Kennismaken met Flutter

Nu in coronatijden mijn ZZP-agenda maar heel mager is gevuld, heb ik besloten om van de nood een deugd te maken. Ik ga twee nieuwe frameworks onderzoeken waarvoor het mij tot op dit moment aan de tijd ontbrak om me er meer in te verdiepen. Dat zijn:

- ✓ Svelte, op https://velte.dev - hierover schreef ik een eerder artikel.
- ✓ Flutter, op https://flutter.dev

In dit artikel kijk ik naar Flutter. Afgelopen week heb ik kennisgemaakt met Flutter, de documentatie gelezen, online tutorial(s) gevolgd en verder rondgesnuffeld naar Flutter-informatie op de voor de hand liggende plekken als Stack Overflow, YouTube en Dev.to.

Disclaimer - ik ben frontend developer. Ik ken frameworks als Angular, Vue en React op mijn duimpje. Ik heb ook gewerkt met PhoneGap/Cordova, Ionic en React Native. Hiermee heb ik applicaties gemaakt. Mijn beleving en beschrijving van Flutter - waarmee je ook mobiele apps maakt - komt dan ook vanuit dit perspectief. Ik maak regelmatig de vergelijking me zaken die ik ken. Dat hoeft voor jou natuurlijk niet zo te zijn! Toch probeer ik het zo eenvoudig mogelijk te houden en een voor iedereen duidelijke beginners/intro te schrijven over Flutter.

In dit artikel beantwoord ik de volgende vragen:

1. Wat is Flutter?
2. Waar wordt het voor gebruikt?
3. Hoe gaat lokale Installatie en Ontwikkeling?
4. Hoe maak ik een eerste applicatie?
5. Hoe maak ik een niet-triviale *proof-of-concept* applicatie: een app die op basis van een zoekvraag van een gebruiker communiceert met een backend en de resultaten in de UI toont.

Ik ben geen beginner in app-development, maar uiteraard wel in Flutter. Daarom heb ik ongetwijfeld zaken gemist of heb ik dingen omslachtig gedaan (omdat ik die nu eenmaal ken uit andere frameworks) die in Flutter veel eenvoudiger kunnen. Aarzel in dat geval niet om het mij te laten weten!

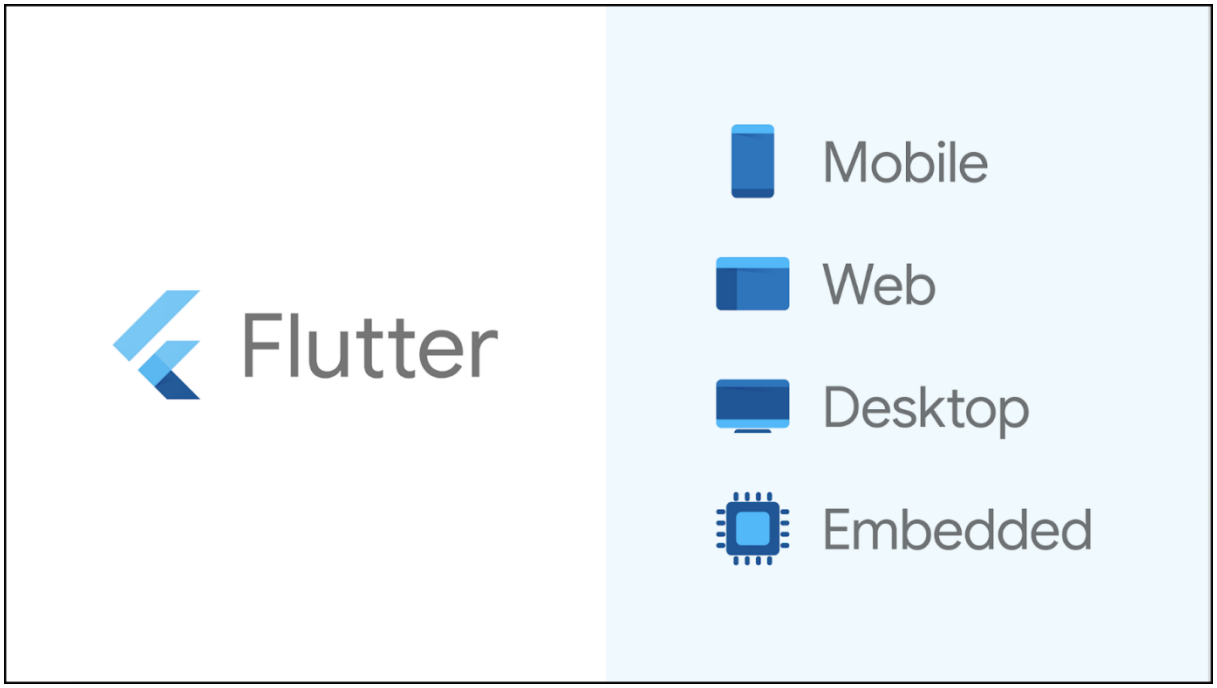
1. Wat is Flutter?

Flutter is een *Mobile UI Framework* van Google, dat wordt gebruikt voor het maken van apps voor iOS en Android. Dit zijn native apps. Het zijn dus geen apps die draaien in een webview van het betreffende platform (zoals Ionic en PhoneGap). De homepage van Flutter is te vinden op https://flutter.dev/.

Het kenmerk van Flutter is dat er een *single code base* is. Apps schrijf je in de eveneens door Google ontwikkelde programmeertaal *Dart* (https://dart.dev/). Jouw Dart-code wordt gebruikt om apps te compileren voor meerdere platforms.

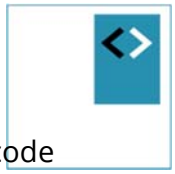
Met Flutter kun je ook apps maken voor embedded systems zoals Google Assistant-devices of Nest-thermostaten. Flutter heeft hiermee dezelfde doelstellingen als bijvoorbeeld Xamarin (voor C#-programmeurs) en React Native (voor React-programmeurs).

Met Flutter ontwikkelde apps kun je publiceren in de Apple App Store of op Google Play. Natuurlijk kun je ze ook intern binnen je bedrijf uitrollen.



Dart

Je schrijft je Flutter-apps in Dart. Dit betekent dat je behalve het Flutter UI-systeem ook een nieuwe programmeertaal moet leren. Je leert dus eigenlijk twee dingen tegelijk. Dart heeft onder meer de volgende kenmerken:

