

DOCUMENTATIERAPPORT

onderdeel van de BACHELORPROEF

Optimaliseren van Flutter Projectontwikkeling met Clean Architecture en SOLID Principles

Evaluatie van Clean Architecture en SOLID principes in Flutter: impact, voor- en nadelen, leercurve, en geschiktheid voor kleinere projecten.

Bachelor	Toegepaste Informatica
----------	------------------------

Keuzetraject Afstudeerrichting	Software Engineer
-----------------------------------	-------------------

Academiejaar	2023 - 2024
--------------	-------------

Student	Robin Monseré
---------	---------------

Interne begeleider	Wim Van Renterghem
--------------------	--------------------

Externe promotor	Glenn Ruysschaert
------------------	-------------------

Toelating tot bruikleen

De auteur(s) geeft (geven) de toelating dit documentatierapport (onderdeel van de bachelorproef) voor consultatie beschikbaar te stellen en delen van dit rapport te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de bepalingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit dit rapport.

3/06/2024

Woord vooraf

Voor u ligt mijn documentatierapport, een essentieel onderdeel van mijn bachelorproef binnen de opleiding Toegepaste Informatica. Gedurende dit onderzoek heb ik mij verdiept in de integratie van Clean Architecture en SOLID-principes binnen een Flutter-omgeving. Deze studie biedt inzicht in hoe deze methodologieën toegepast kunnen worden om robuuste, onderhoudbare en efficiënte mobiele applicaties te ontwikkelen.

Ik wil graag mijn oprechte dank uitspreken aan iedereen die heeft bijgedragen aan de realisatie van dit project. Speciale dank gaat uit naar Glenn Ruyschaert, CTO van Get Driven, die niet alleen de onderzoeksvraag voor deze bachelorproef heeft aangedragen maar ook waardevolle inzichten en feedback heeft geleverd. Ook wil ik graag de interne begeleider van Howest, Wim Van Renterghem bedanken, voor de begeleiding tijdens dit onderzoek.

Daarnaast wil ik mijn medestudenten, vrienden en familie bedanken voor hun steun en voor het zorgvuldig controleren van mijn werk op fouten.

Robin Monseré
Ingelmunster - 03/06/2024

Inhoudsopgave

Woord vooraf

1	Inleiding.....	5
1.1	Algemeen	5
1.2	Probleemstelling	5
1.3	Onderzoeksvraag	5
1.4	Experiment	5
2	Experiment.....	6
2.1	Onderzoek.....	6
2.1.1	Clean Architecture.....	6
2.1.2	Solid Principles	6
2.2	Uitvoering	8
2.3	Verschillen	9
2.3.1	Folderstructuur.....	9
2.3.2	Prestaties.....	10
2.3.3	App-grootte	11
2.3.4	Leercurve.....	11
3	Conclusie.....	12
	AI Engineering Prompts.....	13
	Referentielijst.....	14
	Bijlagen.....	15

1 Inleiding

1.1 Algemeen

Deze bachelorproef is tot stand gekomen door mijn stage bij Get Driven. Get Driven is de marktleider in het aanbieden van chauffeurs die u in uw eigen wagen vervoeren. Get Driven lanceerde de eerste "wij rijden u in uw wagen" app begin 2019 na ruim anderhalf jaar ontwikkelen. Deze app is ontwikkeld in Flutter en wordt nog steeds onderhouden en bijgewerkt.

Eén van de grootste uitdagingen waarmee ontwikkelaars worden geconfronteerd, is het structureren van de app op een manier die niet alleen effectief is voor het huidige moment, maar ook voorbereid is op toekomstige groei. Dit is precies waar mijn onderzoeksvraag om draait: de implementatie van "Clean Architecture".

1.2 Probleemstelling

Het probleem dat dit onderzoek wil aanpakken, richt zich op het beoordelen van de impact van Clean Architecture op de ontwikkeling van Flutter-projecten, inclusief het gebruik van SOLID principes. Het is van belang om te begrijpen hoe de implementatie van Clean Architecture en SOLID principes de ontwikkeling van Flutter-projecten kan beïnvloeden, en of deze benaderingen daadwerkelijk voordelen opleveren voor ontwikkelaars.

Hoe hebben Clean Architecture en SOLID principes invloed op het ontwikkelen van applicaties en wat zijn de voor- en nadelen?

