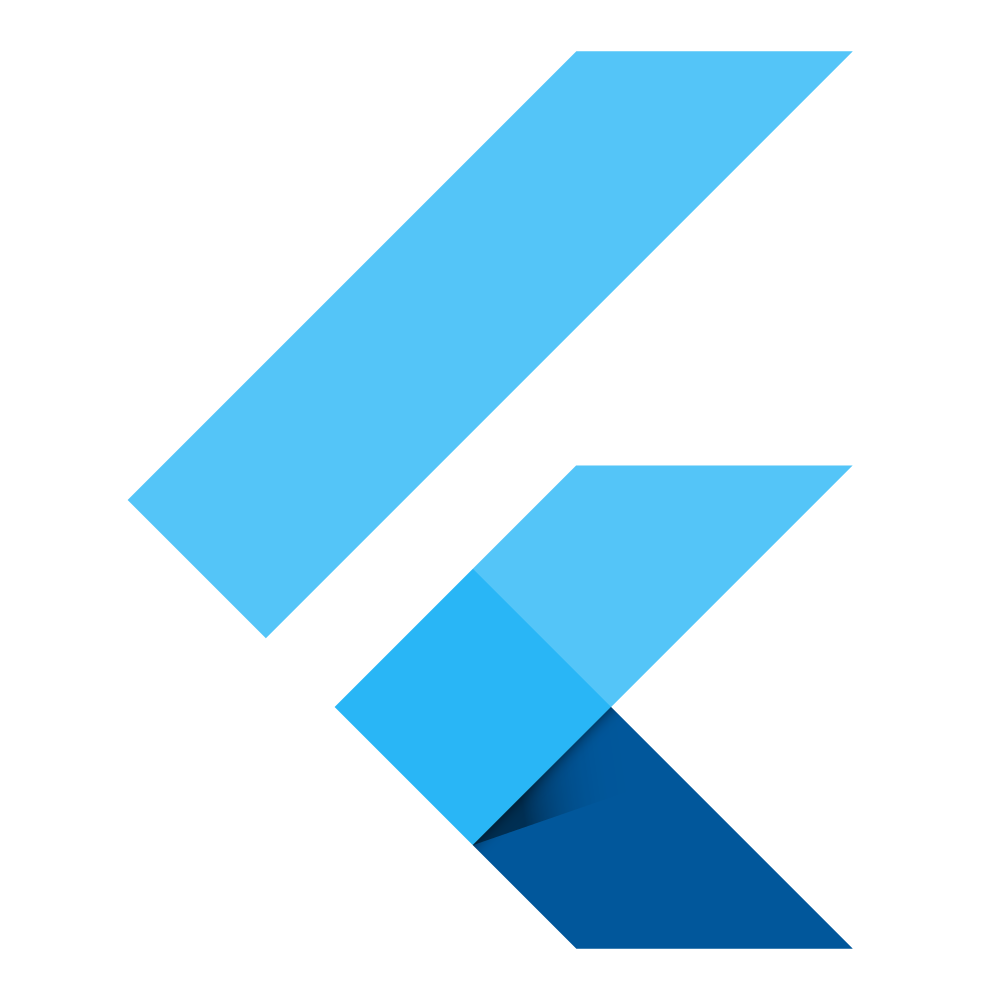
Waarom zou je flutter gebruiken?

Een verdieping in flutter

Ricky van den berg (2124376) **|** ICT-Verdieping **|** 26/04/2019

Inhoudsopgave

[1 Inleiding 2](#_Toc10204034)

[2 Wat is flutter? 3](#_Toc10204035)

[2.1 Wat is een Mobile Framework? 3](#_Toc10204036)

[2.2 Hoe Werkt flutter? 4](#_Toc10204037)

[2.2.1 Dart Shell 4](#_Toc10204038)

[2.2.2 Engine 4](#_Toc10204039)

[2.2.3 Framework 4](#_Toc10204040)

[2.3 Flutter framework voor een developer 5](#_Toc10204041)

[2.3.1 Dart 5](#_Toc10204042)

[2.3.2 Widgets 5](#_Toc10204043)

[2.3.3 Compositie 6](#_Toc10204044)

[2.3.4 State 7](#_Toc10204045)

