Onderzoeksvoorstel

State management in Flutter: een vergelijkende studie.

Bachelorproef, 2022-2023

Angela Degryse

E-mail: angela.degryse@student.hogent.be
Co-promotor: T. Alenus (The mobility Factory, tim.alenus@themobilityfactory.coop)

Samenvatting

In dit onderzoek worden de benaderingen van state management vergeleken met elkaar op vlak van performantie en ook de complexiteit ervan. Dit wordt onderzocht om de performantie van de applicatie van The Mobility Factory te optimaliseren. Dit onderzoek was nodig omdat naarmate de complexiteit van de applicatie toenam, de applicatie minder goed presteerde. Om dit te bereiken wordt een proof-of-concept gemaakt door een specifieke functie van de TMF-applicatie te implementeren, waarbij elke versie een andere benadering van staatsbeheer hanteert. De applicaties worden met elkaar vergeleken o.b.v. CPU-gebruik, opstartsnelheid, applicatiegrootte... De bevindingen zullen de best presterende state management benadering ontdekken en ook het type state management dat het beste past bij de applicatie van TMF. Er wordt verwacht dat Bloc het best benadering is van state management wat betreft complexiteit van de code en prestaties.

Keuzerichting: Mobile & Enterprise development
Sleutelwoorden: Flutter, State management, performantie

Inhoudsopgave

1. Introductie

Smartphones zijn een integraal onderdeel van ons dagelijks leven geworden. Daarom worden er steeds meer vraag om mobiele applicaties te ontwikkelen. Ook wordt cross-platform ontwikkelen steeds aantrekkelijker voor ontwikkelaars omdat ze in een kortere tijd applicaties kunnen ontwikkelen voor meerdere platformen.

The Mobility Factory (TMF) is een coöperatie die IT-oplossingen biedt aan bedrijven die elektrische autodelen diensten aanbieden. De applicaties worden door de TMF-ontwikkelaars in Flutter geïmplementeerd. Omdat de applicaties van TMF steeds groter en complexer worden, werkt de applicatie steeds minder performant. Daarom kwam de vraag om de performantie van hun applicatie te optimaliseren.

In Flutter zijn er tal van mogelijkheden voor het beheren van states. Afhankelijk van de complexiteit van de applicatie moet er overwogen worden welke state management benadering het best bij de applicatie past en of het de meest performante optie is. In dit onderzoek wordt eerst de verschillende soorten state management benaderingen grondig besproken en daarna worden ze vergeleken met elkaar op basis van prestatie en code complexiteit aan de hand van een proof-of-concept. Hieruit wordt de onderzoeksvraag dus gecreëerd: *Op welk manier kan states bijgehouden worden in Flutter en welk past het bij bij de applicatie van The Mobility Factory?*. De hoofdonderzoeksvraag kan verder verdeeld worden in de volgende deelonderzoeksvragen:

- Welk benadering van state management is het performantste qua cpu-gebruik, opstartsnelheid, geheugengebruik...?
- Hoeveel moeite zal het The Mobility Factory kosten om de verschillende benaderingen van state management te implementeren? Hoe complex is het om te integreren in de bestaande code?

2. State-of-the-art

Flutter is een open source raamwerk van Google voor het bouwen van multi-platform applicaties vanuit één codebase, of met andere woorden een cross-platform raamwerk. Flutter werd in 2018 uitgebracht en wordt ondertussen door meer dan

twee miljoen ontwikkelaars gebruikt om meer dan 350 duizend Flutter applicaties te ontwikkelen. [1]

2.1. Application performance, een onderdeel van User Experience

Applicatieperformantie is de werkelijke performantie die wordt ervaren door de eindgebruikers van de applicatie, zoals de gemiddelde responstijd bij normale belasting of piekbelasting (Rouse, 2014).

Door een onderzoek van Google wordt er vastgesteld dat naarmate de laadsnelheid oploopt, hoe meer geneigd de gebruikers zijn om de website te verlaten (An, 2010). Bij een laadsnelheid van meer dan 3 seconden van een webpagina, vertrekt 53% van de gebruikers bij het inladen van de webpagina.