1.3 Onderzoeksvraag

Optimaliseren van Flutter Projectontwikkeling met Clean Architecture en SOLID Principles.

1.4 Experiment

Om deze vraag te kunnen beantwoorden, heb ik beslist om eerst onderzoek te doen naar Clean Architecture en SOLID Principles. Na een grondig onderzoek, neem ik deze kennis mee om een Flutter project te maken en alles van CA en SOLID toe te passen.

Op deze app voer ik dan analyses uit en neem ik ook mijn persoonlijke ervaring mee in het creëren van de applicatie.

2 Experiment

2.1 Onderzoek

2.1.1 Clean Architecture

Clean Architecture is een softwarearchitectuurontwerp dat ervoor zorgt dat het ontwerp van een systeem robuust, onderhoudbaar en testbaar is. Het systeem werd bedacht door Robert C. Martin en het biedt een aanpak waarbij de architectuur in lagen wordt georganiseerd. Deze lagen zijn deel van een cirkel en het doel is dat de afhankelijkheden altijd van buiten naar binnen gericht zijn, oftewel van minder belangrijke details naar meer fundamentele aspecten van de applicatie. Elk van deze lagen heeft een specifieke verantwoordelijkheid en rol binnen het algehele systeem. Dit principe staat bekend als de 'Dependency Rule' en is cruciaal voor het behouden van een mooie scheiding en onafhankelijkheid tussen de verschillende onderdelen van de applicatie.

De binnenste laag, vaak de 'Entities' laag genoemd, bevat de bedrijfslogica en entiteiten. Deze laag is volledig onafhankelijk van externe invloeden zoals de database. Naarmate we naar buiten bewegen, vinden we de 'Use Cases' laag, die de toepassingsspecifieke bedrijfsregels implementeert. Dan, de 'Interface Adapters' laag, die gegevens converteert tussen de formaten die nodig zijn voor de use cases en de externe agentschappen zoals de database of het web. De buitenste laag, de 'Frameworks & Drivers', handelt alle interacties af met externe systemen en frameworks, en biedt zaken zoals database toegang en de webinterface.

Deze manier zorgt ervoor dat een wijziging in een extern systeem, zoals een aanpassing aan de database, een minimale impact heeft op de kernlogica van de applicatie. Dit maakt het systeem niet alleen robuuster en flexibeler, maar verhoogt ook de onderhoudbaarheid en maakt het systeem makkelijker te testen.

2.1.2 Solid Principles

SOLID zijn 5 object georiënteerde ontwerpprincipes die helpen met het creëren van onderhoudsvriendelijke projecten. Om deze juist te kunnen toepassen, is het belangrijk dat ik deze volledig onder de knie heb.

S: Single Responsibility Principle (SRP)

Een klasse heeft maar één verantwoordelijkheid, dit betekent dat die klasse dan ook maar 1 reden heeft om aangepast te worden. Dit leidt tot propere, verstaanbare en beter onderhoudbare code. Het risico op fouten wordt verminderd en het is ook makkelijker om uit te breiden.

O: Open/Closed Principle

Klassen, functies en modules moeten open zijn voor uitbreiding maar gesloten voor wijzigingen, dit betekent dat de functionaliteit van een onderdeel uitbreidbaar moet zijn zonder dat bestaande code of functionaliteit moet worden aangepast.

Het idee is om bestaande code uit te breiden in plaats van aan te passen, dit minimaliseert het risico op het maken van fouten in de bestaande code.

Dit kan toegepast worden door gebruik te maken van abstracties en interfaces.

L: Liskov Substitution Principle

LSP is een ontwerpprincipe dat stelt dat objecten van een subklasse moeten kunnen worden vervangen door objecten van de superklasse zonder dat er problemen oplopen in het programma. Met andere woorden, als S een subtype is van T, dan moet een object van type T vervangen kunnen worden door een object van type S, zonder dat dit de correctheid van het programma in gevaar brengt.

I: Interface Segregation Principle

ISP is een ontwerpprincipe in de objectgeoriënteerde programmeerwereld dat stelt dat een klasse geen afhankelijkheden moet hebben van interfaces die het niet gebruikt. Het doel van het ISP is om interfaces te ontwerpen die specifiek zijn voor de behoeften van de klasse, waardoor onnodige afhankelijkheden worden vermeden. Door interfaces op te splitsen in kleinere, gerichte interfaces, kunnen klassen alleen die methoden implementeren die relevant zijn voor hun gebruik, waardoor de complexiteit van de implementatie wordt verminderd.