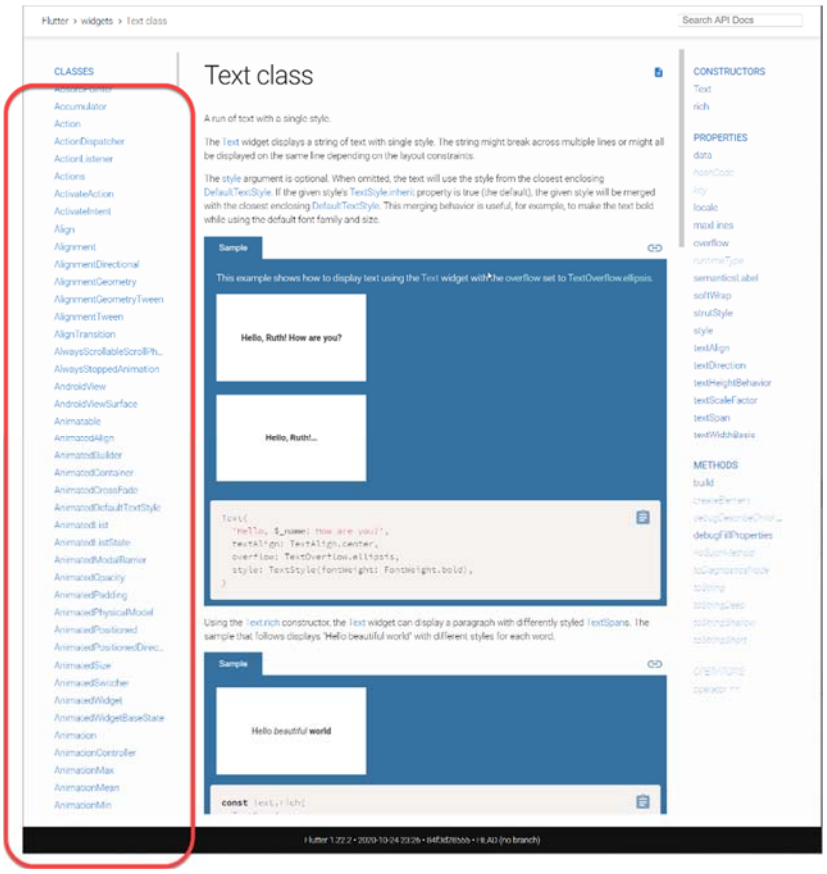


- ✓ Dart-code compileert naar native Android-code (Java of Kotlin) en native iOS-code (Objective-C of Swift). Er is ook een experimentele module die compileert naar JavaScript-code, zodat je je app ook op het web kunt gebruiken. Deze moet je in Flutter apart inschakelen.
- ✓ Dart lijkt op andere programmeertalen zoals Java, JavaScript of TypeScript. Je kunt daarom als programmeur snel overweg met Dart. Er zijn concepten als classes, componenten/widgets, loops, variabelen, functies enzovoort. Dart leent zowel concepten uit de OO-talen als uit de Functional Programming-wereld.
- ✓ Dart is een *staticly typed language*, anders dan JavaScript. Het betekent dat je variabelen altijd als een bepaald type moet declareren. Voor de JavaScript-programmeur zijn er een aantal nieuwe keywords, die echter wel overeenkomsten hebben met JavaScript.
- ✓ Ik was compleet nieuw in Dart. Mijn ervaring is dat je het snel oppikt als je al programmeerervaring hebt. Er zijn wel subtiele verschillen waar je regelmatig over struikelt.

UI-system

De kern van Flutter zijn de werkelijk talloze (honderden!) verschillende user interface componenten. In Flutter-taal heten dit *widgets*. Zoals je in moderne webframeworks (Angular, Vue, React, Svelte) denkt in componenten, denk je in Flutter in widgets. Elke applicatie is een boomstructuur van widgets. Denk aan containers, knoppen, dialoogvensters, icons en verder alles wat je op het scherm ziet.

Dit wordt erg ver doorgevoerd. Als je witruimte binnen- of rondom een element wilt instellen, doe je dit via een `Padding()`-widget of een `Margin()`-widget. Daarnaast heeft elke widget – net als in HTML – talloze eigenschappen (*attributes*) waarmee je de inhoud verder instelt.



Mobile web

Flutter heeft zich erg laten inspireren door het mobiele web. Er is geleerd van de lessen van CSS-lay-out, zoals CSS-flexbox en Grid. Flutter bootst deze flexibele flow van elementen op een pagina na met eigen syntaxis. Als je dit in CSS beheerst, zul je relatief snel de Flutter-manier van lay-out begrijpen.

Als je dit nog niet kent, zul je wel een aantal keren je hoofd breken over de vraag "waarom het zo moeilijk is om een knop onderin de pagina te krijgen" of anderszins een element op de goede plek te positioneren. Je moet een fluïde geest hebben voor het realiseren van fluïde lay-outs.

Flutter gebruikt Material Design (eveneens ontwikkeld door Google) als uitgangspunt.

Documentatie

De kwaliteit van een framework valt of staat vaak met de kwaliteit van de documentatie. Een framework kan nog zo goed zijn, als je niet weet hoe alle mogelijkheden benut worden, heb je er niks aan. Wat dat betreft heeft team Flutter zijn werk goed gedaan en

gezorgd voor prima documentatie. Zie ook de voorgaande afbeelding. Daar kan de rest van Google eigenlijk nog een voorbeeld aan nemen.

Bij elke widget zijn codevoorbeelden opgenomen, vaak in een interactieve app, zodat je er zelf mee kunt oefenen. De talloze properties en methods worden beschreven; er zijn vaak video's aanwezig (*'widget of the week'*) waarin het gebruik van een widget wordt uitgelegd en er zijn voldoende verwijzingen naar andere bronnen aanwezig.

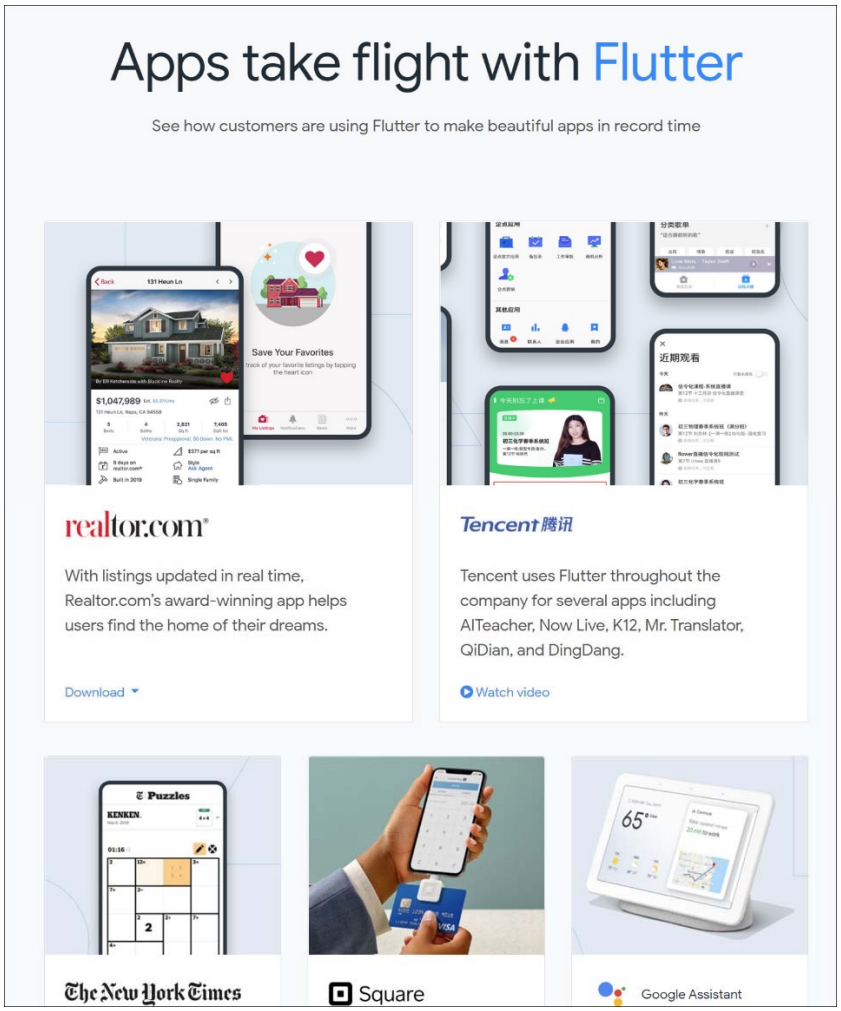
Het lastige is vooral de *hoeveelheid* documentatie. Als je niet wéét dat een bepaalde taak door een aanvullende widget wordt uitgevoerd, kun je lang zoeken. Het is eerder te veel dan te weinig. Maar als je zelf een beetje kunt filteren, pik je de elementaire onderdelen er gelukkig snel uit.

Mijn oordeel over de documentatie: ++

2. Waar wordt Flutter voor gebruikt?

Flutter wordt op dit moment vooral gebruikt voor crossplatform mobiele applicaties. Inmiddels gebruiken honderden grote en kleine bedrijven Flutter als tool. Dit zijn onder meer Google zelf, Alibaba, Tencent, eBay, BMW en de New York Times. Geen kleine jongens. Ja, er *zijn* ook implementaties voor embedded devices, maar deze zijn vooralsnog vooral van Google zelf. Ik heb geen voorbeelden van andere fabrikanten gezien. Wellicht dat we op termijn ook appjes kunnen maken voor Raspberry Pi of Arduino-boards.

