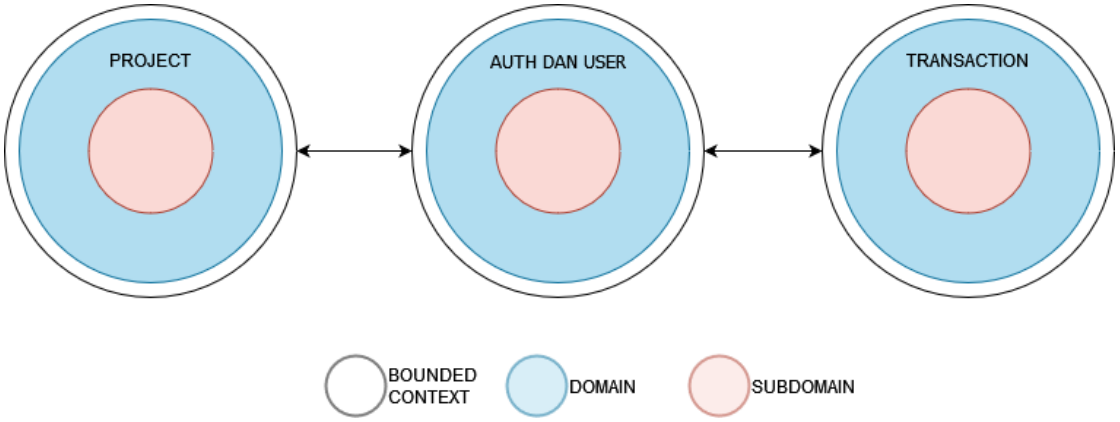
Tabel 1. Fungsional dan Non Fungsional Fitur

Fitur	
Fungsional Fitur	Non Fungsional Fitur
User Authentication and Authorization	Menampilkan data *social investor* dan peternak
Create Project	Menampilkan data *social view project* (diambil dari setiap project diklik atau dilihat)
Show Detail Project	Menampilkan grafik social transaksi / investasi per 7 hari
Accept / Pending Project	Menampilkan grafik nilai transaksi / investasi per 7 hari
Reject Project	Menampilkan data klik pada detail project
Invest to Project	Tombol *share project* untuk membagikan detail project ke WhatsApp

Setelah dilakukan analisa kebutuhan barulah dapat mengimplementasikan *Domain Driven Design* dan *Clean Architecture*. Langkah awal dalam mengimplementasikan DDD yakni mendefinisikan *obtiquitous language* seperti di bawah ini:

1. Investor adalah masyarakat umum yang ingin berinvestasi pada platform AliFarm Digital
2. Admin adalah pengguna yang ditugaskan oleh AliFarm Digital untuk mengelola platform AliFarm Digital
3. Peternak adalah pengguna yang membutuhkan pendanaan dalam menjalankan projectnya
4. Project adalah rencana peternak dalam menjalankan proses peternakan. Dalam project terdapat satu atau lebih kandang, jumlah hewan, rencana pakan, nilai project, dan tenor project
5. Setiap project memiliki kandang dan setiap kandang memiliki hewan
6. *Invest to project* merupakan kegiatan dari investor untuk memberikan pendanaan kepada peternak

Setelah *obtiquitous language* telah terdefinisi maka langkah selanjutnya yakni mendefinisikan *bounded context* dan *context* map seperti gambar di bawah ini:



Gambar 2. Context Map

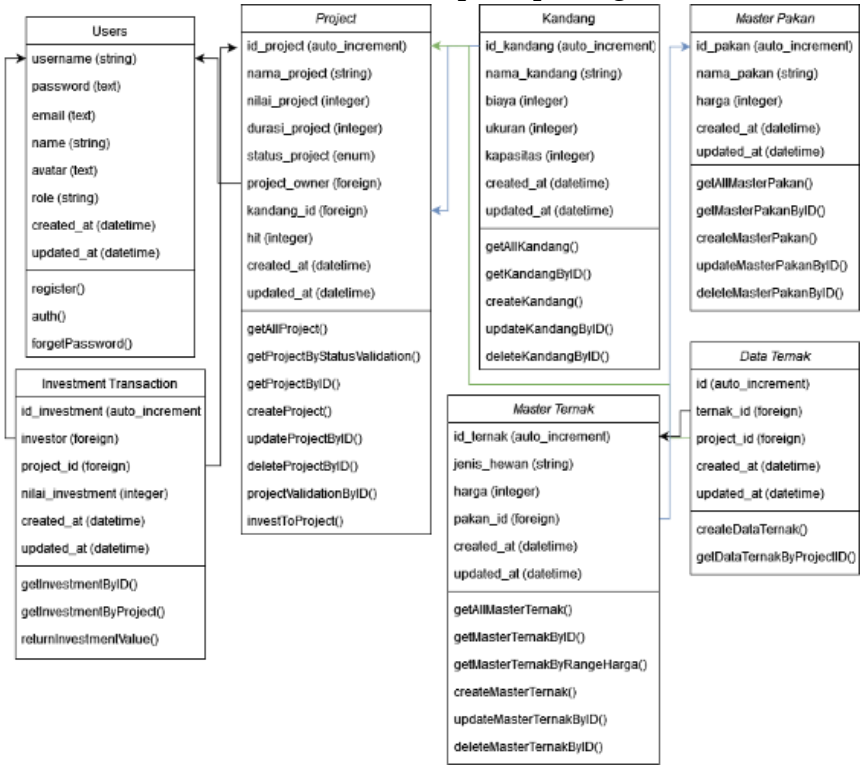
Satu subdomain memiliki satu konteks terbatas, dan ada relasi *supplier-customer* di mana supplier berfungsi sebagai pemberi informasi dan customer berfungsi sebagai penerima informasi. Setelah mendefinisikan *bounded context* dan map konteks, subdomain yang dihasilkan dimodelkan, dan hasilnya adalah sebagai berikut:

1. Project
   a. Setiap project ditampilkan pada halaman admin
   b. Semua project ditampilkan pada landing page investor untuk investor dapat memilih project dan berinvestasi
   c. Setiap project ditampilkan secara detail kepada investor untuk memudahkan investor dalam mengambil keputusan berinvestasi
   d. Project dapat dibuat oleh peternak dan admin
   e. Setiap project yang dibuat akan difilter oleh sistem untuk memenuhi kriteria tertentu
   f. Setiap project ditampilkan dari sisi admin dan peternak untuk mereview kembali project yang telah dibuat
   g. Setiap project memiliki tenor atau jangka waktu project diselesaikan
2. Auth dan User
   a. Halaman login berfungsi menerima input username dan password dari pengguna
   b. Halaman register berfungsi menerima input dari investor yang ingin berinvestasi pada platform AliFarm Digital
   c. Setiap input dari halaman register diproses untuk divalidasi dan disimpan pada database
   d. Setiap pengguna dapat mengakhiri sesi login
   e. Setiap username dan password yang diinputkan pengguna dicocokkan dengan database yang ada
   f. Setiap informasi pengguna ditampilkan pada dashboard admin kecuali password
   g. Admin dapat menambahkan pengguna baru dengan tipe pengguna admin, peternak, atau investor
   h. Setiap pengguna yang ditambahkan admin divalidasi seperti halnya ketika pengguna dengan tipe investor melakukan registrasi dan disimpan pada database
   i. Jumlah pengguna dari masing masing tipe ditampilkan dalam bentuk diagram
   j. Setiap admin dapat merubah data pengguna sehingga terdapat halaman untuk merubah data pengguna untuk admin

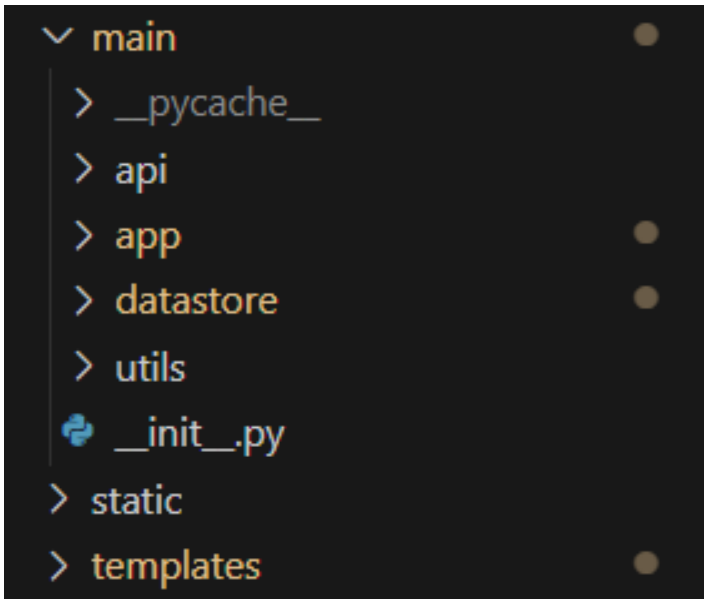
k. Setiap data pengguna yang dirubah oleh admin divalidasi oleh sistem dan disimpan pada database
l. Setiap admin dapat menghapus data pengguna