D: Dependency Inversion Principle

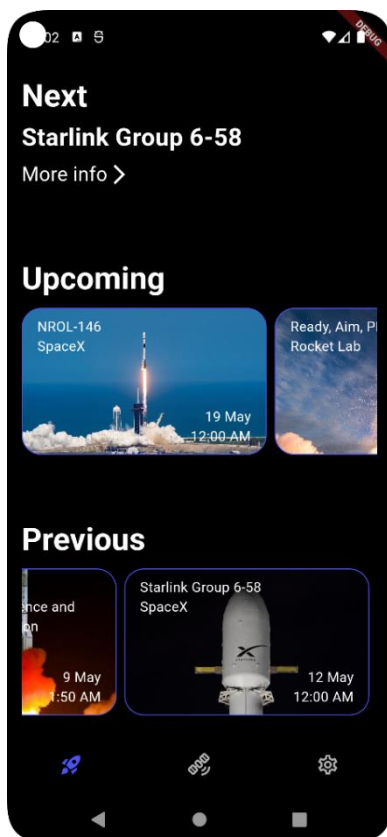
High level modules mogen niet afhankelijk zijn van low level modules, maar beide moeten afhankelijk zijn van abstracties. Bovendien zouden deze abstracties niet afhankelijk moeten zijn van details, maar de details zouden afhankelijk moeten zijn van abstracties. Dit leidt tot een minder directe koppeling tussen softwarecomponenten en verbetert de mogelijkheid om componenten te hergebruiken en te onderhouden.

2.2 Uitvoering

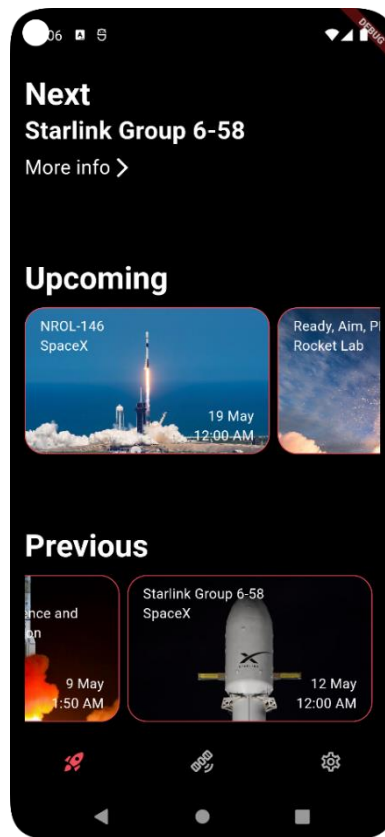
Om een grondige vergelijking te kunnen maken tussen het wel of niet gebruik van SOLID en Clean Architecture, is het belangrijk om beide methoden toe te passen op een gelijkaardige applicatie. Voor deze reden heb ik de SpaceGaze applicatie gemaakt. De app haalt de meest recente data op in verband met raketlanceringen en geeft dit weer aan de gebruiker.

Naast de homepage, is er ook nog een pagina met meer gedetailleerde info over een lancering. Deze pagina is ook op beide apps visueel volledig gelijk.

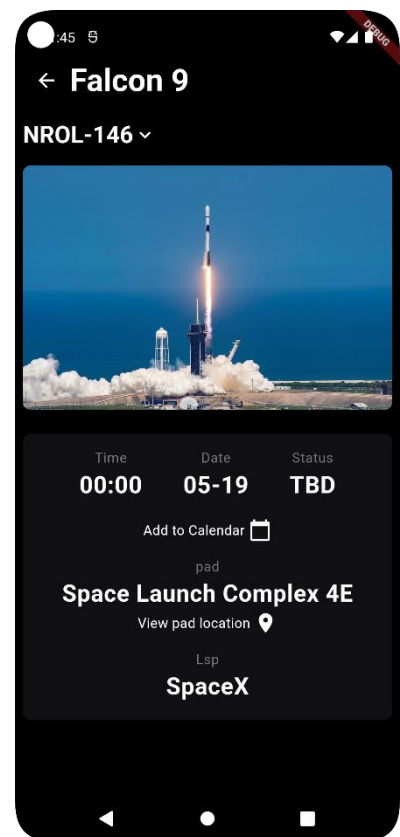
De broncode voor beide apps is te vinden op Github (zie bijlagen). Visueel is er zo goed als geen verschil tussen beide applicaties, daarom is het accentkleur blauw bij de app waar SOLID en Clean Architecture is toegepast en rood waar dit niet het geval is.