Volgens Nielsen (2010), een UX-expert, is responsiviteit van een applicatie een basisregel voor het ontwerp van gebruikersinterfaces. De gebruikers moeten snel door de applicatie kunnen navigeren. Volgens Nielsen is een laadsnelheid tussen 1 en 10 seconden aanvaardbaar. Een laadsnelheid van langer dan 10 seconden kan ervoor zorgen dat de gebruiker afgeleid wordt waardoor het moeilijker is om geavanceerde taken tot een goed einde te brengen tegen dat de applicatie reageert.

2.2. Technieken om applicatie performantie te verbeteren

Flutter werd ontwikkeld om performant te zijn. Maar de prestatie van een Flutter applicatie kan sterk beïnvloed worden door hoe de applicatie ontworpen en geïmplementeerd wordt. [2]

Dit zijn 3 voorbeelden om de prestatie van een Flutter applicatie te verbeteren:

1. **Memory management**
   Het memory management in Flutter wordt afgehandeld door de Dart Virtual Machine (DVM), die een garbage collector heeft die automatisch ongebruikte geheugen terughaalt. Er kunnen echter nog steeds geheugenlekken optreden als objecten niet op de juiste manier uit het geheugen worden vrijgegeven, waardoor ze in de heap blijven hangen en onnodige bronnen verbruiken. Om geheugenlekken te vermijden moet de Dispose methode gebruikt worden om onnodige objecten te verwijderen. (Sabbagh, 2023)

2. **Netwerk optimalisatie**
   Om netwerk prestatie te verhogen moet er gebruik gemaakt worden van efficiënte netwerk libraries zoals Dio. Ook door de gegevens te cachen op het apparaat van de gebruiker kan er onnodige netwerkverzoeken vermeden worden en wordt de latentie verminderd. [3]

3. **State management**
   Binnen dit onderzoek wordt er gefocust op dit onderdeel. Dit wordt verder beschreven in het volgende puntje 2.4

2.3. State management binnen Flutter

Flutter beschrijft een state als informatie die enerzijds synchroon kan worden gelezen wanneer de widget wordt gebouwd en anderzijds kan veranderen tijdens de levensduur van de widget. [4]

States worden niet van parent widget naar child widget doorgegeven. De state van een widget wordt beheerd door zichzelf of door een parent widget en als de state van de parent widget wijzigt, moet alle widgets onder die parent widget ook herladen worden. Daarom kan het apart beheren van states zorgen voor efficiëntere updates en betere prestaties van de applicatie, aangezien enkel betreffende widgets heropgebouwd moeten worden. Deze functie is vooral gunstig voor complexe applicaties, waar het herbouwen van de volledige widgetstructuur tijdrovend kan zijn en kan leiden tot ondermaatse prestaties. Het is dus van groot belang dat de juiste benadering van state management wordt toegepast, zodat het optimaal presteert en een naadloze gebruikerservaring garandeert. (siddhardha, 2023)

Dit zijn de mogelijke benaderingen van state management in Flutter:

- SetState
- Provider
- GetX
- RiverPod
- Redux
- BLoC / Rx
- MobX

2.4. Relevante studies

State management wordt door het Flutter-team als complex bestempeld en er wordt veel gediscussieerd onder de Flutter-ontwikkelaars over wat de beste aanpak is.

Een vergelijkbaar onderzoek dat reeds werd gedaan over state management is door De Vriest (2019). Hoewel zijn onderzoek maar 3 jaar oud is, is Flutter heel sterk geëvolueerd sindsdien. Zijn onderzoek concludeerde dat Provider het meest

[1] https://flutter.dev/
[2] https://docs.flutter.dev/perf/best-practices
[3] https://www.linkedin.com/pulse/flutter-performance-optimization-techniques-best-practices/
[4] https://api.flutter.dev/flutter/widgets/State-class.html

performant is voor een simpel CRUD-applicatie. Maar in zijn onderzoek waren er tekortkomingen, zoals het enkel testen op een Android-toestel en ook werden data niet van een API opgehaald, wat niet realistisch is voor de realiteit.