3. Transaksi
   a. Setiap transaksi yang dilakukan terhubung dengan *payment gateway*
   b. Setiap admin dapat melihat data semua transaksi
   c. Setiap data transaksi yang telah berhasil dilunasi atau telah kadaluarsa terupdate secara *real time* pada database
   d. Setiap investor dan admin dapat melihat data pengembalian modal. Untuk investor dapat melihat data pengembalian masing masing sedangkan untuk admin dapat melihat semua data pengembalian
   e. Setiap investor dapat melihat data transaksinya baik yang telah berhasil, pending, atau gagal
   f. Setiap admin dan investor dapat melihat detail transaksi yang menyajikan data nominal, waktu, dan kode transaksi

Setelah subdomain telah dimodelkan barulah dapat menggambarkan class diagram dan mengimplementasikan Clean Architecture seperti pada gambar di bawah ini:



Gambar 3. Class Diagram

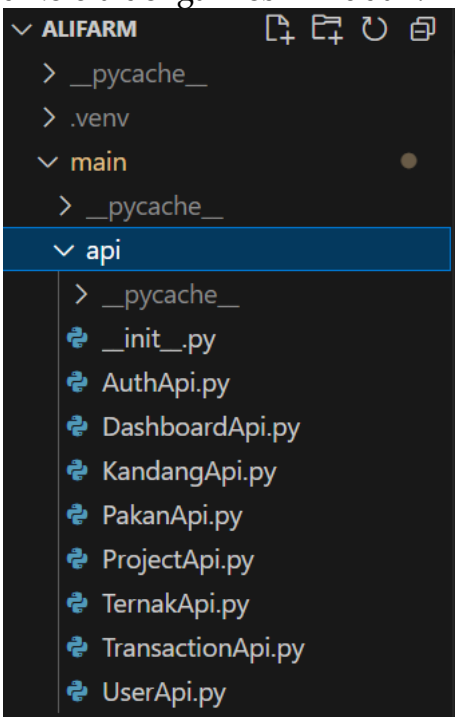


Gambar 4. Clean Architecture

Hasil dan Pembahasan

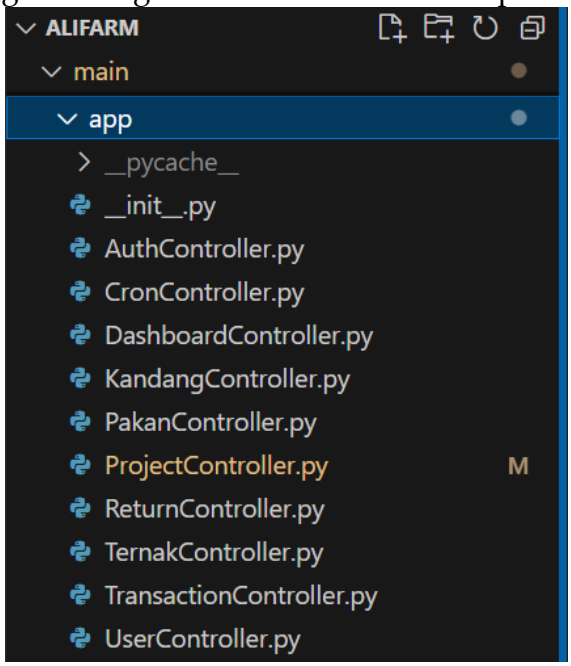
A. Arsitektur

Seperti yang ditunjukkan pada Gambar 4, pemisahan route aplikasi AliFarm Digital dipisahkan menjadi 8 file router. Pemisahan route ini merupakan bagian dari penerapan Clean Architecture, di mana setiap route bertanggung jawab untuk menangani permintaan (request) dari klien dan mengarahkan permintaan tersebut ke controller yang sesuai. Contoh route yang dapat dilihat adalah route untuk autentikasi pengguna, transaksi, dan manajemen proyek. Dengan pemisahan route yang jelas, aplikasi menjadi lebih terorganisir dan setiap endpoint API dapat dikelola dengan lebih mudah.



