Project Dossier

REIZEN TECHNOLOGIE

Christian Sollai
Daan Van Hirtum
Koen De Deckers
Nino Kerkhofs
Pieter Coenen
Sibert Schurmans
Stijn Meerten
Zeno Frooninckx

Inhoud

Opdracht omschrijving

Voor het vak ICT Projects Agile hebben we de opdracht gekregen om een cross-platform applicatie te maken volgens de Agile methodiek. Deze applicatie moet dienen ter vervanging van de papieren reisplanning voor de reizen van UCLL. De applicatie moet voldoen aan een aantal eisen die zijn gesteld door de productowner. De productowner voor deze app, is onze docent Mr. Segers. De eisen van de app zijn de volgende: De gebruikers moeten kunnen inloggen met hun r-nummer en een wachtwoord. Zodra de gebruiker zich heeft ingelogd en de voorwaarden heeft geaccepteerd, moet de gebruiker zich voortaan niet meer inloggen en wordt hij/zij direct doorverwezen naar de home pagina. Op de home pagina ziet de gebruiker de activiteiten van de huidige dag, het hotel en de algemene info. Voor het vertrek van de reis is er ook en countdown te zien. Boven aan het scherm is er op elke pagina een balkje met een telefoon te zien. Als de gebruiker hierop klikt, dan krijgt hij/zij een dropdown met alle noodnummers van de trip. Als de gebruiker op een nummer klikt, kan er meteen gebeld worden. Verder is er ook een pagina waar de volledige planning staat. Een pagina met alle hotels, een pagina om de auto indeling te raadplegen en een pagina waar de nummers van alle reizigers terug te vinden zijn. Wanneer de app wordt opgestart, met wifi-verbinding, wordt automatisch de nieuwe data gedownload uit de database. Als de app wordt opgestart met 4G, krijgt de gebruiker een waarschuwing en krijgt hij/zij de keuze om te updaten. Binnen de app is er ook een refresh icoon naast de noodnummers, zodat de gebruiker altijd manueel kan updaten.

Agile

Agile is een methodiek voor het maken van projecten, en de tegenhanger van het watervalsysteem. Bij het watervalsysteem wordt het hele project geanalyseerd voor er code wordt geschreven en is er pas feedback van de klant bij de oplevering van het product. Hierdoor bestaat de kans dat de klant niet tevreden is met het product.
Bij het agile systeem is daarentegen veel meer feedback van de productowner. Agile betekent flexibel, en dit zien we ook terug bij deze methodiek. Agile is geen methode op zich, maar er zijn meerdere methodes die onder de noemer agile vallen.
Voor ons project gebruiken we 2 methodes van agile. Deze methodes zijn Scrum en Kanban. Deze 2 methodes vullen elkaar aan en zijn ideaal om samen te gebruiken.
De methode die centraal staat is Scrum. Binnen Scrum gaan we de volledige app opdelen in kleinere delen, dit noemen we user stories. Deze user stories zijn kleine onderdelen die al volledig werkende zijn. Als de klant beslist om in het midden van het project te stoppen, dan heeft hij/zij al een werkend product met de afgewerkte user stories.
Wanneer alle user stories zijn bepaald, gaan we deze uitschijven in de “Als → wil ik → zodat ik”-structuur. Op deze manier is alles duidelijk opgesteld. Bij twijfel of dit wel is wat de klant wilt, kan de klant altijd geconsulteerd worden. Wanneer alle user stories volgens deze structuur zijn opgesteld, vormen deze samen de backlog.

De backlog bevat alle user stories die gemaakt moeten worden vooraleer het product volledig af is naar de wensen van de klant. Wanneer de volledige backlog af is gaan we elke user story een story point geven.

Dit punt geeft aan hoeveel werk en hoe moeilijk het is om een story af te werken. Deze punten gaan we geven aan de hand van Poker planning.
Poker planning is een proces waarbij iedereen een set speciale kaarten heeft. Deze kaarten hebben allemaal een nummer. Het laagste getal is 0 en het hoogste is 100. De overige nummers zijn ½, 1, 2, 3, 5, 8, 13, 20 en 40. Verder is er ook nog een vraagteken, voor als je de story niet begrijpt, en een pauze kaart, voor als je pauze nodig hebt. Om de poker planning uit te voeren kiezen we eerst een taak die iedereen kent en geven dit de moeilijkheid 1.