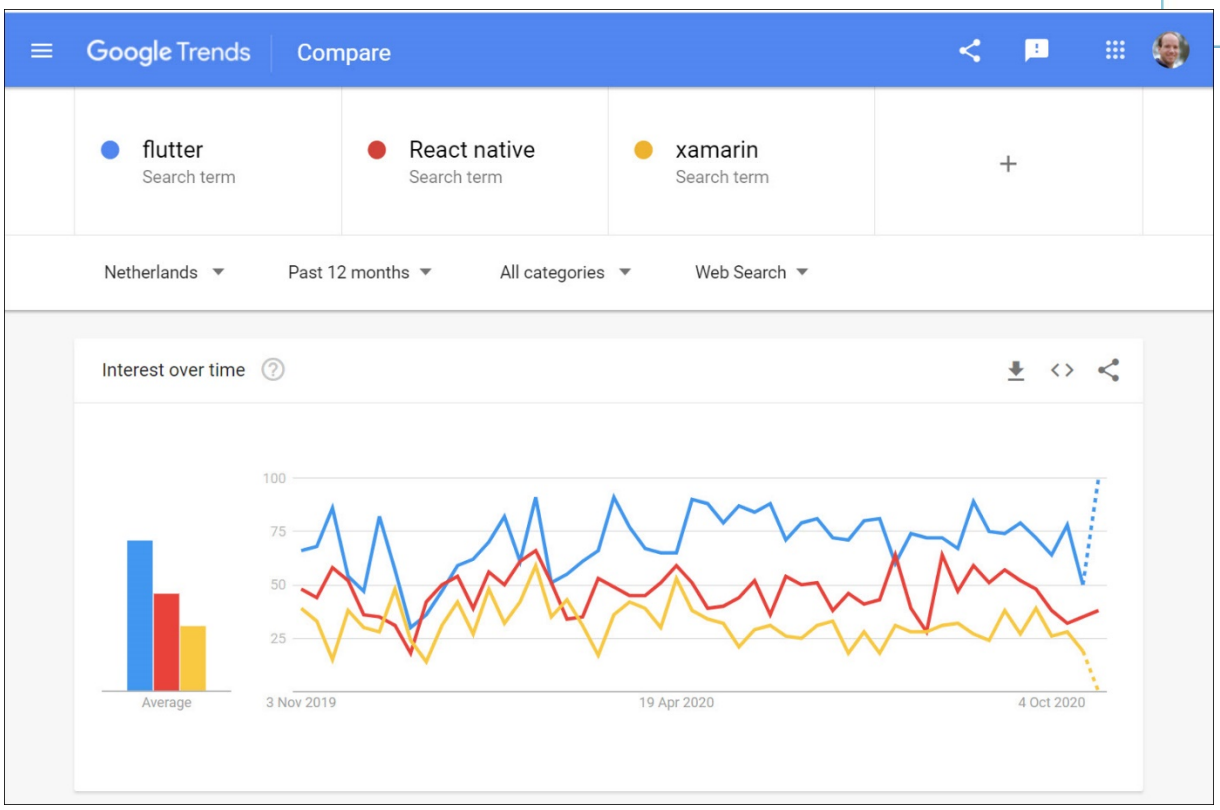
Een showcase-pagina van Flutter-apps is te vinden op https://flutter.dev/showcase.



Flutter in Nederland en België

Persoonlijk ken ik geen nog bedrijven in Nederland of België die hun app(s) met Flutter hebben ontwikkeld, al hoor ik van (outsourcing- en detacherings-) relaties dat er wel degelijk vraag is naar Flutter-kennis. Het is een interessante techniek die zich al afdoende bewezen heeft.

Om te zien of er interesse is, kun je bijvoorbeeld kijken naar Google Trends. Een eenvoudige vergelijking tussen Flutter, React Native en Xamarin laat zien dat de interesse voor Flutter groter is dan voor de andere twee. De trend wordt ook door Google als *rising* aangeduid (al durf ik er mijn hand niet voor in het vuur te steken dat dit niet komt omdat het een Google-technologie is).



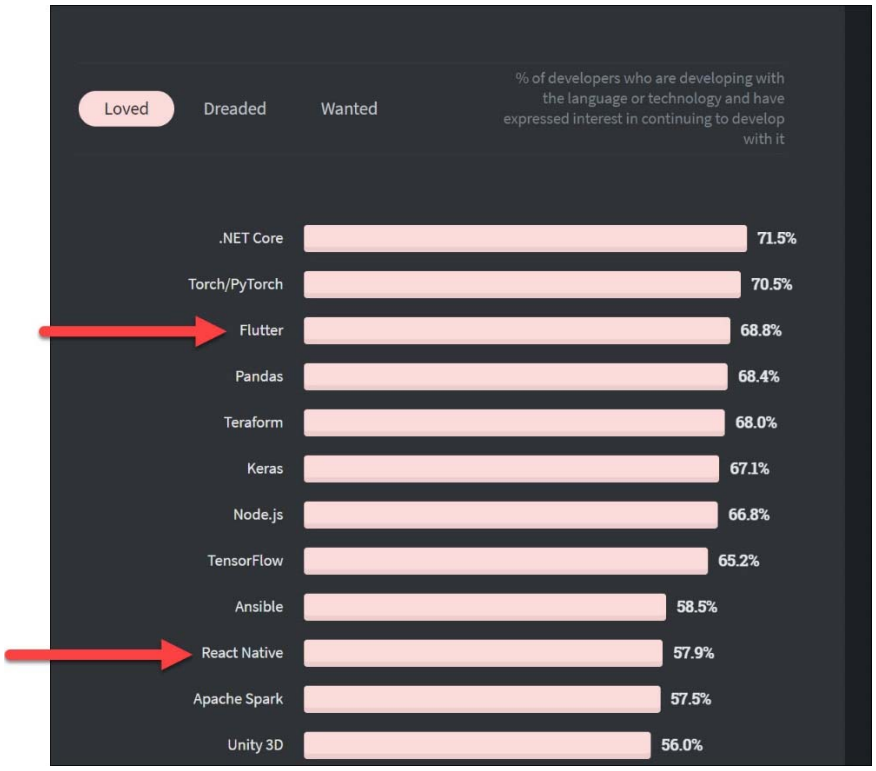
Bekijk zelf de huidige trend op https://trends.google.com/trends/explore?geo=NL&q=flutter,React%20native,xamarin.

Brede blik

Voor een wat bredere blik op de vraag naar Flutter-kennis en -applicaties kun je bijvoorbeeld kijken naar de jaarlijkse *Stack Overflow Developer Survey*. Die van 2020 is te vinden op https://insights.stackoverflow.com/survey/2020.

Je ziet dan dat Flutter in de categorie *Other frameworks, Libraries en Tools* nog juist in de top-10 staat. Achter React Native, maar ver voor andere frameworks als Xamarin en Cordova.

In de categorie *Most Loved* staat Flutter op plek 3, terwijl React Native op de tiende plaats is beland. Xamarin en Cordova bungelen in dit rijtje onderaan (maar scoren zoals verwacht dan juist weer heel hoog in de categorie *Most Dreaded*). Dat geeft mijns inziens wel aan dat Flutter, hoewel het nog niet een supergrote technologie is, wel een techniek is die redelijk populair is onder developers en waarmee bovendien graag gewerkt wordt.



3. Hoe gaat lokale ontwikkeling en installatie?

Veel webframeworks kun je uitproberen zonder het downloaden van een tool of lokale installatie. Op https://stackblitz.com of https://codesandbox.io kun je direct online aan de slag met React, Angular, Vue, Nuxt, TypeScript en wat al niet. Dat is niet het geval bij Flutter.

Je moet hiervoor lokaal de juiste SDK's downloaden voor het platform van jouw keuze. Dus: de Android SDK voor Android-apps of Xcode plus talloze aanvullende bibliotheken voor MacOS applicaties. Of beiden. Daarnaast moet je een editor installeren met de nodige Flutter- en Dart-plug-ins. Dat kost je vele gigabytes. En veel tijd.

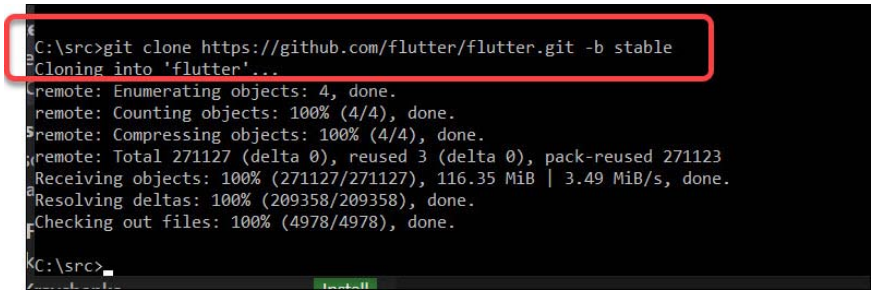
Het is allemaal niet superlastig en is gelukkig een eenmalig klusje, maar reken in het gunstigste geval wel op anderhalf uur of langer voordat je aan de slag kunt met een simpele Hello World-applicatie.

Je wilt je app natuurlijk ook testen op een echte telefoon. Android-telefoons kun je meestal gewoon koppelen aan je computer, voor iPhones is de werkwijze een stuk omslachtiger. En als je de pech hebt dat je telefoon niet herkend wordt door je pc, ben je

zomaar weer uren verder met het zoeken naar een oplossing en installeren van de juiste drivers.