Een andere vergelijkbaar onderzoek werd door Slepnev (2020) geschreven. In zijn onderzoek werd de verschillende benaderingen van state management gegroepeerd en vergeleken. In zijn onderzoek werden er criteria opgesteld om het best passende benadering te kiezen voor de verschillende use cases.

3. Methodologie

Eerst zal een vergelijkende studie uitgevoerd worden door de verschillende benaderingen van state management op te sommen en hun voor- en nadelen te vergelijken. Uit de gevonden state management benaderingen zullen er vier uitgekozen worden om toe te passen in het proof-of-concept.

In het methodologiegedeelte zullen verschillende versies van een bepaalde feature van de applicatie van TMF geïmplementeerd worden, waarbij voor elke versie een andere benadering van state management wordt toegepast. De applicatie van TMF die nagemaakt zal worden is een beheerapplicatie om reservaties en gebruikers te beheren.

De prestaties van de applicatie zullen getest worden op basis van CPU-gebruik, laadsnelheid en nodige opslag voor de applicatie. Er wordt ook een afweging gemaakt tussen de prestaties die worden geleverd door state management en de complexiteit van de integratie ervan in de bestaande code.

Het proof-of-concept zal geïmplementeerd worden met Flutter op Visual Studio Code. De applicatie van TMF wordt voornamelijk gebruikt op de web, daarom zal de performantie van de proof-of-concept getest worden als een webapplicatie en ook op een fysieke iOS-toestel. De code complexiteit van de state managements zullen vergeleken worden op basis van aantal nodige lijnen code. Om de performantie te meten zal er automated integration testen geschreven en uitgevoerd worden aan de hand van Flutter Driver. Bij elk uitvoering van de testen zullen de performantie gemeten worden. [5]

4. Verwachte resultaten

Uit eerdere onderzoeken blijkt dat Provider het best benadering van state management is op vlak van complexiteit en ook op prestatie vlak. Dit is ook de state management benadering die aanbevolen wordt door Flutter. Maar omdat de applicatie van TMF complexer is, wordt er verwacht dat Bloc een betere benadering van state management zal zijn.

5. Discussie, conclusie

Er zijn talloze manieren om de prestaties van een Flutter-applicatie te verhogen. Een van de manieren is om het meest geschikte state management toe te passen, afhankelijk van de complexiteit van de geïmplementeerde applicatie.

Het doel van deze studie is om TMF-ontwikkelaars een idee te geven van welke state management opties beschikbaar zijn in Flutter en wat het meest performant is voor hun applicatie.

Referenties

An, D. (2010). Find out how you stack up to new industry benchmarks for mobile page speed. Verkregen juli 21, 2023, van https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/mobile-page-speed-new-industry-benchmarks/

De Vriest, J. (2019). *Een vergelijkende studie tussen verschillende benaderingen van State Management in Flutter* [Bachelor's Thesis]. Hogeschool Gent. Verkregen augustus 5, 2023, van https://scriptie.hogent.be/2019-2020/339_201640029_PBA-TIN_scriptie.pdf

Nielsen, J. (2010). Website Response Times. Verkregen juli 21, 2023, van https://www.nngroup.com/articles/website-response-times/

Rouse, M. (2014). Application performance. Verkregen juli 21, 2023, van https://www.techopedia.com/definition/30457/application-performance

Sabbagh, A. (2023). Flutter Performance Optimization Techniques Best Practices. Verkregen augustus 16, 2023, van https://blog.devgenius.io/identifying-and-handling-memory-leaks-in-flutter-apps-1ad5e7d499e7

siddhardha. (2023). My Experience with Flutter State Management. Everything You Need to Know About it. Verkregen augustus 16, 2023, van https://levelup.gitconnected.com/everything-you-need-to-know-about-flutter-state-management-from-my-experience-19f56729f3c4#:~:text=State%2C%20in%20Flutter%2C%20refers%20to,itself%20or%20its%20parent%20widget.

Slepnev, D. (2020). *State Management approaches in Flutter* [Bachelor's Thesis]. South-Eastern Finland University of Applied Sciences. Verkregen augustus 15, 2023, van https://www.theseus.fi/bitstream/handle/10024/355086/Dmitrii_Slepnev.pdf

[5] https://docs.flutter.dev/cookbook/testing/integration/profiling