In ons geval hebben we de taak “1 item uit een database halen” gekozen. Nu gaan we alle stories een punt geven aan de hand van deze moeilijkheid. Iedereen steekt tegelijk het punt op dat hij vind dat deze story verdient que moeilijkheid. De personen met de hoogste en de laagste punten geven hun argumenten, en dan gaat iedereen opnieuw tegelijk zijn punt opsteken. Dit gaat door tot iedereen hetzelfde nummer opsteekt. Dit nummer wordt dan het story point voor die user story.

Wanneer dit voor alle stories gedaan is, hebben we een aantal punten voor het hele project. Nu gaan we kijken hoelang we bezig zijn met 1 punt. Zo kunnen we gaan inschatten hoelang we bezig zijn met het project en wat de kostprijs gaat zijn.
Nu gaan we alle stories ook nog een priority geven. Hier gaan we kijken welke stories de klant als eerste wil hebben en welke de meeste waarde hebben. Aan de hand van deze priority gaan we selecteren welke stories we als eerste gaan afronden.

Nu kunnen we gaan beginnen met onze sprints. Een sprint is een periode van 2 tot 4 weken waarin we bepaalde user stories gaan afwerken. Gedurende deze periode gaan we ook dagelijks kijken hoever iedereen staat en of er problemen zijn. De eerste sprint noemen we sprint 0.

In deze sprint gaan we nog niet aan het project werken, maar gaan we indien nodig info over het project opzoeken. Deze sprint duurt meestal niet zolang als een volledige sprint.
Om een sprint te beginnen gaan we eerst een sprint planning houden. Hier wordt beslist hoeveel punten we gaan opnemen en welke stories we gaan opnemen. We gaan ook voor alle user stories acceptatiecriteria schrijven. Dit zijn alle items waaraan dit deel moet voldoen om volledig klaar te zijn. Verder gaan we ook de gekozen user stories opdelen in kleinere taken. Deze taken zijn vergelijkbaar met de acceptatiecriteria. Deze taken worden verdeeld onder alle leden van het team.

Het team dat aan het project werkt is verdeeld in meerdere rollen. We hebben het ontwikkelteam, zij gaan alle code schrijven. De scrummaster, hij zorgt ervoor dat het scrum systeem goed werkt. En we hebben de productowner, hij/zij is de klant en bij hem/haar kunnen we terecht met eventuele vragen. In ons project is er enkel een scrummaster bij de besprekingen, en dit is telkens een andere persoon.

Om de werking zo soepel mogelijk te maken is er dagelijks een stand-up bij de sprint. Dit is een korte “vergadering” van maximum 15 minuten waarbij iedereen zegt wat hij gisteren heeft gedaan, wat hij morgen gaat doen en of hij problemen heeft. In ons geval was het niet haalbaar om dit dagelijks te doen dus hebben wij dit minimum 2 keer per week gedaan en ook zoveel mogelijk gecommuniceerd via een chatgroep op Discord.

Verder is er na elke sprint een sprint meeting. Hier gaan we samen met de productowner kijken of alles is afgeraakt en of er opmerkingen zijn over het opgeleverde deel. Als er stories niet af zijn of er zijn opmerkingen, dan worden deze opgenomen in de volgende sprint. Om dit makkelijk uit te voeren gebruiken wij easybacklog.com.

Na iedere sprint gaan we ook een restrospective houden. Bij een restrospective gaan we opschrijven wat we gaan blijven doen, waarmee we gaan stoppen en wat we moeten beginnen doen. Op deze manier kunnen we ons elke sprint verbeteren en zorgen dat alles vlotter verloopt. Als dit is gedaan, dan kunnen we de sprintplanning van de volgende sprint gaan uitvoeren. Dit proces gaan we herhalen tot het hele project af is.