Figuur 1: SpaceGaze applicatie met Solid Principles and Clean Architecture



Figuur 2: SpaceGaze applicatie zonder Solid Principles and Clean Architecture



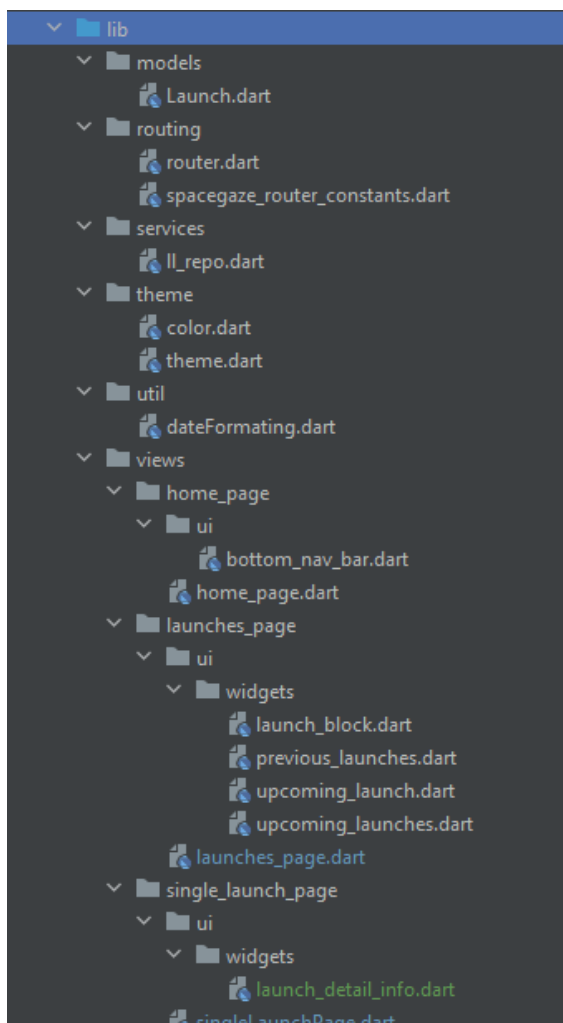
Figuur 3: Info pagina

2.3 Verschillen

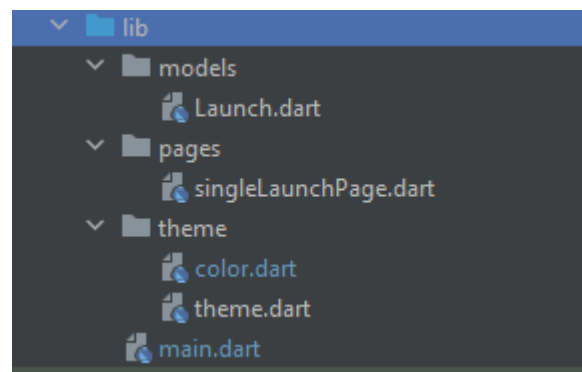
2.3.1 Folderstructuur

Het verschil tussen beide applicaties is echter wel heel duidelijk te zien in de folder structuur van beide projecten. Een van de belangrijkste aspecten van zowel SOLID als Clean Architecture is de nadruk op de scheiding van verantwoordelijkheden. Elk onderdeel van de software heeft een specifieke verantwoordelijkheid en is geïsoleerd van anderen. Dit betekent dat in plaats van enkele monolithische modules, de applicatie wordt opgesplitst in meerdere kleinere, beheersbare onderdelen. Elk van deze onderdelen kan een eigen map of submap vereisen om de code overzichtelijk en onderhoudbaar te houden.

Dit betekent ook dat in grotere projecten, het veel makkelijker is om code terug te vinden, en zorgt ervoor dat verschillende teams tegelijkertijd aan verschillende onderdelen van de applicatie kunnen werken zonder elkaar in de weg te zitten. Deze structuur zorgt er ook voor dat het hergebruiken van code eenvoudiger is en om veranderingen door te voeren zonder onverwachte bijwerkingen in andere delen van de applicatie.