[3 Wat kan een onervaren flutter developer bereiken in 4 uur? 9](#_Toc10204046)

[3.1 Idee en Design 9](#_Toc10204047)

[3.2 Project creatie 10](#_Toc10204048)

[3.3 Model 10](#_Toc10204049)

[3.4 Logica 10](#_Toc10204050)

[3.5 User Interface (UI) 11](#_Toc10204051)

[4 Conclusie 12](#_Toc10204052)

[Verwijzingen 13](#_Toc10204053)

# 1 Inleiding

Om mijzelf als Informaticus door te ontwikkelen, hebben we vanuit Avans Hogeschool tijd gekregen in het Curriculum om een verdieping te doen in een onderwerp naar keuze. Ik heb het onderwerp flutter en mobile development gekozen. Omdat ik in mijn vrije tijd las over het Flutter framework en het werd beschreven als de oplossing voor het Mobile Development probleem. Namelijk, er zijn veel verschillende Mobile Development Frameworks online te vinden. Deze frameworks komen allemaal met voor en nadelen. Maar welke voor en nadelen zijn belangrijk jou als persoon of voor je bedrijf. Ik heb mezelf in een van deze frameworks verdiept om kennis te vergaren van het framework. Zodat ik deze kennis in de toekomst kan gebruiken bij dergelijke onderzoeken, bedrijfsbeslissingen of persoonlijke projecten.

Het doel van het verslag is om erachter te komen waarom je flutter zou willen gebruiken in projecten? Om dit te onderzoeken ga ik gebruik maken van twee methoden. Ik ga beginnen met een literatuur onderzoek naar flutter om relevante kennis op te doen over het flutter framework. Met deze informatie ga ik later proberen een app te maken in flutter om de kennis de testen in de praktijk. Een aantal randvoorwaarden zijn van toepassing. Namelijk het document is te kort om doorslaggevend te zijn, dit vereist dieper onderzoek en vergelijkingen. Er wordt daarom vanuit gegaan dat je de informatie in dit document vergelijkt met (persoonlijke) informatie met betrekking tot vergelijkbare frameworks en hieruit je eigen conclusie trekt. Er wordt verwachte dat de lezer enige kennis heeft van informatica.

Om de hoofdvraag te beantwoorden is het document opgedeeld in twee hoofdstukken en een persoonlijke conclusie. In hoofdstuk 2 ga ik opzoek naar wat flutter is. Dit beantwoord ik doormiddel van 3 subhoofdstukken. In 2.1 ga ik in op wat een mobile framework is, om een basis begrip te ontwikkelen. Zodat ik in hoofdstuk 2.2 in kan gaan op hoe flutter in elkaar zit. In hoofdstuk 2.3 wordt duidelijk hoe je gebruik maakt van flutter als ontwikkelaar. Deze 3 subhoofdstukken zullen dan een beeld geven van wat flutter is. In hoofdstuk 3 ga ik doormiddel van praktijkonderzoek vastleggen hoe het is om een flutter app te ontwikkelen. Met het contrast van het theorie hoofdstuk en het praktijk hoofdstuk zal ik in hoofdstuk 4 een conclusie trekken.

# 2 Wat is flutter?

Flutter is ontwikkeld door engineers van Google en is sinds 2006 openbaar. Het is een open source en cross platform Software Development Kit. Flutter apps zijn geschreven in de programmeertaal Dart, wat is overgenomen door Google om ervoor te zorgen dat ontwikkelaars zo snel en efficiënt mogelijk te maken om apps te schrijven voor mobiele telefoons, desktop computers en websites. (codemagic, 2019)