Naast Scrum maken we ook gebruik van Kanban. Dit gaan we gebruiken tijdens onze sprints. Een Kanban bord is een ideaal hulpmiddel tijdens onze sprint om bij te houden hoever iedereen staat. Een Kanban bord heeft de vakken To Do, Testing en Done. We plakken de taken van de sprint op dit bord om te kunnen bijhouden hoever we staan met de sprint. Wij gebruiken hiervoor Trello. Dit is een online tool om een Kanban bord te maken en te delen. Ons bord heeft de volden Tasks(To Do), Busy, Testing en Done. We hebben het dus iets aangepast ten opzichte van het originele Kanban. Dit zijn alle methodes die we gaan gebruiken van agile.

Product Backlog

Bij een eerste analyse van de opdrachtomschrijving tracht je de grote functionele eenheden af te splitsen. In scrum technologie noemen we dit “epic stories”. In ons project waren er twee epics: Backend en App waarvoor we de userstories hebben uitgewerkt.

Epic	Story	Story points	priority
App 1	**Als** geregistreerde gebruiker **wil ik** wanneer de app opstart een overzicht krijgen van de dagplanning **zodat ik** weet wat ik die dag moet doen en de gegevens krijg van de auto en kamer indeling.	5	
App 2	**Als** geregistreerde gebruiker met netwerkverbinding **wil ik** telkens als ik de app open de nieuwe info van de remote database ophalen **zodat ik** de meest recente gegevens kan raadplegen in de app en deze opgeslagen worden op het toestel.	13	
App 3	**Als** geregistreerde gebruiker **wil ik** alle noodnummers via een klik kunnen raadplegen **zodat ik** in geval van nood de informatie snel ter beschikking heb.	3	
App 4	**Als** gebruiker **wil ik** mij (eenmalig) kunnen aanmelden via mijn inloggegevens indien ik netwerkverbinding heb **zodat ik** algemene voorwaarden kan bevestigen waarna ik toegang krijg tot de app.	5	
App 5	**Als** geregistreerde gebruiker **wil ik** de refreshknop kunnen bedienen **zodat ik** de nieuwe data ophaal met een waarschuwing indien ik mobiel ben.	3	
App 6	**Als** geregistreerde gebruiker **wil ik** de knop overzicht kunnen bedienen **zodat ik** een chronologisch overzicht krijg van de reis gegroepeerd per bestemming.	5	
App 7	**Als** geregistreerde gebruiker **wil ik** de lijst van alle hotels kunnen raadplegen **zodat ik** kan zien in welke hotels ik ga verblijven.	5	
App 8	**Als** geregistreerde gebruiker **wil ik** in de lijst van hotels op een hotel kunnen klikken **zodat ik** informatie krijg van het betreffende hotel en de indeling van alle kamers voor dat hotel kan raadplegen.	8	
App 9	**Als** geregistreerde gebruiker **wil ik** de auto-indeling kunnen raadplegen **zodat ik** kan zien wie bij wie in de auto zit	5	
App 10	**Als** geregistreerde gebruiker **wil ik** de telefoonnummers van mijn medereizigers kunnen bekijken **zodat ik** deze kan contacteren.	5	
App 11	**Als** geregistreerde gebruiker **wil ik** specifieke contacten kunnen opzoeken door de naam van een persoon in te geven in een zoekbalk in de pagina "contacten" **zodat ik** een specifieke persoon kan contacteren indien nodig.	5	
App 12	**Als** geregistreerde gebruiker **wil ik** alle relevante info van de reis kunnen raadplegen **zodat ik** voor het vertrek weet waarmee er rekening gehouden moet worden.	5	

App 13	**Als** geregistreerde gebruiker **wil ik** op een planning kunnen drukken **zodat ik** alle activities in de planning kan zien.	3	
Bac 1	**Als** organisator **wil ik** als ik in het menu op “Planning” druk **zodat ik** alle dagplanningen kan zien van de reis waar ik aan gekoppeld ben.	5	
Bac 2	**Als** organisator **wil ik** als ik op de knop “toevoegen druk” **zodat ik** deze dagplanning aan de reis kan toevoegen.	5	
Bac 3	**Als** organisator **wil ik** als ik op de pagina planning zit en bij een bepaalde planning op de knop “wijzig planning” druk **zodat ik** deze planning kan wijzigen.	3	
Bac 5	**Als** organisator **wil ik** als ik in het menu op “algemene info” druk, naar een pagina word herleid **zodat ik** de algemene info kan zien.	5	
Bac 6	**Als** organisator **wil ik** op de pagina algemene info iets in het tekst vak type en op de knop “opslaan” drukken **zodat ik** de algemene info kan aanpassen.	3	
Bac 7	**Als** organisator **wil ik** op de pagina planning op een knop drukken **zodat ik** aan deze planning een activity kan koppelen en wegschrijven.	5	

Sprint 0

Volgens de scrum methode moet elke sprint een stukje werkende code opleveren met een toegevoegde waarde voor het gehele project.