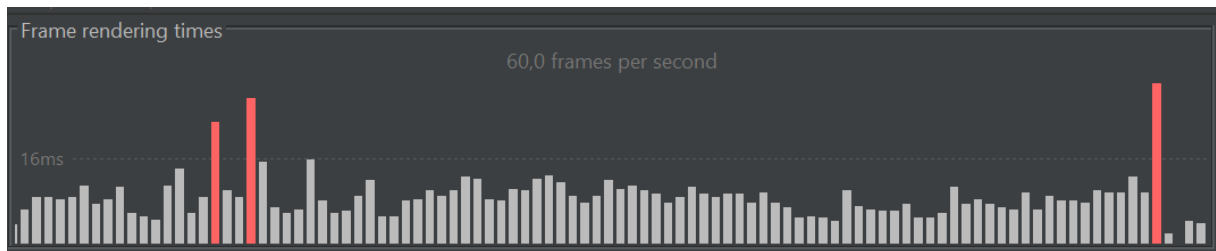
Figuur 4: Folder structuur bij SOLID en Clean Architecture



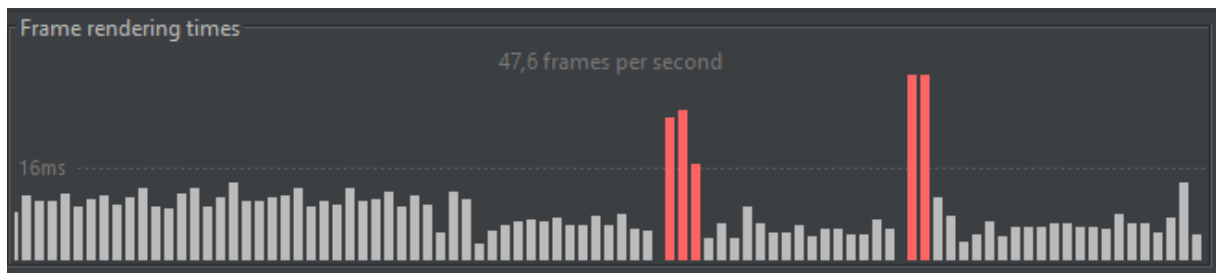
Figuur 5: Folder structuur zonder SOLID en Clean Architecture

2.3.2 Prestaties

Omdat beide applicaties uiteindelijk relatief klein zijn, is het moeilijk om significante prestatieverschillen tussen hen te beoordelen. Echter, als gebruiker van de applicatie is er weinig verschil merkbaar in de prestaties. Maar als we naar de onderstaande afbeeldingen kijken, zien we wel een duidelijk verschil.



Figuur 6: Frame rendering times bij Clean Architecture en SOLID



Figuur 7: Frame rendering times zonder Clean Architecture en SOLID

De afbeeldingen tonen de frame rendering tijden zoals gemeten met de Flutter Performance Profiler. Flutter streeft ernaar dat applicaties een framerate van 60 frames per seconde (fps) halen, wat betekent dat elke frame niet meer dan 16 milliseconden (ms) mag duren om gerenderd te worden. De Flutter Performance Profiler biedt een uitgebreide set tools om de prestaties van een applicatie te analyseren.

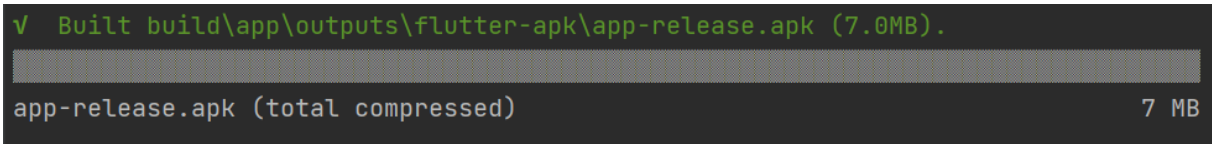
In de grafieken vertegenwoordigt elke balk de tijd die nodig is om een enkele frame te renderen. De rode balken geven aan dat de frame rendering tijd de 16 ms overschrijdt, wat resulteert in een lagere framerate en minder vloeiende animaties. De witte balken daarentegen geven frames weer die binnen de gewenste 16 ms zijn gerenderd.