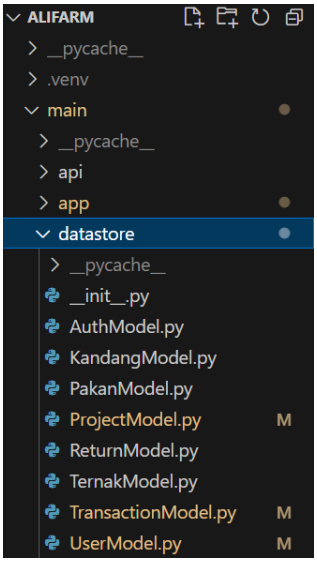
Gambar 5. Pemisahan Route Aplikasi

Seperti yang ditunjukkan pada Gambar 5, pemisahan core aplikasi yang berisi logika bisnis utama dari aplikasi. Bagian core ini mencakup berbagai controller yang mengelola alur bisnis aplikasi. Misalnya, AuthController mengelola proses autentikasi dan otorisasi, sedangkan TransactionController mengelola semua proses yang berkaitan dengan transaksi. Pemisahan core ini memastikan bahwa logika bisnis terpisah dari lapisan presentasi dan data, sehingga meningkatkan modularitas dan pemeliharaan aplikasi.



Gambar 6. Pemisahan Core Aplikasi

Seperti yang ditunjukkan pada Gambar 6, datastore yang berisi model-model yang digunakan untuk berinteraksi dengan database. Model ini merepresentasikan entitas-entitas dalam domain bisnis, seperti pengguna, proyek, transaksi, dan lainnya. Setiap model memiliki tanggung jawab untuk operasi CRUD (Create, Read, Update, Delete) terhadap entitas yang diwakilinya. Dengan memisahkan datastore, aplikasi dapat mengelola data secara terstruktur dan konsisten



Gambar 7. Pemisahan Datastore

B. Domain dan Subdomain

Domain *project* terdiri dari 7 subdomain, masing-masing merepresentasikan fungsi spesifik dalam proses aplikasi terkait dengan *project*. Subdomain ini termasuk:

1. project(): menangani perenderan antarmuka pengguna yang menampilkan semua data project dari sisi admin sistem
2. allProject(): menangani perenderan antarmuka pengguna yang menampilkan semua data project dengan status on progress untuk investor dapat berinvestasi
3. projectInvest(): menangani perenderan antarmuka pengguna yang menampilkan detail project tertentu untuk investor berinvestasi
4. create(): menangani perenderan antarmuka pengguna untuk menambahkan project baru
5. store(): menangani pemrosesan data dari input pengguna pada subdomain create()
6. detail(): menangani perenderan antarmuka pengguna yang menampilkan detail project dari sisi admin dan peternak
7. validateProject(): menangani pemrosesan data project yang telah diverifikasi atau ditolak oleh admin

ProjectApi.py

main > api > ProjectApi.py > ...

```python
from ..app import *

@app.route('/project', methods=['GET'])
async def project():
    return await ProjectController.project()

@app.route('/project/all', methods=['GET'])
async def allProjects():
    return await ProjectController.allProjects()

@app.route('/project/invest/<id_project>', methods=['GET'])
async def projectInvest(id_project):
    return await ProjectController.projectInvest(id_project)

@app.route('/project/create', methods=['GET'])
async def createProject():
    return await ProjectController.create()

@app.route('/project/store', methods=['POST'])
async def storeProject():
    return await ProjectController.store()

@app.route('/project/detail/<id_project>')
async def detailProject(id_project):
    return await ProjectController.detail(id_project)

@app.route('/project/status/pending', methods=['GET'])
async def pending():
    return await ProjectController.pending()
```

Gambar 8. Domain project berisi 7 subdomain

Domain auth dan user yang terdiri dari masing masing 5 dan 7 subdomain, masing-masing merepresentasikan fungsi spesifik dalam proses autentikasi otorisasi pengguna dan proses aplikasi terkait pengguna. Kedua domain ini tidak dapat dipisahkan secara konteks namun harus dipisahkan pada arsitektur kode untuk penerapan Clean Architecture. Subdomain ini termasuk:

1. login(): menangani perenderan antar muka pengguna untuk login
2. viewRegister(): menangani peerenderan antar muka pengguna untuk registrasi akun
3. registrasi(): menangani pemrosesan pendaftaran pengguna dan terhubung dengan domain datastore
4. authenticate(): menangani pemrosesan autentikasi pengguna