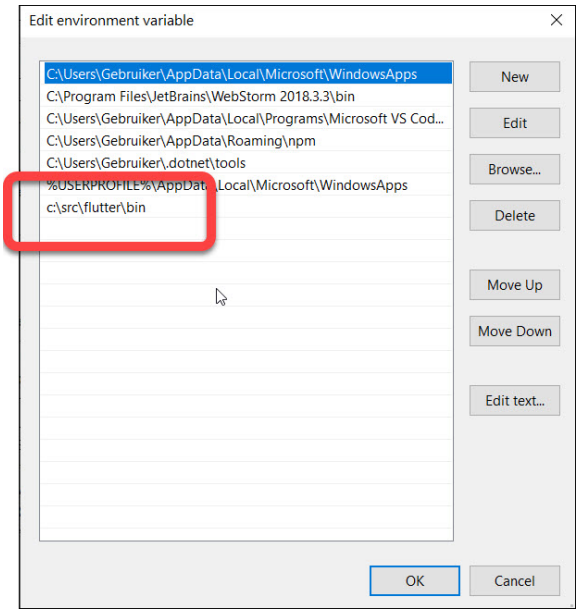
Wat heb je nodig?

Om te kunnen beginnen met Flutter moet je een aantal stappen doorlopen. Op de Flutter-website wordt dit onder *Getting Started* uitvoerig beschreven. Ik heb deze procedure gevolgd voor Windows en het liep relatief goed (op een connectieprobleem met mijn Samsung S8 na. Maar daar kon Flutter niks aan doen).

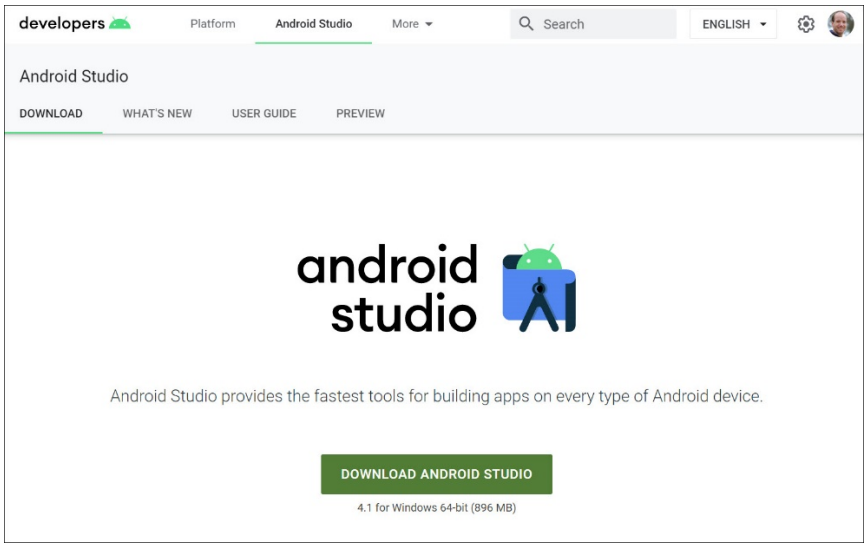


Je moet hebben:

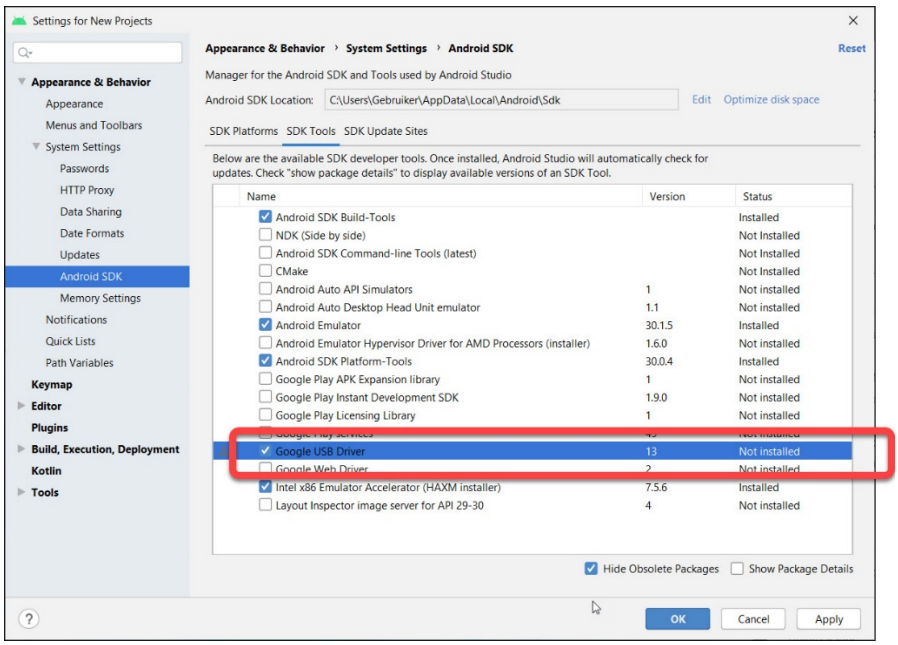
- ✓ **Git** als versiebeheertool. Zelfs als je zelf Git niet gebruikt (je hebt bijvoorbeeld SVN of TFS), heeft Flutter het nodig.
- ✓ De **Flutter SDK**. Dit is een repo die je kloont, of een zip-bestand dat je kunt downloaden en uitpakken. De standaardlocatie is C:\src\flutter. Deze heb ik aangehouden.
- ✓ Je Windows- of Mac **Path** instellen zodat flutter bekend is in de Terminal-omgeving of in Powershell. Hiervoor moet je in Windows de omgevingsvariabelen aanpassen (zie afbeelding).



- ✓ **Android Studio** en de **Android SDK's.** Dit kost veel schijfruimte. De procedure is nu gelukkig geautomatiseerd (een paar jaar geleden, toen ik hetzelfde deed voor PhoneGap, moest dit allemaal nog met de hand), maar het kost veel tijd. Android Studio is gratis en is te vinden op https://developer.android.com/studio.
- ✓ Na installatie van Android Studio (dit installeert de Android SDK, de SDK command-line tools, de SDK Build-tools en de IDE zelf) moet je nog de juiste Flutter- en Dart-plug-ins installeren.
- ✓ Je kunt ook **Visual Studio Code** gebruiken. Hiervoor zijn Flutter- en Dart-extensions beschikbaar. Ook dan heb je echter Android Studio nodig, of op zijn minst de hiervoor genoemde SDK's. Die kun je eventueel met de opdracht flutter doctor installeren. Dit pad heb ik niet gevolgd.



- ✓ Als alle software is geïnstalleerd wil je een **android device** kunnen gebruiken. Hiervoor moeten de Developer options (*Ontwikkelaarstools*) en USB debugging op je apparaat zijn ingeschakeld. Hoe dit werkt, verschilt per merk (Samsung, Huawei, Sony enzovoort).
- ✓ Voor Windows moet je vanuit Android Studio in dat geval nog de **Google USB-driver** installeren als generieke driver. Voor Mac is dat niet nodig.
- ✓ Mocht je niet op een fysiek apparaat willen of kunnen testen, dan kun je altijd een Android Emulator aanmaken via de AVD, de *Android Virtual Device Manager*.



Android Emulator

De Android Emulator is gelukkig een stuk sneller dan vroeger. Het is zowaar een valide oplossing geworden om je apps te testen, zo heb ik gemerkt. Een paar jaar geleden was het snéller om naar de winkel te lopen en een nieuwe Android-telefoon te kopen dan om de emulator op te starten en hier de app in te draaien. Dat is nu een stuk verbeterd.

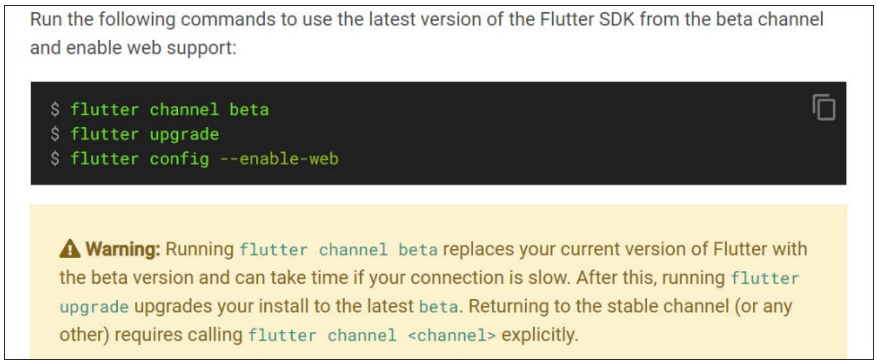
Alle moderne pc's ondersteunen *VM acceleration* en grafische kaarten kennen *hardware acceleration*, waardoor de apps ook in de emulator redelijk responsief aanvoelen.