Elke Sprint bevat hiervoor een combinatie van analyse, ontwerp, infrastructuur, ontwikkeling, testen en integratie.

In deze optiek verschilt sprint 0 van alle andere sprints. Sprint 0 is de voorbereiding van het gehele project en levert geen werkende code op. Sprint 0 mag echter ook nooit de intentie hebben om tegemoet te komen aan het waterval systeem.

In sprint 0 gaan we de globale aspecten van het project vast leggen. De programmeer omgeving en de te gebruiken technologieën worden vastgelegd evenals een basisstructuur voor de database en een globale opzet voor de userinterface. Belangrijk is dat je bij de design elementen zoals de basisstructuur voor de database en de globale opzet van de userinterface niet te ver in detail gaat (waterval systeem). Het moet telkens een leidraad zijn waarop je verder werkt in de volgende sprints.

Wij noemen dit sprint 0, anderen noemen dit de voorbereiding.

Flutter & Dart + MVVM

Flutter & Dart

Inleiding

De revolutie van cross-platform appontwikkeling begon toen Microsoft in 2011 de Xamarin SDK lanceerde. Omdat Xamarin gebruikt maakt van de taal C# die al veel ontwikkelaars kennen, wordt het plots gemakkelijker om mobiele apps te schrijven voor de (op dat moment nog drie) verschillende besturingssystemen. In 2013 gaf Ionic een platform aan de web developers die hun skills wilden gebruiken om mobiele apps te gebruiken. Facebook lanceerde twee jaar later React.js dat gebruikt maakt van Javascript. Flutter is het antwoord van Google op de evolutie van de cross-platformmarkt. De eerste versie van Flutter lanceerde pas in december 2018.

Dart

Dart is de programmeertaal die gebruikt wordt om Flutter apps te schrijven. Het is ook een product van Google en is ontwikkeld voor web-, server- en mobiele toepassingen alsook voor IoT-toestellen. Het is een objectgeörienteerde taal met een syntaxis die lijkt op C. voor web toepassingen kan ze worden gecompileerd naar Javascript. Ze ondersteunt interfaces, abstracte klassen, generics en optional typing.

Flutter

De vier grote componenten waaruit Flutter bestaat zijn het Dart platform, de Flutter engine, de foundation library en de design-specific widgets.

Het Dart platform voorziet de “Just in time compilation” functie voor Flutter. Hiermee kan de ontwikkelaar zijn app “hot reloading” waarmee aanpassingen van broncode onmiddellijk kunnen worden doorgevoerd in een draaiende app, zonder dat de app moet herstarten. Deze functie is een grote reden dat Flutter populairder wordt.

De Flutter engine, die in C++ is geschreven, voorziet de basisfuncties van Flutter. Het implementeert de basisbibliotheken, de plug-in architectuur, de bestand en netwerk IO (input & output) en een hele set platform-, lay-out- en basiswidgets.

De foundation library is geschreven in Dart en voorziet de basisklassen en functies die worden gebruikt om een Flutterapplicatie op te bouwen.

Ten slotte zijn er ook twee bibliotheken van design-specific widgets om te gebruiken in Flutter. “Material Design” implementeert de design language van Google, “Cupertino” implementeert de “Human Interface Guidelines iOS design” van Apple.