## 2.1 Wat is een Mobile Framework?

Als je een aantal jaar geleden een app wilde maken voor IOS en Android dan moest je een aparte app maken voor beide besturingssystemen. Dit vereiste veel werk en kennis van meerdere platformen van developers en bedrijven. Met als gevolg dat er mensen waren die opzoek gingen naar een manier om het werk simpeler te maken. Namelijk een manier om maar 1 project te programmeren wat resulteert in applicaties voor meerdere platformen (cross platform frameworks). In de loop der jaren zijn er een aantal van deze frameworks naar voren gekomen waaronder: React Native, Xamarin, Ionic, Flutter, Cordova en PWA’s. Deze frameworks hebben allemaal een eigen manier van werken. Met als resultaat dat ze allemaal op een andere manier performen (Big, 2019). Ionic apps worden bijvoorbeeld geïnstalleerd met een soort webbrowser zo kunnen ze gebruik maken van de webstack (HTML, CSS, JavaScript). Hierdoor heeft een app die gemaakt is in Ionic extra schrijfruimte nodig omdat het een basis nodig heeft om mee te werken. Dit worden ook wel non native apps genoemd, omdat het niet gebruik maakt van het originele framework van een Android of IOS-telefoon. React Native pakt het anders aan, ze maken nog steeds gebruik van de Webstack voor de opbouw maar dit wordt doormiddel van de “bridge” omgezet in native code. Dit maakt de app vaak sneller dan bijvoorbeeld een app gemaakt in Ionic. Maar door de bridge is er nog wel performance verlies. Wat vooral zichtbaar is bij het opstarten van de app. (Hoober, 2018) Er zijn nog wel een aantal nadelen te bedenken bij deze frameworks. Zo heeft Airbnb React Native achter wegen gelaten omdat ze tegen onverwachte technische en organistische problemen aanliepen tijdens het ontwikkelen van hun React Native app (Peal, 2018).

‌

## 2.2 Hoe Werkt flutter?

Flutter pakt het anders aan dan de andere mobile frameworks. Flutter’s aanpak lijkt meer op dat van een Game Engine. Het maakt namelijk niet gelijk gebruik van de elementen die de native frameworks beschikbaar stellen. (Flutter, n.d.)

### 2.2.1 Dart Shell

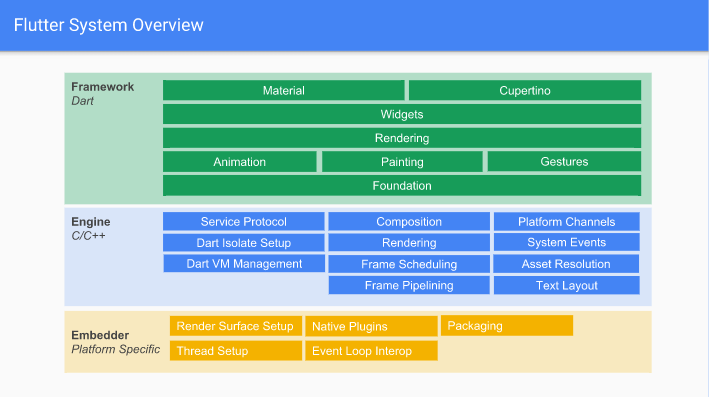
De dart shell is een huls voor het flutter framework. Deze shell bindt de 3 platformen samen. De platformen zijn IOS, Android en Dart Library. De shell onderhoudt de connecties naar de 3 platformen en verzorgt de communicatie met de native API’s van Android en IOS. (Hickson, 2018)

### 2.2.2 Engine

De Flutter Engine ligt boven op de Dart Shell laag en is verantwoordelijk voor het uitvoeren van de taken van het framework. Het is een laag die nauw met de shell samenwerkt om de acties van het flutter framework uit te voeren. Het is de laag die flutter efficiënt laat zijn. (Hickson, 2018)

### 2.2.3 Framework

Het framework is het meest belangrijke gedeelte voor een ontwikkelaar die een app wilt maken met flutter. Het framework dient als een instructiehandboek om gebruik te maken van de engine. De engine heeft alle kracht maar is niet handig om direct mee te werken. Het framework biedt een abstracte laag met instructies om gebruik te maken van de kracht van de flutter engine. Het framework maakt zoals eerder vermeld geen gebruik van de native elementen, maar in plaats daarvan tekent hij zijn eigen elementen. Dit is wat flutter uniek maakt. In React Native is het namelijk zo, als je een button aanmaakt dan wordt dit vertaald en doorgegeven aan het IOS en Android framework als: maak een nieuwe button aan. Hierdoor behoud React Native de stijl van IOS en Android. Bij Flutter worden de instructies voor het tekenen van de button gemaakt in de engine. De engine vertaald deze instructies dan naar instructies die IOS en Android kunnen begrijpen. Hierdoor zal een app gemaakt in flutter er hetzelfde uit zien voor zowel IOS als Android. (Flutter, n.d.)