Flutter op het web

Je kunt Flutter ook gebruiken voor het maken van webapplicaties. Hiervoor moet je *eerst* alle voorgaande stappen doorlopen.

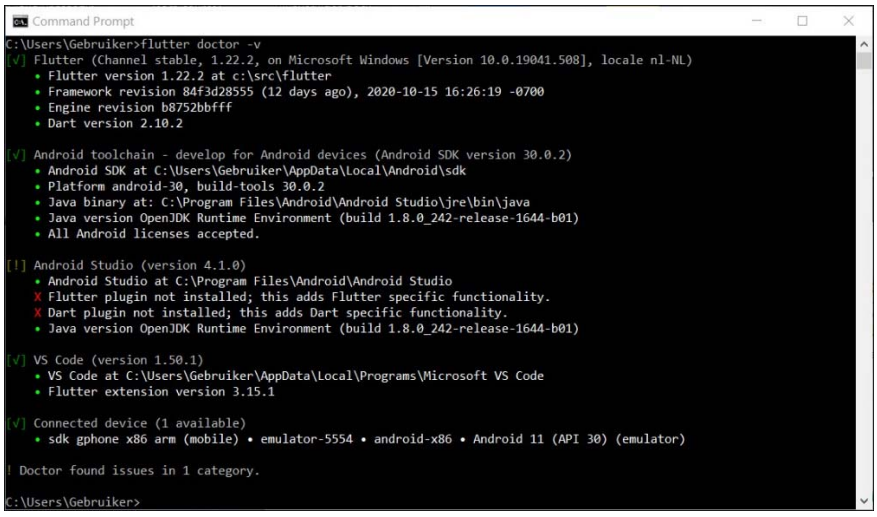
Daarna kun je flutter upgraden naar het beta channel en met de flag `--enable-web` het maken van webprojecten inschakelen. Zoals gezegd, dit is allemaal nogal experimenteel op dit moment. Het is omgeven met waarschuwingen en nog in de betafase. Ik heb het niet geprobeerd.

In de toekomst verwacht ik dat dit soepeler is opgelost en geïntegreerd in de totale Flutter-workflow. Je leest er voor nu eventueel meer over op https://flutter.dev/docs/get-started/web.



Flutter doctor

De command line tool voor het werken met Flutter heet – niet verrassend – `flutter`. De totale installatie kun je checken met de opdracht `flutter doctor`. Met de flag `-v` (verbose) toon je complete uitvoer.



De tool lijkt evenwel niet altijd goed te werken, want ik heb bijvoorbeeld wel degelijk de Flutter plug-in en Dart plug-in voor Android Studio geïnstalleerd. Maar flutter doctor herkent ze niet, zoals in bovenstaande afbeelding is te zien. In ieder geval is het een handige aanvullende analysetool om mogelijke problemen op te lossen.

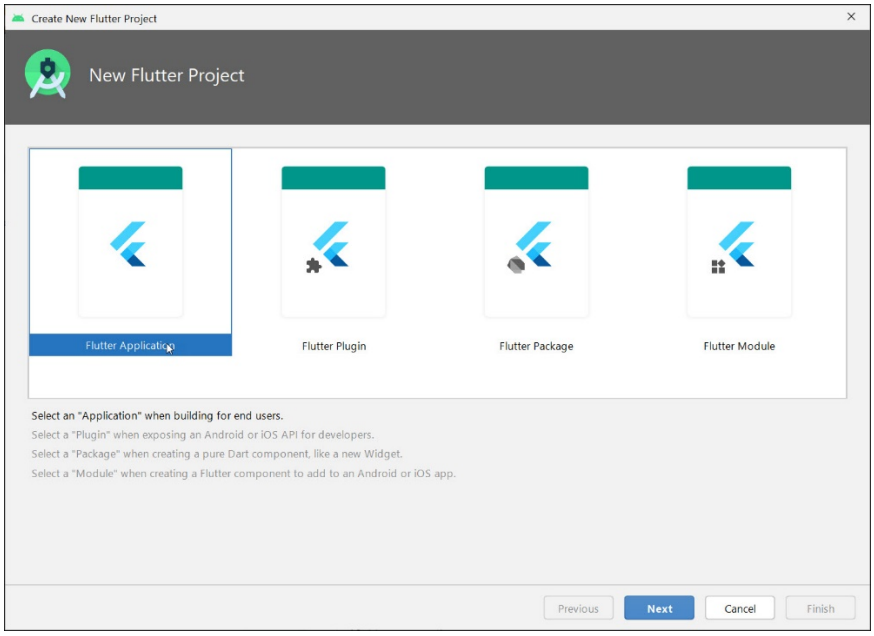
4. Hoe maak ik een eerste applicatie?

Als de installatieperikelen achter de rug zijn kun je een eerste applicatie maken. Hier beschrijf ik het pad voor de 'officiële' editor, Android Studio. Maar ook in Visual Studio Code met de nodige extensions heb ik succesvol Flutter-apps geschreven.

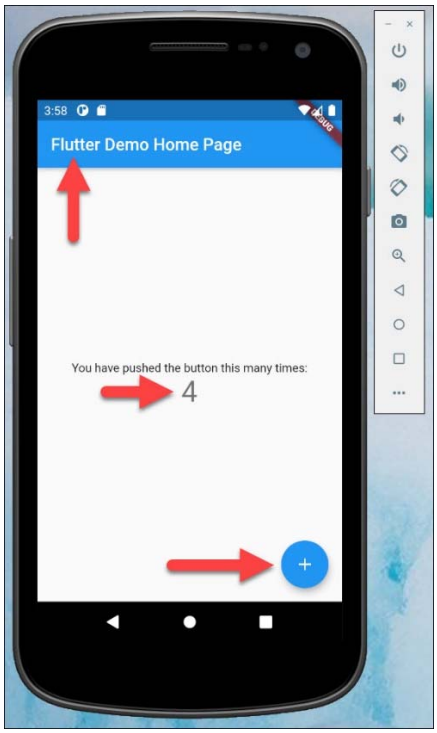
Android Studio biedt wat mij betreft echter net wat meer handige panels om inzicht te krijgen in de structuur van je applicatie zodra deze wat complexer wordt. Ook het

beheren van AVD's en plug-ins is wat soepeler. Maar dat is een kwestie van persoonlijke voorkeur.

Visual Studio Code is een stuk kleiner, start sneller op en als je dit al gewend bent voor webontwikkeling, dan zou ik je zeker aanbevelen hiermee ook Flutter-apps te maken.



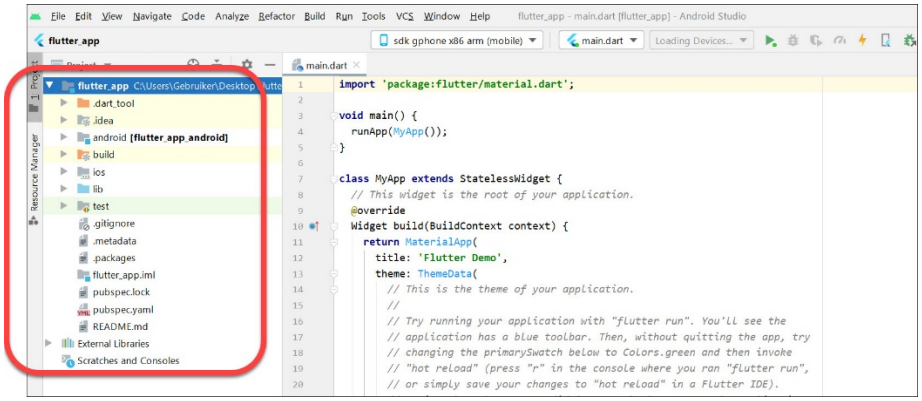
In Android Studio kies je na het installeren van de plug-ins voor het maken van een `New Flutter Project`. Als je de standaardinstellingen overneemt, krijg je een applicatie met een homepage met daarop een aantal Widgets. Dit zijn onder meer een `AppBar`, een counter-variabele, tekst, en een knop (`FloatingActionButton`) om de teller op te hogen.