Zoals te zien is in de grafieken, zijn er enkele rode pieken die aangeven dat sommige frames langer dan 16 ms duren om te renderen. Dit betekent dat er af en toe een vertraging optreedt in de rendering, wat kan leiden tot merkbare haperingen voor de gebruiker. Ondanks dat deze pieken relatief zeldzaam zijn, kunnen ze toch de gebruikerservaring beïnvloeden. In figuur 6 is het duidelijk te zien dat het gemiddelde aantal frames per seconde geen 60 maar 47,6 is.

Deze prestatieproblemen kunnen deels worden toegeschreven aan de structuur van de applicatie. Het toepassen van SOLID en Clean Architecture principes speelt een cruciale rol bij het creëren van efficiënte en onderhoudbare code. Deze principes helpen bij het scheiden van verantwoordelijkheden en zorgen ervoor dat de code goed georganiseerd is, wat bijdraagt aan betere prestaties. Echter, het is ook belangrijk om te erkennen dat de prestaties kunnen variëren afhankelijk van het gebruikte apparaat. In deze tests is een OnePlus 7 Pro gebruikt. Het is mogelijk dat andere smartphones, met verschillende hardwareconfiguraties, andere resultaten laten zien.

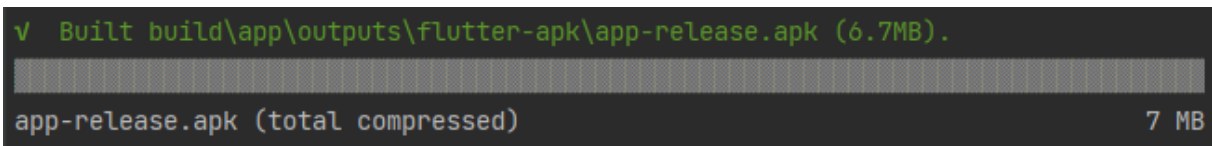
2.3.3 App-grootte

Persoonlijk dacht ik dat het gebruik van SOLID en Clean Architecture een merkbaar verschil zou geven in de grootte van de app. Echter, aangezien de applicatie relatief klein is, is er weinig verschil in de uiteindelijke bestandsgrootte. Zoals te zien in onderstaande afbeeldingen, is er slechts een verschil van 0,3 MB tussen de twee versies van de app. Dit verschil is waarschijnlijk te wijten aan de toevoeging van een extra package, de `go_router` package, in plaats van de SOLID principes of Clean Architecture.



```
✓ Built build\app\outputs\flutter-apk\app-release.apk (7.0MB).  
app-release.apk (total compressed) 7 MB
```

Figuur 8: App-grootte bij Clean Architecture en SOLID



```
✓ Built build\app\outputs\flutter-apk\app-release.apk (6.7MB).  
app-release.apk (total compressed) 7 MB
```

Figuur 9: App-grootte zonder Clean Architecture en SOLID

2.3.4 Leercurve

De leercurve om Clean Architecture en SOLID toe te passen was een stuk groter dan ik verwacht had. Het kostte aanzienlijk meer tijd en moeite om deze concepten volledig te begrijpen en correct toe te passen in de applicatie. Vooral het doorgronden van de abstracties en het strikt naleven van de SOLID-principes vereiste veel tijd. Ondanks deze initiële moeilijkheden, begon ik na verloop van tijd de voordelen te zien van een goed gestructureerde code die voldoet aan deze principes.

Het toepassen van Clean Architecture en SOLID-principes heeft niet alleen mijn eigen begrip van het project verbeterd, maar het maakt ook toekomstige aanpassingen veel eenvoudiger. Als ik later extra functionaliteiten wil toevoegen of als iemand anders aan het project moet werken, zorgt deze gestructureerde aanpak ervoor dat de code gemakkelijker te begrijpen en te onderhouden is. Dit resulteert uiteindelijk in een meer robuust en schaalbaar systeem, waardoor de aanvankelijke investering in de leercurve zich op de lange termijn zeker terugbetaalt.

3 Conclusie

De studie naar het optimaliseren van Flutter-projectontwikkeling met Clean Architecture en SOLID-principes heeft aangetoond dat deze methodologieën significante voordelen bieden, vooral voor grotere projecten, maar ook waardevol kunnen zijn voor kleinere projecten.