Figuur 1: Lagen in flutter

## 2.3 Flutter framework voor een developer

Hoe dat het bij flutter op de achtergrond werkt is voor de meeste developers niet relevant om een app te maken in flutter. Wat wel belangrijk is, is hoe het framework is opgebouwd en hoe dat je de elementen en functies eruit gebruikt om een app te maken.

### 2.3.1 Dart

Om apps in flutter te ontwikkelen maak je gebruik van de programmeertaal Dart. Dart is een Object Georiënteerde taal die wordt onderhouden door Google. Dart heeft veel weg van populaire bestaande talen zoals Java, C# en Typescript. De reden dat ze niet voor een van de bestaande talen hebben gekozen is omdat, de dart developers nu invloed hebben op de features van de taal. Een groot aantal features die het flutter team zocht in een programmeertaal was niet compleet of niet goed genoeg in de bestaande talen. Door dart te gebruiken wat net als flutter ook door google developers wordt onderhouden is flutter niet afhankelijk van een taal die een functie niet snel genoeg of niet kan implementeren. (Flutter, n.d.)

### 2.3.2 Widgets

Het meest belangrijke in het flutter framework zijn “Widgets”. Als je de officiële documentatie doorneemt dan komt dit vaak terug. Maar wat is een widget en wat betekend het in flutter?

Een tekst vak, menubalk of container zijn allemaal widgets. Een widget is een element in je app die een beschrijving geeft van een element in het UI. Dit betekend echter niet dat een widget per se zichtbaar moet zijn. In flutter is bijna alles wat je programmeert een Widget. Kortom, Het enige wat een widget niet doet is berekenen. (Flutter, n.d.)

Widgets zijn aanwezig om het de ontwikkelaar makkelijker te maken. Een widget is namelijk herbruikbaar. Neem het voorbeeld uit (figuur 2) waarin te zien is hoe een aantal elementen uit het UI dezelfde eigenschappen hebben. In flutter zijn dit perfecte kandidaten zijn voor een losstaande widgets.

### 2.3.3 Compositie

In flutter werk je met zoveel mogelijk met composities. Dit betekend dat je uit meerdere widgets een compositie maakt, waaruit een nieuwe widget ontstaat. In figuur 2 is te zien hoe een planeet regel bestaat uit een foto widget, meerdere tekst widgets en button widget. Door deze widgets samen te voegen krijg je een planeet artikel widget. Deze widget kan je vervolgens gebruiken om nieuwe planeet artikelen te genereren met nieuwe planeet data. (Flutter, n.d.)