In de voorbeeldcode is ruim voldoende commentaar opgenomen (lees dit!) om te zien hoe de applicatie werkt en wat je kunt aanpassen om deze demo-app een ander uiterlijk te geven.

Structuur van de applicatie

Omdat de applicatie naar meerdere platformen kan worden gepubliceerd, staan er ook meerdere mappen in de boomstructuur. Je ziet ze in de afbeelding.



- ✓ Op root niveau staan directories als `/android`, `/ios` en `/.dart-tool`. Dit zijn folders voor de diverse platforms en met aanvullende tools. Meestal zul je hier nooit handmatig in aan het werk zijn, hooguit als je voor publicatie een aantal permissies wilt instellen, een icon wilt plaatsen of meer.

- ✓ De meeste tijd zul je doorbrengen in de folder `/lib`. Hierin staan je werkbestanden. Het startbestand is `main.dart`. Vanuit de folder `/lib` worden bij een build de bestanden gecompileerd en gedistribueerd naar de verschillende platform-directories.
- ✓ In `/lib` kun je subfolders maken om je bestanden te groeperen en organiseren, al heten dat hier om een of andere duistere reden *packages*. Kies dus `New`, `Package` als je een submap wilt maken.
- ✓ Op het rootnivau staan configuratiebestanden als `.pubspec.yaml`, `.gitignore` en meer.

Voor webdevelopers: `.pubspec.yaml` is te vergelijken met `package.json`.

Helaas is er echter niet een handige tool zoals NPM om aanvullende packages te installeren, dat zul je zelf met de hand moeten doen. Hierbij moet je ook het aantal spaties inspringing respecteren, anders is de structuur van het bestand niet correct. Beetje ouderwets en omslachtig, maar het werkt.

Een overzicht van beschikbare packages voor Flutter en Dart is te vinden op https://pub.dev. Deze site kun je volgens mij het beste vergelijken met https://www.npmjs.com. waar je NPM-packages kunt vinden. Maar het is een compleet nieuw, afzonderlijk package- en module- ecosysteem, naast NPM of NuGet (als je uit de .NET-wereld komt).

Widgets in Flutter

De complete applicatie is in Flutter gebouwd rond het concept van *Widgets*. Een widget kun je vergelijken met een component in Angular, Vue of React. Op de achtergrond is een widget een class en daarmee gewoon een functie. Je kunt een widget aanroepen of *invoken*.

Flutter kent **twee verschillende typen widgets**:

- ✓ *StatelessWidget* Dit is een widget waarvan de state (oftewel: data, variabelen) in de levensduur van de widget niet veranderen. Je kunt wel data hebben in een StatelessWidget, maar deze blijft dus steeds hetzelfde. De documentatie beschrijft een StatelessWidget als volgt: *"A stateless widget is a widget that describes*

part of the user interface by building a constellation of other widgets that describe the user interface more concretely." De voornaamste taak van een StatelessWidget is dan ook het aanroepen van andere (meer dynamische) widgets.

Flutter Basisstructure stateless widget

main.dart

```dart
class Contact extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      // Overige widgets die de Contactpagina beschrijven
    );
  }
}
```

- ✓ *StatefulWidget* Hiervan kan de data gedurende de levensduur veranderen. Een StatefulWidget bestaat bovendien uit twee delen:
  - o Een class waarmee de stateful widget wordt beschreven en waarin de functie `createState()` wordt aangeroepen om de data voor deze klasse te initialiseren.
  - o Een state-class waarmee de data/variabelen van de widget worden beschreven. Deze wordt altijd voorafgegaan door een underscore. In Dart geeft een underscore voor een klasse- of variabelenaam aan dat het betreffende object private is (zoals in `_userName`).
- ✓ Een Statefulwidget heeft dus bijvoorbeeld de hoofdklasse `AboutUs()`. Deze bevat een aanroep naar `createState()` waarin de bijbehorende state-klasse `_AboutUs()` wordt aangeroepen. Vanuit Android studio kun je met de shortcut `stful <Tab>` snel deze twee klassen genereren.

Flutter basisstructuur StatefulWidget

main.dart

```dart
class AboutUs extends StatefulWidget {
  @override
  _AboutUsState createState() => _AboutUsState();
}

class _AboutUsState extends State<AboutUs> {
  @override
  Widget build(BuildContext context) {
    return Container(
      // Overige UI-widgets die de About Us-pagina beschrijven.
    );
  }
}
```

Een StatelessWidget of StatefulWidget worden niet rechtstreeks aangeroepen. Je maakt altijd een klasse die `extend` ten opzichte van het gewenste type widget.

De functie build()

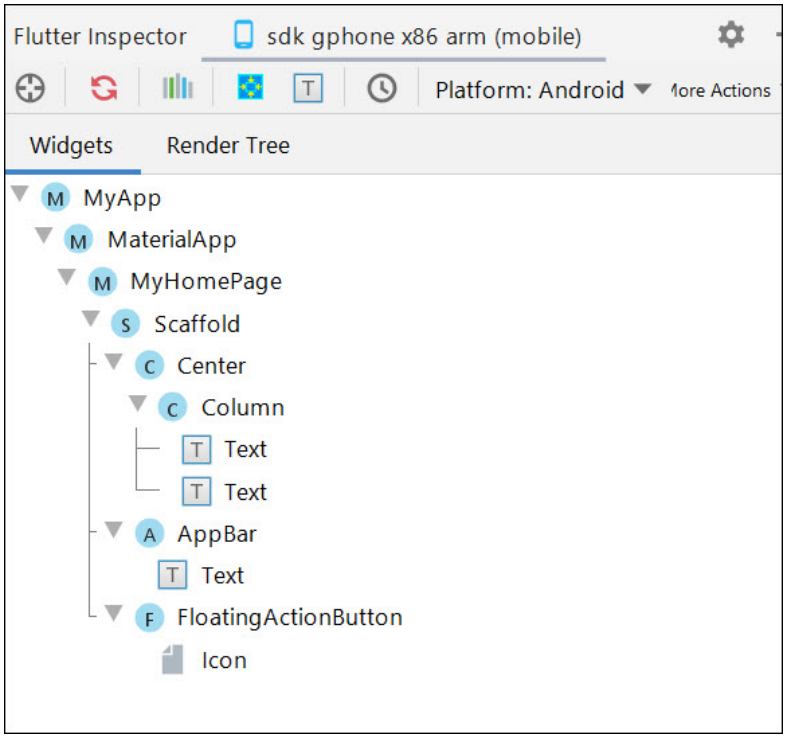
Elke widget *moet* een functie `build()` implementeren. Hierin wordt de widget tree opgebouwd.

De inhoud van `build()` is weer een andere widget! Dit is vaak de widget `Container()` of `Scaffold()`, waarin weer andere widgets zijn genest die stukjes UI voor hun rekening nemen.

Wanneer in een StatefulWidget de state wordt gewijzigd, wordt de functie `build()` opnieuw uitgevoerd om de wijzigingen in de UI door te voeren. State wordt altijd gewijzigd via een functie `setState()`.

De manier waarop Flutter met state omgaat deed me erg denken aan React.

In de voorbeeldapplicatie zie je bijvoorbeeld dat de widget tree bestaat uit een App met een Homepage waarin een widget `Scaffold()` is geplaatst. Deze roept op zijn beurt weer de widget `Center()` aan waarin een widget `AppBar()` is opgenomen, een widget `Text()` en een `FloatingActionButton()`.



De afbeelding komt uit Android Studio, het paneel `Flutter Inspector`. In code ziet de widget tree er op deze manier uit:

Flutter Widget Tree van basisppplicatie Homepage

main.dart

```dart
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ),
    );
  }
}
```

Gewoon een stukje tekst op de pagina typen of een afbeelding plaatsen kun je wel vergeten. Deze moet je minimaal wrappen in een widget `Text()`of een widget `Image()`. Bij afbeeldingen kun je daarnaast weer kiezen uit verschillende typen afbeeldingen (online, lokaal, avatars).

Tekst kan daarbij eigenschappen hebben zoals het lettertype, de uitlijning, de lettergrootte enzovoort. Dit kun je niet zomaar instellen (zoals in CSS), maar moet je regelen via de widget `TextStyle()`. Daarbij is 'rood' niet gewoon een kleur, maar een enum die is vastgelegd via het object `Colors.red`. Zoals gezegd, er zijn superveel widgets en properties die je moet leren kennen.