De gekozen app, SpaceGaze, was relatief klein, wat betekende dat de voordelen van Clean Architecture en SOLID minder duidelijk waren in vergelijking met wat zichtbaar zou zijn geweest bij grotere projecten. Voor kleinere projecten kan de extra structuur en abstracties aanvankelijk overbodig lijken en meer werk met zich meebrengen dan bij eenvoudiger architecturen.

Je hebt een app nodig met meerdere modules zoals gebruikersbeheer, productbeheer, bestel- en betalingssystemen, die allemaal baat zouden hebben bij een duidelijke scheiding van verantwoordelijkheden en robuuste, onderhoudbare code om de volledige voordelen van Clean Architecture en SOLID-principes te realiseren. Dan zou makkelijker zijn om de verschillen te merken.

AI Engineering Prompts

“Kan je deze tekst controleren op fouten”

“Hoe kan SOLID of Clean Architecture gebruikt worden in deze code” + code snippet

“Herschrijf deze lijst met bronnen naar het IEEE formaat, vb:”

“Hoe kan ik de prestaties van deze functie verbeteren? + code”

“Welke SOLID-principes worden geschonden in deze code, en hoe kan ik dat oplossen?”

“Welke refactoringsuggesties heb je voor deze code om de leesbaarheid te verbeteren?”

“Documenteer dit stukje code”

“Gebruikt deze code het Open Closed principe?”

Verschillende keren gebruikt om bugs op te lossen tijdens de ontwikkeling van de applicaties.

Referentielijst

A. Krishna, "What is SOLID? Principles for Better Software Design," freecodecamp.org, 22 Mar. 2024. <https://www.freecodecamp.org/news/solid-principles-for-better-software-design/> [Geraadpleegd: 22 maart 2024].

Amandeep Singh, "A Deep Dive into Clean Architecture and Solid Principles" medium.com, 30 Oktober 2023. https://medium.com/@unaware_harry/a-deep-dive-into-clean-architecture-and-solid-principles-dcdcec5db48a. [Geraadpleegd: 23 maart 2024].

Y. Abdulrahman, "Clean Architecture in Flutter | MVVM | BloC | Dio," medium.com, 23 Mar. 2024. <https://medium.com/@yamen.abd98/clean-architecture-in-flutter-mvvm-bloc-dio-79b1615530e1>. [Geraadpleegd: 23 maart 2024].

Flutter Guys, "SOLID principles in Dart," YouTube, 22 Mar. 2024. https://youtu.be/A5Q5F7ICquQ?si=CJF5drfHb2GqB6_Q. [Geraadpleegd: Mar. 22, 2024].

CodeOpinion, "Clean Architecture Example & Breakdown - Do I use it?" YouTube, 22 Mar. 2024. https://www.youtube.com/watch?v=Ys_W6MyWOCw. [Geraadpleegd: Mar. 22, 2024].

Flutter, "documentation," flutter.dev, 23 Mar. 2024. <https://docs.flutter.dev/>. [Geraadpleegd: Mar. 23, 2024].

The Space Devs, "Space Devs API documentation," thespacedevs.com, 5 Apr. 2024. [Online]. <https://ll.thespacedevs.com/2.2.0/swagger/>. [Geraadpleegd: Apr. 5, 2024].

Wikipedia, "Robert C. Martin," wikipedia.org, 5 Apr. 2024. https://en.wikipedia.org/wiki/Robert_C._Martin. [Geraadpleegd: 5 april 2024].

Mihaly Nagy, "Thoughts on Clean Architecture," medium.com, 5 Apr. 2024. [Online]. <https://medium.com/android-news/thoughts-on-clean-architecture-b8449d9d02df>. [Geraadpleegd: 5 april 2024].

Filip Hracek, "Performance testing of Flutter apps" medium.com, 19 Apr. 2024. <https://medium.com/flutter/performance-testing-of-flutter-apps-df7669bb7df7>. [Geraadpleegd: 19 april 2024].

Get Driven, <https://getdriven.app/nl>. [Geraadpleegd: 17 maart 2024].

"SOLID," Wikipedia, Nov. 13, 2023. <https://nl.wikipedia.org/wiki/SOLID> [Geraadpleegd: 31 mei 2024].

Bijlagen

Github repo van het “clean” project

https://github.com/robinmonsere/spacegaze_bap_clean

Github repo van het “normal” project

https://github.com/robinmonsere/spacegaze_bap_normal