Figuur 2: Voorbeeld app, opgebouwd uit widgets

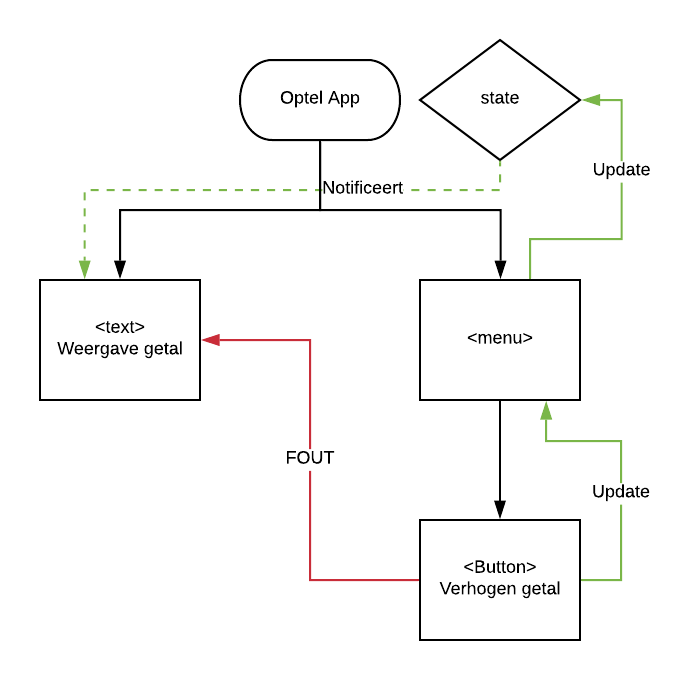
### 2.3.4 State

State of status is vaak een lastig onderdeel van development. In Flutter schrijf je apps doormiddel van “Declarative Programming”. Declarative programming is een principe die je toepast tijdens het werken met state. Wat declarative programming in een notendop zegt is dat: de state verantwoordelijk is voor de UI. Een simpele widget in flutter die bijvoorbeeld een getal bijhoudt en deze doorgeeft aan een widget die tekst weergeeft. Verkrijgt zijn data dus niet direct van het getal widget. De widget laat het model weten dat het nummer is veranderd. Waarop de state alle widgets die dit nummer nodig hebben notificeert om zichzelf te updaten. Zoals te zien in het voorbeeld diagram (figuur 3).

In flutter zijn 3 soorten widgets, namelijk:

* **StatelessWidgets:** Deze widgets hebben geen data die gevolgd moeten worden. De widget is hierom statisch en wordt niet geupdate door veranderingen in de state.
* **StatefulWidgets:** Hebben wel data die veranderd. Deze widgets kunnen zichzelf forceren om opnieuw te laden om de veranderingen in de data correct weer te geven.
* **InheritedWidgets**: Zoals in het 2e paragraaf van state vermeld is, kan het zijn dat een widget van 10 diep soms bij de state moet die 10 hoog zit. Met geneste widgets alleen zal deze data door de Widget Tree takken naar boven getransporteerd moet worden. Waarvoor elke keer kostbare functies uitgevoerd moeten worden. Om dit tegen te gaan komt flutter standaard met een InheritedWidget. Deze widget kan in 1 keer bij een referentie naar de state zonder dat hij door de takken van de Widget Tree naar boven hoeft te klimmen.

In het diagram uit figuur 3 wordt een simpele “Widget Tree” uitgebeeld. Een Widget Tree is een soort familieboom voor een flutter app. Hierin hangen de widgets die in elkaar genest zijn onder elkaar. Het managen van de state is makkelijker met een Widget Tree van je app op papier. In grotere apps zullen deze trees namelijk erg gecompliceerd worden. Zo kan het voorkomen dat de state van 10 widgets diep helemaal terug naar boven moet reizen om vervolgens in een andere tak 5 diep moet gaan om zijn doel te bereiken. Dit moet voorkomen worden, want het zorgt voor vertraging in je app. Om dit probleem te verhelpen zijn er een aantal handige packages beschikbaar om het gemakkelijker te maken om te werken met state. (Flutter, n.d.)



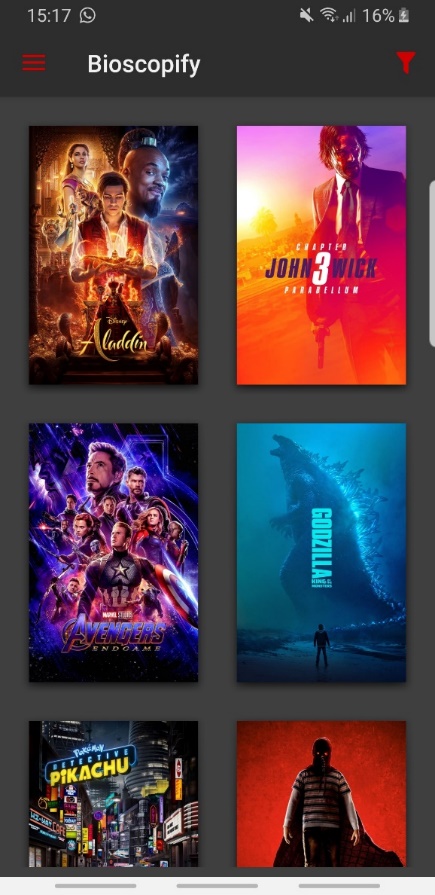
Figuur 3: Simpele widget tree in een flutter optel app.

# 3 Wat kan een onervaren flutter developer bereiken in 4 uur?

In dit hoofdstuk ga ik persoonlijk aan de slag met flutter om te ervaren hoe het is om er mee te werken. Mijn opvattingen in dit hoofdstuk zijn dus geen feiten, maar meningen. Het doel is om de denkwijze van een onervaren flutter developer weer te geven. In de hoop dat deze ervaring aansluit bij de resultaten van het eerdere onderzoek.