Installatie Flutter

1. https://storage.googleapis.com/flutter_infra/releases/stable/windows/flutter_windows_v1.9.1+hotfix.2-stable.zip
2. Uitpakken naar C-schijf
3. Omgevingsvariabelen van computer aanpassen.
   3.1. Zoek omgevingsvariabelen bij start
   3.2. Selecteer Path en klik op bewerken
   3.3. Klik op nieuw en typ C:\flutter\bin
   3.4. Klik op Ok
4. Open een command prompt en run flutter doctor
5. Run flutter doctor –android licenses
6. Accepteer alle licenses door op y en enter te duwen. (6x)

7. Start de Android Studio Ide
8. Ga naar File>Settings>Plugins
9. Selecteer flutter en klik op Install en restart de Ide

10. Klik op Tools>AVD Manager>Create Virtual Device
11. Selecteer een smartphone naar keuze en klik op Next
12. Download Android Q en klik op Next
13. Selecteer Hardware – GLES2.0 onder Emulated Performance en klik op Finish
14. Sluit het venster en kies de geïnstalleerde smartphone in de bovenste balk
15. Wacht tot de emulator is opgestart en klik dan pas op play

Om hot reloading te testen:

16. Ga naar regel 95 en verander het woord pushed naar clicked
17. Klik op het bliksemschichtje

Voorbeeld: basisapp hello world

```dart
import 'package:flutter/material.dart';
void main() => runApp(HelloWorldApp());

class HelloWorldApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hello World App',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Hello World App'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Maak je eerste app via: https://flutter.dev/docs/get-started/codelab

Widgets

Futurebuilder

Futurebuilder is een widget die we gaan gebruiken wanneer we data moeten inladen van een snapshot. Een snapshot is data die uit een database komt. De futurebuilder gaat zichzelf opbouwen aan de hand van de opmaak die we hebben geprogrammeerd en de data die uit het snapshot gehaald wordt.

Een futurebuilder werkt asynchroon, dit wil zeggen dat we andere functionaliteit kunnen uitvoeren terwijl de data wordt opgehaald. Om aan te tonen dat de data wordt opgehaald maken we gebruik van een CircularProgressIndicator. Dit is een laadicoon dat op het scherm zichtbaar is tot de widget gebouwd is en de juiste data opgehaald.
Een futurebuilder is dus een ideaal hulpmiddel om widgets asynchroon op te bouwen aan de hand van data die we uit een database gaan halen.

Stateless/statefull

In Flutter zijn alle UI-componenten widgets. De Widgets die code voor een bepaald scherm inladen kan men opdelen in twee types: stateless en statefull.

Statefull widgets zijn onveranderlijk en kunnen hun state niet veranderen terwijl dat de app draait. In een statefull widget kan de buildfunctie dus slechts één keer worden aangeroepen. De build methode zorgt voor het tonen van de widgets op het scherm. Voorbeelden van statefull widgets zijn: Icon, Chip en Text.

Statefull widgets hebben een veranderlijke staat en kunnen meerdere keren aangepast worden. Voorbeelden van statefull widgets zijn: Checkbox, Radio en Textfield.

Scaffold

Een scaffold gaan we binnen een widget gebruiken voor het basis materieel design van de widget te gaan opstellen. Scaffold geeft ook API's mee om opmaakelementen te laten zien binnen de widgets. Via deze elementen kunnen we dan onze widgets gaan opbouwen.

Dependencies

Het is vaak handig om packages te gebruiken die andere ontwikkelaars hebben geschreven, om niet alles volledig zelf te moeten ontwikkelen. Packages kan je vinden op pub.dev. Om deze toe te voegen in de app, moet de naam van het package in pubspec.yaml onder dependencies worden geschreven. Klik hierna op "packages get" of geef het commando "flutter pub get" in cmd. Hierna kunnen de functies van het geïnstalleerde package overal in de app worden gebruikt. Bovenaan de widget moet dan enkel het package geïmporteerd worden. Packages kunnen tot slot ook geüpdatet worden door in pubspec.yaml op "packages upgrade" te klikken.

MVVM

MVVM (Model-View-ViewModel) is een architecturaal patroon om code te schrijven. Bij een MVVM-structuur gaan we het grafische aspect (de view) scheiden van de backend logica. Dit wil zeggen dat we bij een MVVM-structuur de view zien als een template die we gaan opvullen met info. Op deze manier kunnen we gemakkelijk info gaan wijzigen/toevoegen zonder de hele opmaak van het project te moeten gaan wijzigen.