Compositie

Je maakt in Flutter dus widgets, die bestaan uit andere widgets. Deze widgets kunnen zelf weer widgets genereren en invoegen in de tree. Dat is erg krachtig en zorgt er voor dat je relatief eenvoudig generieke (bedrijfs-) widgets kan maken die je kunt hergebruiken in andere applicaties.

Als je het goed indeelt en beheert, is de *reusability* van widgets groot.

Je maakt dan als het ware een 'compositie' voor nieuwe applicaties op basis van bestaande widgets. Flutter implementeert hiermee een beetje het idee dat ook achter React Hooks en in de *Composition API* van Vue.js zit. Beslist een moderne architectuur en een denkwijze die nog jaren meekan. In ieder geval hoogstwaarschijnlijk langer dan de levensduur van de apps die je met Flutter maakt.

De widget tree

Mijn ervaring na een weekje Flutter is dat de widget tree enorm snel, enorm groot kan worden. Het gevaar loert dat het een onoverzichtelijk zooitje wordt.

Android Studio probeert je wel te helpen met visuele commentaren (die niet in de code worden geplaatst) die aangeven waar een bepaalde widget eindigt en waar een nieuwe begint, maar het blijft een enorme diepe piramide aan geneste widgets die je aan het maken bent.

Je kunt dit enigszins oplossen door custom widgets te maken die [een deel van] de widget tree samenstellen en retourneren, maar het blijft erg lastig om de structuur te

doorzien en te analyseren waar je een nieuwe rij of knop moet plaatsen als je bestaande code wilt aanpassen.

De afbeelding toont een supereenvoudige pagina met een aantal rijen en kolommen met daarin wat tekst. Je ziet nu al de enorme mate van nesting. Dit wordt er bij grotere applicaties alleen maar erger op.

Mijn advies: leer zo snel mogelijk het lay-out-systeem van Flutter met de widgets `Scaffold()`, `Container()`, `Column()` en `Row()` en de uitlijningsmogelijkheden. Pas dan kun je code begrijpen en eenvoudige pagina's maken.

Bestaande pagina's kun je dan sneller ontleden. Deze pagina helpt bij het begrijpen van het lay-outsysteem van Flutter: https://flutter.dev/docs/development/ui/layout/tutorial.

Dart

De programmeertaal Dart was relatief eenvoudig te leren. Ik heb zelf natuurlijk een JavaScript-achtergrond (en een.NET, maar geen Java), maar kon er snel mee overweg. Enkele items die mij opvielen:

- ✓ Alle variabelen zijn statisch getypeerd. Dit is verplicht.
- ✓ Elk statement wordt afgesloten met een puntkomma. Dit is verplicht.
- ✓ Keywords die mij opvielen:
  - o `dynamic` Vergelijkbaar met het TypeScript-type `any`. Een variabele kan dan verschillende typen inhoud krijgen.

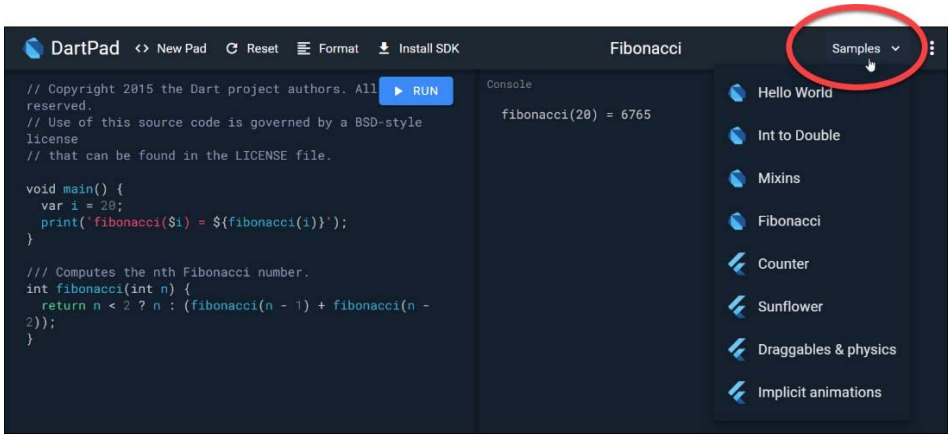
- `final` Variabelen kunnen niet meer gewijzigd worden (vergelijkbaar met `sealed` in andere talen).
- `Future` Een asynchrone waarde. Vergelijkbaar met een `Promise()` in JavaScript.
- `Map` Vergelijkbaar met een object literal in JavaScript, een variabele met key/valuepairs tussen `{...}`.
- Maar waarom worden typen als `int`, `double` en `bool` met een kleine letter geschreven, maar `String` met een hoofdletter?

✓ Een functie heeft *altijd* een returntype. Dit is verplicht het wordt aangegeven voorafgaand aan de naam van een functie, zoals in `void main() => ...`

✓ Het is niet straightforward om met JSON te werken. In JavaScript kun je JSON rechtstreeks benaderen als object, maar Dart beschouwd JSON gewoon als één lange string. Je moet het zelf parsen en omzetten als je binnen de applicatie met objecten verder wilt werken. Daar zijn wel functies voor aanwezig (`jsonEncode()` en `jsonDecode()` in de package `dart:convert`), maar je moet het toch maar weer doen.

Zoals gezegd, er zijn erg veel overeenkomsten met andere programmeertalen. Het leren van Dart is niet de grootste uitdaging als je met Flutter aan het werk wilt.

Je kunt met Dart oefenen in de online omgeving DartPad, op https://dartpad.dartlang.org/. Hier zijn ook een aantal (niet Flutter-specifieke) voorbeelden aanwezig. Ze helpen je om de taal te leren kennen.



Tutorial

Wil je iets verder dan alleen de Hello World-applicatie, dan kun je de online tutorial volgen op https://codelabs.developers.google.com/codelabs/first-flutter-app-pt1/#0. Hierin maak je een oneindige scrolling list, waarbij je kennismaakt met allerlei widgets en externe packages importeert. Het is een goed begin. Ik heb er veel van geleerd en was er in twee uurtjes doorheen.

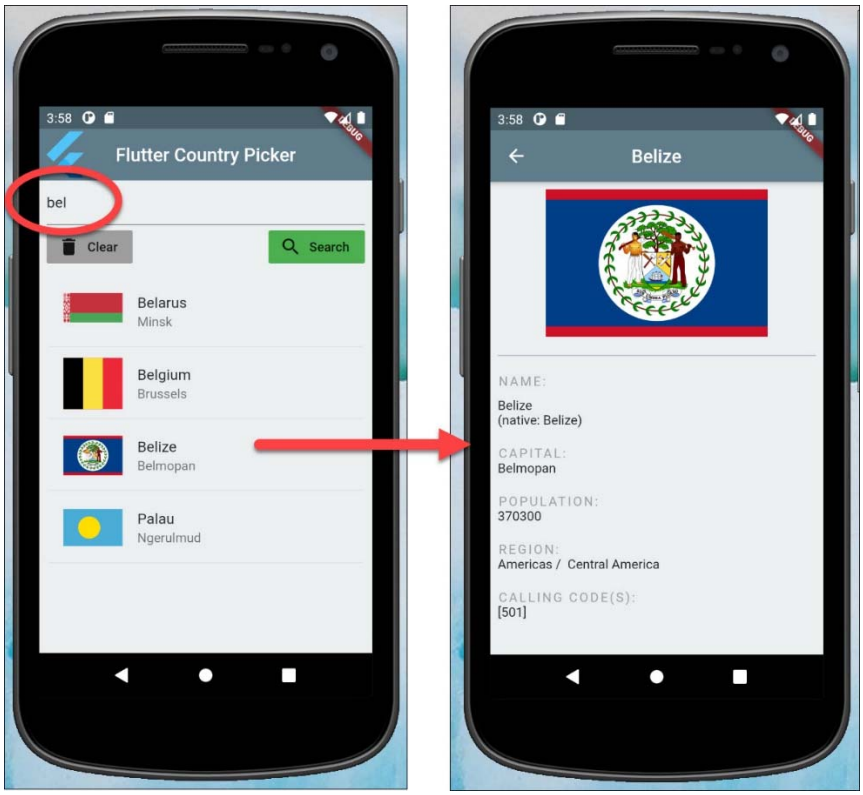
5. Hoe maak ik een Proof-of-concept applicatie?