5. logout(): menangani penhapusan sesi pengguna
6. users(): menangani perenderan antar muka pengguna untuk menampilkan semua data pengguna dari sisi admin
7. addUser(): menangani perenderan antar muka pengguna untuk menambahkan pengguna baru oleh admin
8. storeUser(): menangani pemrosesan data pengguna baru yang diinputkan admin
9. detailUserApi(): menangani penyajian data dalam bentuk JSON untuk dikonsumsi oleh pustaka diagram
10. editUser(): menangani perenderan antar muka pengguna untuk merubah data pengguna dari sisi admin
11. updateUser(): menangani pemrosesan perubahan data pengguna oleh admin
12. deleteUser(): menangani pemrosesan penghapusan data pengguna oleh admin

AuthApi.py

main > api > AuthApi.py > seed

```python
from ..app import *

@app.route('/login', methods=['GET'])
async def login():
    return await AuthController.login()

@app.route('/register', methods=['GET'])
async def viewRegister():
    return await AuthController.formRegist()

@app.route('/register', methods=['POST'])
async def register():
    return await AuthController.register()

@app.route('/login', methods=['POST'])
async def authenticate():
    return await AuthController.authenticate()

@app.route('/logout', methods=['GET'])
async def logout():
    session.clear()
    return redirect('/')

```

Gambar 9. Domain auth berisi 5 subdomain

UserApi.py

main > api > UserApi.py > ...

```python
from ..app import *

@app.route('/users', methods=['GET'])
async def users():
    return await UserController.users()

@app.route('/user/create', methods=['GET'])
async def viewCreateUser():
    return await UserController.addUser()

@app.route('/user/store', methods=['POST'])
async def storeUser():
    return await UserController.storeUser()

@app.route('/api/user/<username>', methods=['GET'])
async def detailUserApi(username):
    return await UserController.detailUserApi(username)

@app.route('/user/edit/<username>', methods=['GET'])
async def viewEditUser(username):
    return await UserController.editUser(username)

@app.route('/user/update/<username>', methods=['POST'])
async def editUser(username):
    return await UserController.updateUser(username)

@app.route('/user/delete/<username>', methods=['POST'])
async def deleteUser(username):
    return await UserController.deleteUser(username)
```

Gambar 10. Domain User berisi 7 subdomain

Domain *transaction* terdiri dari 7 subdomain, masing-masing merepresentasikan fungsi spesifik dalam proses aplikasi terkait dengan transaksi. Subdomain ini termasuk:

1. payment(): menangani proses pembayaran dari investor yang terhubung dengan *payment gateway*
2. transaction(): menangani perenderan antarmuka pengguna yang menampilkan semua data transaksi baik yang berhasil maupun gagal
3. paymentNotification(): menerima notifikasi status pembayaran dari *payment gateway* dan merubah data status transaksi
4. pengembalian(): menangani perenderan antarmuka pengguna yang menampilkan semua data pengembalian dana ke investor dari project yang telah berhasil diselesaikan
5. permodalan(): menangani perenderan antarmuka pengguna yang menampilkan semua data permodalan atau investasi dari investor
6. detailTransaction(): menangani perenderan antarmuka pengguna yang menampilkan data transaksi tertentu berdasarkan parameter id_transaction

TransactionApi.py

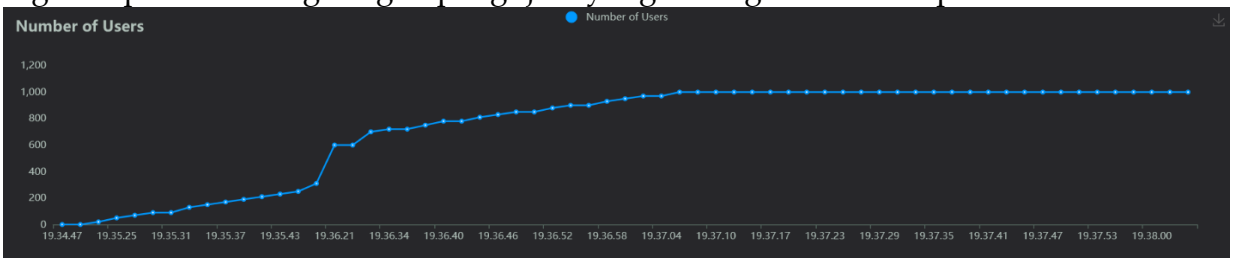
main > api > TransactionApi.py > detailTransaction

```python
from ..app import *

@app.route('/payment', methods=['POST'])
async def payment():
    return await TransactionController.payment()

@app.route('/transactions', methods=['GET'])
async def transactions():
    return await TransactionController.transactions()

@app.route('/payment/notification', methods=['POST'])
async def notification():
    return await TransactionController.paymentNotification()

@app.route('/pengembalian', methods=['GET'])
async def pengembalian():
    return await TransactionController.pengembalian()

@app.route('/permodalan', methods=['GET'])
async def permodalan():
    return await TransactionController.permodalan()

@app.route('/transaction/detail/<id_transaction>')
async def detailTransaction(id_transaction):
    return await TransactionController.detailTransaction(id_transaction)
```