## 3.1 Idee en Design

Om het flutter framework goed te kunnen beoordelen ga ik een app maken die ik al eens eerder heb gemaakt als onderdeel van het curriculum Java mobile development. De app is een overzicht van films die momenteel in de bioscoop draaien. De app zal ongeveer dezelfde functies en UI-componenten hebben als te zien in figuur 4.



Figuur 4: Overzicht films en detailscherm film in Java

## 3.2 Project creatie

Creëren van een flutter applicatie is erg simpel vanwege de uitstekende documentatie op de site van flutter. Het aanmaken van een project is zo simpel als. “Flutter new <naam>” in een terminal invoeren en de gecreëerde volder openen in je editor naar keuze.

## 3.3 Model

Voor de app heb ik gebruik gemaakt van een API. Deze API biedt informatie aan over de films in de vorm van een lijst van films. Elk item uit de list moet omgezet worden om er gebruik van te maken. Hiervoor maak ik gebruik van een simpele model class genaamd Movie. Dart heeft een aantal handige notaties om dit snel te doen. Zoals te zien in figuur 5.



Figuur 5: Model class Movie

## 3.4 Logica

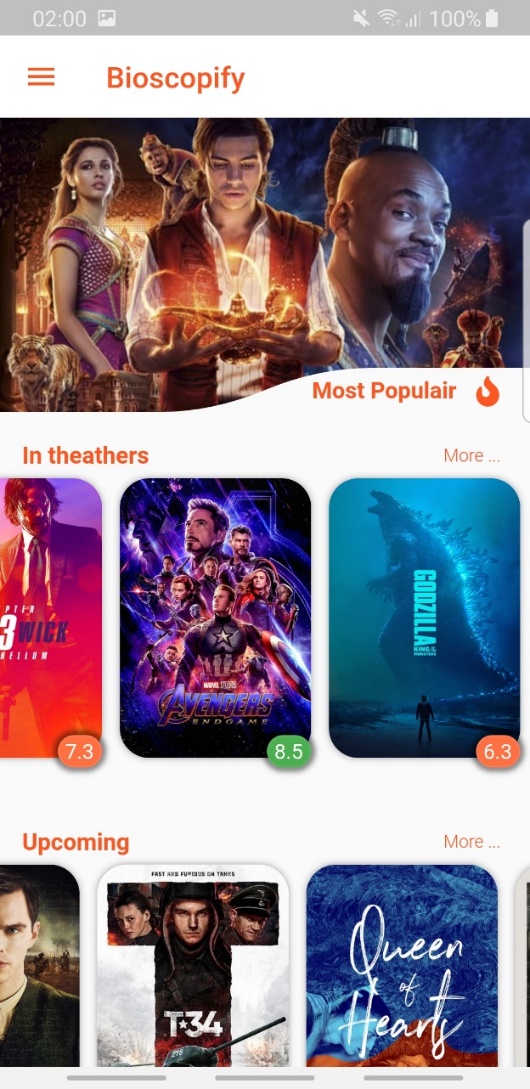
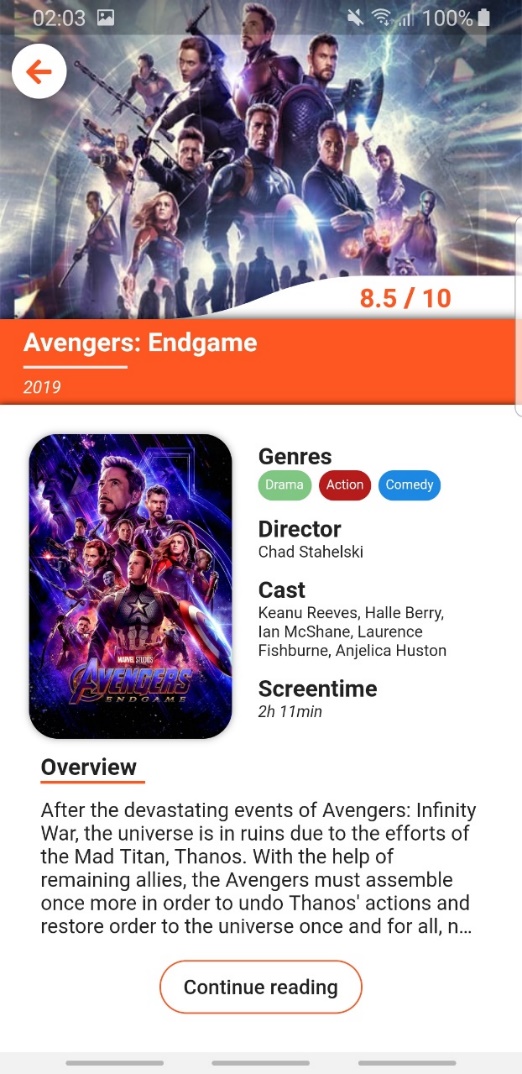
Ik ben begonnen met het maken van de logica. Om requests uit te voeren op de API moest ik een service maken die data uit de API op kan halen. Flutter heeft ingebouwde libraries om hierbij te helpen, hierdoor heb je maar een paar regels nodig. Zoals te zien in figuur 6.