Naast de view is er ook het model. In het model gaan we verwijzen naar de inhoud die in de view gaat komen. Deze inhoud gaan we halen uit een database. Voor elke tabel binnen een database gaan we ook een model aanmaken. Dit model bevat alle velden binnen de tabel. Ann de hand van onze models kunnen we nu de data uit de database halen en deze in de view steken.

De laatste component is het viewmodel. Binnen het viewmodel gaan we de data ophalen en “klaarmaken” om deze te gaan gebruiken binnen de view. Om de data te gaan ophalen maken we gebruik van query’s. Een query is een opdracht die gaat uitgevoerd worden binnen een database. Dit is niet alleen voor het ophalen van data, maar ook voor het invoegen, bewerken of verwijderen van data. We kunnen met de query gegevens ophalen aan de hand van de velden die we hebben aangemaakt in de models van ons project. Wanneer we de info hebben opgehaald kunnen we deze nu in onze view gaan steken. Als zowel het model, de view als het viewmodel werken, hebben we een werkende pagina die we kunnen testen.

In de meeste gevallen gebruiken we een MVVM-structuur die voorgeschreven is door de programmeeromgeving (in ons geval Android Studio) of die we kunnen downloaden van een andere bron (bijvoorbeeld GitHub). Maar omdat Flutter & Dart een taal is die in normale omstandigheden niet met een MVVM-structuur werkt, hebben we zelf voor een structuur moeten zorgen.

Om te beginnen hebben we binnen ons project een mappenstructuur aangemaakt, zodat alles overzichtelijk blijft. We hebben respectievelijk een map voor de models, de viewmodels en de views. Op deze manier moeten we niet gaan zoeken welke bestanden waar staan. Vervolgens hebben we binnen onze groep de nodige afspraken gemaakt over hoe de code geschreven moet worden binnen de MVVM-structuur.
Deze afspraken zijn de volgende:

- Van alle tabellen die we nodig hebben binnen de applicatie wordt een model gemaakt binnen de map models. Buiten de map models wordt er geen code geschreven die hier iets mee te maken heeft. Elk model heeft ook dezelfde naam als de bijbehorende tabel in de database.

Alle query's worden geschreven binnen de map viewmodels. Voor elke pagina die gemaakt wordt binnen het project wordt er een viewmodel aangemaakt met dezelfde naam. Binnen een viewmodel worden enkel de nodige functies en query's geschreven voor die pagina.

- Binnen de views wordt geen code geschreven die te maken heeft met het ophalen van data. Hier wordt enkel de opmaak van de pagina in aangemaakt. De view heeft een duidelijke naam die beschrijft welke pagina gemaakt is met deze view.

Als de bovenstaande afspraken nageleefd worden, is de code die wordt geschreven transparant en overzichtelijk voor iedereen. Dit is ook gemakkelijker om code te gaan toevoegen of code van iemand anders aan te passen.

GitHub

Hedendaags worden programmeer-projecten steeds meer in teamverband uitgevoerd. Om te zorgen dat er geen dubbel werk wordt gedaan en dat de code voor iedereen ten allen tijde beschikbaar is bestaan er meerdere versiecontrolesystemen.

Bij versiecontrolesystemen wordt een versie van het project bijgehouden op een makkelijk bereikbare server, een repository. Teamleden gaan aan de hand van lokale kopieën van deze repository veranderingen aan het project kunnen uitvoeren. Deze veranderingen worden op hun beurt weer toegepast op de algemeen bereikbare repository.

Er zijn meerdere services die het gebruik van repositories aanbieden. In dit verslag gaan we echter alleen naar git (een service) en Github (server/website) kijken. Hierbij gaan de basisprincipes van repositories en hun functies centraal staan.

Er zijn meerdere tools beschikbaar voor met een Github repository te werken maar in dit verslag gaan er slechts twee besproken worden: VCS van Android Studio en Gitbash, een codeline tool van Microsoft specifiek voor git.

Wat is Github

Github is een online platform waarop programmeer-projecten in team uitgewerkt kunnen worden. Hiervoor wordt een project aangemaakt in een "repository". Deze repository weergeeft alle bestanden en veranderingen in/aan het project.

Afhankelijk van het type repository is het downloaden van het project mogelijk voor iedereen met toegang. Dit wil zeggen dat er alleen een internet verbinding nodig is om de laatste versie van het project af te halen. Ook kan iedereen met toestemming aanpassingen aan het project uploaden, zodat deze meteen voor iedereen beschikbaar zijn.