Gambar 11. Domain transaction berisi 6 subdomain

C. Pengujian

Modul web service aplikasi AliFarm Digital telah berhasil dibuat. Untuk pengujian, penulis menggunakan metode loadtesting untuk mengetahui performa aplikasi yang telah dibuat dengan mengirimkan request. Pengujian dilakukan dengan menggunakan locust sebagai software tester. Selama locust dijalankan sesuai dengan skema *spawn user* yang digunakan, locust akan menghasilkan *swarm (user)* secara berkala seperti pada Gambar 12. Sedangkan spesifikasi lingkungan pengujian yang kami gunakan ada pada Tabel 2.



Gambar 12. Spawn User

Tabel 2. Lingkungan Pengujian

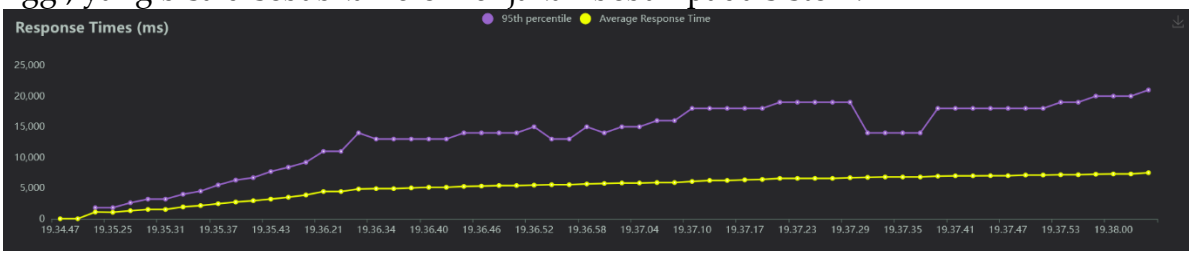
Spesifikasi	Deskripsi
CPU	AMD Ryzen 3 5300U
RAM	8GB DDR4
OS	Windows 10

Uji dilakukan dengan mensimulasikan 1 sampai 1000 pengguna login seperti grafik pada Gambar 3. Jumlah pengguna akan terus bertambah 10 pengguna setiap detik. Setiap pengguna melakukan operasi sesuai dengan end point yang ada pada aplikasi sehingga diperoleh nilai rata rata seperti pada Tabel 3. Detail dari hasil pengujian terdapat pada Gambar 5 yang menunjukkan response time dan Gambar 6 yang menunjukkan *request per second*.

Tabel 3. Rerata Hasil Pengujian

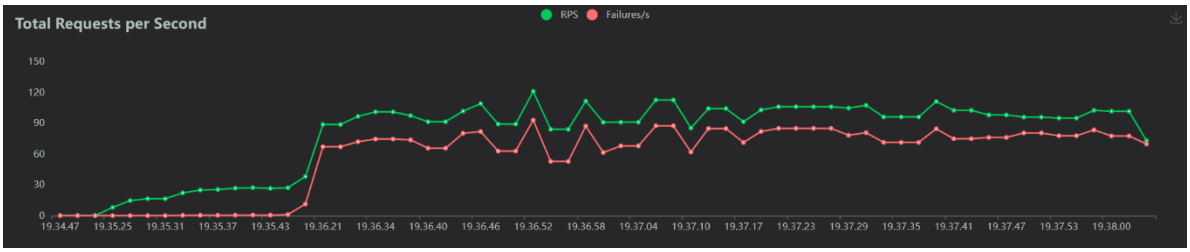
Variabel	Nilai
Average Response Time	7736.505
Min Response Time	196.6748
Max Response Time	35443.37
Average Content Size	12462.27

Garis kuning pada grafik response time mewakili rata-rata waktu respons, sementara garis ungu mewakili waktu respons pada persentil ke-99. Dari grafik tersebut, terlihat bahwa seiring dengan meningkatnya jumlah pengguna, rata-rata waktu respons tetap relatif stabil, berada di sekitar 5.000 ms. Namun, waktu respons pada persentil ke-99 mengalami peningkatan yang signifikan, mencapai lebih dari 20.000 ms. Hal ini menunjukkan bahwa sebagian kecil dari permintaan mengalami waktu respons yang jauh lebih tinggi, yang bisa disebabkan oleh lonjakan beban pada sistem.