Figuur 6: APIservice in Dart

## 3.5 User Interface (UI)

Werken met het UI gedeelte in flutter ervaarde ik als lastig in het begin. Het is een hele nieuwe denkwijze. Waar je in de meeste applicaties verschillende talen hebt voor UI en logica. Is het bij flutter allemaal Dart. Alles is een Widget en bij het maken van diep geneste en lastige UI’s resulteerde dit bij mij in hele lange stukken UI code. De grote stukken UI-code konden wel opgedeeld worden in kleinere stukken echter door mijn onervarenheid werd dit pas duidelijk na het maken van het geheel. Een manier om dit in de toekomst te voorkomen is om van tevoren je UI-design al op te delen in widgets zodat je de stukken code kan opdelen in kleinere delen voordat het uit de hand loopt.

Toch vond ik het maken van de UI doormiddel van code steeds fijner en makkelijker werken en heb ik uiteindelijk een best leuk resultaat weten te boeken (figuur 7).



Figuur 7: Overzicht films en detailscherm film in Flutter

# 4 Conclusie

Door de nieuwe aanpak van flutter op het gebied van Mobile Development. Was het werken met flutter voor mij onwennig, maar niet onprettig. Het forceert je om anders te denken dan dat je misschien gewend bent. Echter toen ik eenmaal opgang kwam ging het ontwikkelen mij erg makkelijk af. Dart was geen hindernis desondanks, het een compleet nieuwe programmeertaal voor mij persoonlijk was. Helaas was het onderzoek te kort om echt te spreken van vergelijkingen met andere frameworks en moet ik de hoofdvraag persoonlijk beantwoorden. Waarom zou je voor flutter kiezen? Omdat, het makkelijk te leren is, de documentatie is uitstekend, de opzet is simpel en snel, het wordt ondersteund door Google en het laat mij veel meer werk met minder code.

# Verwijzingen

Big, S. (2019, 4 7). *How Has Mobile Application Development Changed Over the Years?* Opgehaald van myventurepad: https://myventurepad.com/how-has-mobile-application-development-changed-over-the-years/

codemagic. ( 2019, 1 18). *what is flutter benefits and blimitations*. Opgehaald van codemagic: https://blog.codemagic.io/what-is-flutter-benefits-and-limitations/

Flutter. (sd). *inside flutter*. Opgehaald van flutter: https://flutter.dev/docs/resources/inside-flutter

Flutter. (sd). *State management*. Opgehaald van flutter: https://flutter.dev/docs/development/data-and-backend/state-mgmt

Flutter. (sd). *technical overview*. Opgehaald van flutter: https://flutter.dev/docs/resources/technical-overview

Flutter. (sd). *Why did Flutter choose to use Dart?* Opgehaald van flutter: https://flutter.dev/docs/resources/faq#why-did-flutter-choose-to-use-dart

Hickson, I. (2018). *The Engine architecture*. Opgehaald van github: https://github.com/flutter/flutter/wiki/The-Engine-architecture

Hoober, S. (2018, 8 6). *Mobile Apps: Native, Hybrid, and WebViews*. Opgehaald van uxmatters: https://www.uxmatters.com/mt/archives/2018/08/mobile-apps-native-hybrid-and-webviews.php

Peal, G. (2018, Juli 19). *sunsetting react native*. Opgehaald van medium: https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a