Anders dan bij Svelte kon ik niet in een paar uurtjes de applicatie namaken die ik voor andere frameworks heb gemaakt. Het programmeren in Dart was niet het probleem.

Het kost daarentegen veel tijd om te leren werken met het lay-outsysteem van Flutter. Het uitzoeken wánneer je wélke widget gebruikt om de lay-out te realiseren die je in je hoofd hebt kost de meeste tijd. Zelfs als je zo'n eenvoudige applicatie wilt maken als ik hieronder heb gemaakt.

- ✓ De requirements zijn tamelijk eenvoudig, maar laten toch het complete stappenplan van een real-life app, inclusief communicatie met een backend zien.
- ✓ Toon een eenvoudige user interface waar de bezoeker kan zoeken op (deel van) de naam van een land.
- ✓ Communiceer met een backend - ik gebruik hiervoor de REST Countries API, beschikbaar op https://restcountries.eu - om een serie landen op te halen die voldoen aan het trefwoord van de bezoeker.
- ✓ Toon een lijstje met landen die aan het trefwoord voldoen in de user interface.
- ✓ Bij klikken op een land worden details van het betreffende land getoond in een pagina.

Je kunt deze applicatie zelf bekijken op github, op https://github.com/PeterKassenaar/flutter-demo-countries. Voel je vrij om de applicatie te downloaden, hem te openen in Android Studio en te draaien op je device of in de emulator.



Kijk hoe de data flow door de applicatie en de componenten is. Vervolgens kun je hem verder zelf uitbreiden. Mocht je aanpassingen of verbeteringen hebben, dan houd ik me natuurlijk aanbevolen voor PR's!

Dit is een tamelijk eenvoudige applicatie, nog zonder error handling, testing, animaties of state management. Wel is routing toegevoegd. De code kan nog (veel) verder verbeterd worden. Op de TODO-lijst staat bijvoorbeeld:

- ✓ API-calls centraliseren in een *service* die vanuit diverse widgets wordt aangeroepen. Nu heb ik dat per widget geregeld.
- ✓ Een `CountryModel`-class maken, zodat de opgehaalde landen kunnen worden aangeduid als instanties van die class. Nu zijn het gewoon `<dynamic>` elementen.
- ✓ De widget tree vereenvoudigen door compositie te gebruiken en terugkerende taken uit te besteden aan functies. Nu is elke widget gewoon van boven naar beneden opgebouwd. Het werkt, het ziet er goed uit, maar de code kan compacter en meer DRY.

Conclusie

Flutter is een framework dat zich qua functionaliteit en mogelijkheden absoluut kan meten met langer bestaande alternatieven als Xamarin, NativeScript en React Native. Het is qua performance sowieso beter dan frameworks die hun applicatie wrappen in een webview, zoals Ionic, Capacitor en Cordova.

Met Flutter maak je absoluut mobiele applicaties met een native look-and-feel, zonder dat je hiervoor aanvullende moeite hoeft te doen. De taal Dart is voor programmeurs eenvoudig te leren.

De uitdaging zit hem in het doorzien van de wijze waarop de widget tree wordt opgebouwd en hoe widgets andere widgets kunnen samenstellen en retourneren. Een Flutter-app kan naar mijn idee snel uitgroeien tot een nachtmerrie voor beheerders. Ook degenen die blanco inrollen in een project met een bestaande codebase kunnen het zwaar krijgen. Android Studio probeert je daarbij te helpen met structuurdiagrammen en commentaren, maar ik denk dat het inwerken in een bestaand project serieus tijd gaat kosten.

Op de vraag "*moet ik Flutter gaan leren?*", is zoals gebruikelijk geen eenvoudig antwoord te geven. Het enige juiste antwoord is meestal "*dat hangt er van af*". Om je te helpen de beslissing te nemen, zou je de volgende vragen kunnen beantwoorden.

- Ben je een **beginner in mobiele apps**? Dan is Flutter een uitstekende keuze. Je moet wel veel nieuws leren, maar je apps zijn direct beschikbaar voor Android en iOS. Op termijn kun je het web toevoegen als platform. Je hoeft je niet te verdiepen in C#/Xamarin of Java of eerst React te leren. Met Flutter ben je - ook als bedrijf - relatief snel up-and-running en voorbereid op de toekomst. In ieder geval voor de komen 5-7 jaar is mijn inschatting.
- Ben je **JavaScript-developer** en maak je websites met HTML, CSS en JavaScript? Dan helpt het een beetje als je moderne CSS-lay-outopties kent, maar verder zul je veel nieuws moeten leren. Ga uit van een stevig leertraject.
- Ben je **developer in Angular of Vue**? Dan zul je Dart als nieuwe programmeertaal moeten leren. De overstap vanuit JavaScript is echter relatief eenvoudig. Ik heb het zelf gedaan de afgelopen week. Als je al gewend bent met

TypeScript te werken wordt het nog makkelijker. Het is een kwestie van syntaxis-wijzigingen. Maar toch - je moet er weer iets bij leren en je moet je kennis voor twee verschillende technologieën onderhouden. Dat kost wel tijd - en dus geld. Extra kennis is altijd waardevol, maar komt met een prijs.

- Ben je **React-programmeur**? Ga dan verder met React Native. Ook dit platform is volwassen, er is veel vraag naar, een grote community voor beschikbaar en je hoeft niet veel nieuwe technieken te leren. Flutter is in dat geval overkill. De meerwaarde ervan ten opzichte van React Native is mijns inziens gering.
- Ben je **fullstack JavaScript**-programmeur en schrijf je Node.js-API's en -applicaties en websites in een van de populaire webframeworks? Dan ligt het wellicht meer voor de hand om een JavaScript-mobiele toepassing te gebruiken als Cordova of, nieuwer, Ionic/Capacitor. Tenzij ruwe performance of pure native look-and-feel een topprioriteit is. Daarin is Flutter weer beter, omdat het niet de overhead van een webframework en -view heeft en niet met een CSS-implementatie de native UI-onderdelen hoeft na te bootsen.
- Ben je **Java- of C#-developer**? Dan sta je voor een lastige keuze.
    - Als *Java-programmeur* staat native Android-development (met Java of Kotlin) waarschijnlijk dichter bij je en ben daar sneller productief in. Je maakt dan echter geen crossplatform apps. Het kan de moeite lonen Dart te leren en het UI-systeem van Flutter onder de knie te krijgen. Omdat je al gewend bent te werken met Material Design, Futures en asynchrone code zal dat evenwel ook niet al te lastig zijn.
    - Als *C#-developer* ligt de keuze voor Xamarin meer voor de hand. Hiermee maak je wel crossplatform apps. Maar het is niet meer het meest voor de hand liggende framework voor app-ontwikkeling. Voordeel is echter wel dat Xamarin uitstekend is geïntegreerd in Visual Studio (waar je nu waarschijnlijk toch al mee werkt) en uitstekend past binnen de stack van andere .NET-applicaties en technieken (Internet Information Server, SQL Server, AD enzovoort). Als je 'gewone' applicaties in .NET maakt, en mobiele apps in Flutter gaat schrijven, moet je een compleet nieuwe stack en -toolset leren. Je kunt je afvragen of dat de moeite waard is.

Maak op basis van bovenstaande vragen een keuze. Je kunt altijd eens kijken op https://flutter.dev/ om te zien of het wat voor je is. En uiteraard help ik je ook graag verder als jij of je bedrijf een Flutter-vraag heeft.

Auteur bio

Dit artikel is geschreven door *Peter Kassenaar*. Peter heeft meer dan 20 jaar ervaring in webdevelopment en heeft talloze boeken over webdevelopment en verwante technieken gepubliceerd. Hij is een bekende trainer in West-Europa en reist de wereld over om training te geven. Peter staat bekend om zijn intensieve, geconcentreerde manier van lesgeven en praktische hands-on workshops. Daarbij verliest hij echter nooit het belang van de individuele deelnemer uit het oog. Hij biedt veel persoonlijke aandacht en ondersteuning tijdens de workshops en oefeningen. Je kunt Peter volgen op Twitter (@PeterKassenaar, Nederlands en Engels) en LinkedIn (https://www.linkedin.com/in/peterkassenaar/). Zijn persoonlijke website is te vinden op https://www.kassenaar.com.

Ik geef training in frontend technieken zoals Angular, React en Vue - en binnenkort ook Flutter. Ook in coronatijd, ook virtueel, ook op jouw bedrijf. Laat weten wat je leervraag is, dan zoeken we een oplossing!