Gambar 13. Response Time

Pada Gambar 14 menampilkan grafik request per second (RPS) dengan dua metrik: total RPS dan jumlah kegagalan (*failures*). Garis hijau menunjukkan jumlah total request yang diterima per detik, sementara garis merah menunjukkan jumlah request yang gagal per detik. Dari grafik ini, dapat dilihat bahwa RPS meningkat tajam seiring dengan bertambahnya jumlah pengguna, mencapai puncak di sekitar 120 request per detik. Namun, terdapat fluktuasi pada jumlah request yang gagal, dengan rata-rata kegagalan sekitar 40 request per detik. Hal ini menunjukkan bahwa seiring dengan meningkatnya jumlah pengguna, sistem mulai mengalami kesulitan dalam menangani beban, yang menyebabkan peningkatan jumlah kegagalan.



Gambar 14. Request per Second

Simpulan

Dari penelitian yang dilakukan terhadap implementasi Domain Driven Design (DDD) dan Clean Architecture dalam pengembangan aplikasi web service Alifarm Digital, dapat disimpulkan bahwa langkah-langkah pengembangan yang melibatkan analisis kebutuhan, perancangan berdasarkan DDD, dan implementasi dengan Clean Architecture memastikan efisiensi dalam pengembangan aplikasi. DDD digunakan untuk memodelkan subdomain seperti invest the project, detail proyek, dan pembagian hasil proyek, sementara Clean Architecture membantu dalam memisahkan tampilan, logika bisnis, dan sumber data untuk memudahkan pemeliharaan dan pengembangan aplikasi. Meskipun implementasi DDD dan Clean Architecture memberikan manfaat yang signifikan, penelitian juga mengidentifikasi beberapa tantangan. Lonjakan waktu respons pada persentil ke-99 yang tinggi dan fluktuasi dalam jumlah request yang gagal menunjukkan bahwa sistem Alifarm Digital masih menghadapi kesulitan dalam menangani beban yang tinggi. Grafik request per second yang menunjukkan peningkatan tajam dalam jumlah total request per detik seiring dengan bertambahnya pengguna menekankan pentingnya monitoring secara terus-menerus dan pemeliharaan sistem untuk memastikan kualitas layanan yang optimal. Dengan demikian, penelitian ini memberikan wawasan yang berharga tentang pentingnya penerapan DDD dan Clean Architecture dalam pengembangan aplikasi web service untuk memastikan efisiensi, skalabilitas, dan kualitas layanan yang optimal.

Daftar Pustaka

Ahmad, E. (2022). Towards unified management of software capstone projects in Saudi universities: a survey-based study. *Arab Gulf Journal of Scientific Research*, *40*(2), 118–138. https://doi.org/10.1108/AGJSR-04-2022-0037

Alulema, D. (2023). SI4IoT: A methodology based on models and services for the integration of IoT systems. *Future Generation Computer Systems*, *143*, 132–151. https://doi.org/10.1016/j.future.2023.01.023

Anhar, F. F., & Anggraeny, F. T. (2022). IMPLEMENTASI CLEAN ARCHITECTURE MVVM DAN REPOSITORY PATTERN UNTUK PENGEMBANGAN APLIKASI

ANDROID JUAL BELI BARANG BEKAS "SECONDHAND." *Scan : Jurnal Teknologi Informasi dan Komunikasi*, *17*(2), Article 2. https://doi.org/10.33005/scan.v17i2.3317

Budi, C. S., & Bachtiar, A. M. (n.d.). *IMPLEMENTASI ARSITEKTUR MICROSERVICES PADA BACKEND COMRADES*.

Blond, K. E. (2023). A decision-making framework for the kc-46a maintenance program. Proceedings - Annual Reliability and Maintainability Symposium, 2023. https://doi.org/10.1109/RAMS51473.2023.10088176

Chamari, L. (2023). An End-to-End Implementation of a Service-Oriented Architecture for Data-Driven Smart Buildings. IEEE Access, 11, 117261–117281. https://doi.org/10.1109/ACCESS.2023.3325767

Chessa, A. (2023). Data-Driven Methodology for Knowledge Graph Generation Within the Tourism Domain. IEEE Access, 11, 67567–67599. https://doi.org/10.1109/ACCESS.2023.3292153

Crowdfunding Pada Teknologi Keuangan Islam | SOSMANIORA: Jurnal Ilmu Sosial dan Humaniora. (n.d.). Retrieved May 16, 2024, from https://journal.literasisains.id/index.php/sosmaniora/article/view/441

Deljouyi, A. (2022). MDD4REST: Model-Driven Methodology for Developing RESTful Web Services. *International Conference on Model-Driven Engineering and Software Development*, 93–104. https://doi.org/10.5220/0011006300003119

Fajri, A. R. (2022). *Penerapan Design Pattern Mvvm Dan Clean Architecture Pada Pengembangan Aplikasi Android (Studi Kasus: Aplikasi Agree)*. https://dspace.uii.ac.id/handle/123456789/40624

Giner-Miguelez, J. (2022). Enabling Content Management Systems as an Information Source in Model-Driven Projects. *Lecture Notes in Business Information Processing*, *446*, 513–528. https://doi.org/10.1007/978-3-031-05760-1_30

Koblitz, J. (2023). MediaDive: the expert-curated cultivation media database. *Nucleic Acids Research*, *51*(1). https://doi.org/10.1093/nar/gkac803

Li, Q. (2022). Sharing platform of digital specimen of wood canker based on WebGIS in Xinjiang province: Architecture, design and implementation. *Proceedings - 2022 International Conference on Computers, Information Processing and Advanced Education, CIPAE 2022*, 102–106. https://doi.org/10.1109/CIPAE55637.2022.00029

Ling, Z. (2023). Semantic-driven construction of geographic entity association network and knowledge service. *Cehui Xuebao/Acta Geodaetica et Cartographica Sinica*, *52*(3), 478–489. https://doi.org/10.11947/j.AGCS.2023.20210349

Mandal, S. (2022). Designing a collection tree using Omeka. *Library Hi Tech News*, *39*(5), 16–21. https://doi.org/10.1108/LHTN-02-2022-0019

Miao, M. (2023). Digital Health Implementation Strategies Coproduced With Adults With Acquired Brain Injury, Their Close Others, and Clinicians: Mixed Methods Study With Collaborative Autoethnography and Network Analysis. *Journal of Medical Internet Research*, *25*(1). https://doi.org/10.2196/46396

Mufidah, D., & Setiawan, H. (2022). *Analisis Framing Berita Nasib Aset Indra Kenz Akibat Kasus Binomo Media Detik dan Tirto*. *6*.

Ngo, Q. M. (2022). In-Person Versus Telehealth Setting for the Delivery of Substance Use Disorder Treatment: Ecologically Valid Comparison Study. *JMIR Formative Research*, *6*(4). https://doi.org/10.2196/34408

Prasetyo, A. D., Kautsar, I. A., & Azizah, N. L. (2022). Rancang Bangun Aplikasi Pelaporan Fasilitas Umum Berbasis Web Service Dalam Rangka Menuju Sidoarjo Smart City Dan Open Data. *JIPI (Jurnal Ilmiah Penelitian Dan Pembelajaran Informatika)*, *7*(4), 1271–1280.

Prayoga, H. W., Akbar, R. J., & Fabroyir, H. (2021). Rancang Bangun Sistem MyITS Dorm Menggunakan Metode Domain Driven Design dan Onion Architecture. *Jurnal Teknik ITS*, *10*(2), A298–A305. https://doi.org/10.12962/j23373539.v10i2.69815

Ramadhan, M. F., & Zukhri, Z. (2023). PENGEMBANGAN REST API SISTEM UIIADMISI DENGAN MENGGUNAKAN PENDEKATAN DOMAIN DRIVEN DESIGN. *JURNAL ILMIAH INFORMATIKA*, *11*(02), Article 02. https://doi.org/10.33884/jif.v11i02.8017

Saifulloh, T., Kharisma, A. P., & Brata, D. W. (2023). Pengembangan Aplikasi Perangkat Bergerak Pencarian Partner Lomba berbasis Android menggunakan Clean Architecture. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, *7*(4), Article 4.

Stefanovska, E. (2022). Evaluating Micro Frontend Approaches for Code Reusability. *Communications in Computer and Information Science*, *1740*, 93–106. https://doi.org/10.1007/978-3-031-22792-9_8

Yang, H. (2022). Modeling of Internet of Things Service Platform Based on X Language. *Lecture Notes in Electrical Engineering*, *805*, 643–653. https://doi.org/10.1007/978-981-16-6320-8_65