Wat is een repository

Een repository is de verzameling van bestanden in een project en de bijbehorende documenten zoals een README. Veranderingen aan een repository worden door Github bijgehouden en gesynchroniseerd aan de hand van verschillende commando's. Zodra een bestand aan een online of remote repository is toegevoegd is hij voor iedereen beschikbaar. Bij het toevoegen van een bestand aan een lokale repository zal Github bij het synchroniseren van de remote repository met de lokale repository, het lokaal bestand toevoegen aan de remote repository.

Het aanmaken en opzetten van een Github-repository
VIA GITHUB.COM

Voor het aanmaken van een repository moet je een geregistreerd Github-gebruiker zijn. Zodra alle stappen van de registratie doorlopen zijn, is het mogelijk een repository aan te maken.

Om een repository aan te maken is er op de profielpagina onder het tab repositories een knop new te vinden. Druk hierop en er volgt een doorverwijzing naar een formulier voor het aanmaken van een repository.

De enigste vereiste van dit formulier is dat je een naam invult voor de repository. Het is verder mogelijk om de beschrijving, de toegankelijkheid aan te passen en een paar startbestanden (zoals een README) mee te laten aanmaken.

Zodra het formulier naar eigen voorkeur is ingevuld, met minimum een repository-naam, is het mogelijk om op de knop “Create repository” te drukken, en de repository aan te maken. Zodra je op de knop hebt gedrukt word je meteen doorverwezen naar de hoofdpagina van je nieuwe repository.

De laatste stap voor het opzetten van een repository is het toevoegen van “Collaborators”, de personen die gaan meewerken aan het project. Dit kan onder het tab “Settings” van de repository. Hier kan onder het tab “Collaboraters” iemand toegevoegd worden aan de hand van een Github-gebruikersnaam of een email-adres. Hou er rekening mee dat alleen personen met een Github-account kunnen meewerken aan een repository.

Dit is de basisopzet van een repository in Github op de webpagina. Er zijn verdere instellingen mogelijk qua beveiliging en dergelijke, maar hier gaan we niet verder op in omdat dit voor gevorderden is.

Om deze repository te gebruiken moet er een initieel project geüpload worden. Dit kan zowel via Android studio als via Gitbash. Om dit te doen wordt er een lokale repository aangemaakt die gaat gelinkt worden met de online repository ook wel de remote repository genoemd.

INITIALISEREN VIA ANDROID STUDIO

Voor het initialiseren van een repository via Android Studio moeten de volgende stappe in een geopend project ondernomen worden:

- Onder “VCS” → klik op “Enable Version Control Integration”.
- Selecteer in het pop-up menu “git” en druk vervolgens op “OK”.
- Onder “VCS” → klik op “Commit…”.
- Zorg dat in het pop-up menu alle files geselecteerd zijn.
- Vul een commit message in.
- Druk op “Commit”.
    - Op dit punt wordt er een controle van het project gedaan. Indien er fouten of waarschuwingen in het project zijn komt er een pop-up bericht om te vragen om deze na te kijken. Indien hier op “Commit” wordt geduwd, worden deze fouten en waarschuwingen genegeerd. Bij ”Review” krijg je een overzicht waar deze fouten en waarschuwingen zich bevinden.
- Onder “VCS” → “Git” → klik op “Push…”.
- In het pop-menu, klik in de lijst naast “master” op “Define remote”.
- Vul bij “URL:” de url van de gewenste repository in en druk op “OK”. Deze URL heeft de volgende stijl: https://github.com/”github-gebruikersnaam”/”repository-naam”.
- Druk op “Push”.

Na even wachten staat het initiële project op de repository en kan deze door anderen afgehaald en aangepast worden.

INITIALISEREN VIA GITBASH

Voor het initialiseren van een repository via Gitbash moeten de volgende stappen ondernomen worden:

- Navigeer met het “cd” commando naar de plaats waar het project opgeslagen staat.

```
$ cd C:/ProjectenMap/Project
```

- Maak een lokale repository aan op het project met “git init”.

```
$ git init
```

- Voeg alle bestanden van het project toe aan de lokale repository met “git add .”.

```
$ git add .
```

- Zet alle veranderingen in de lokale repository klaar om doorgestuurd te worden en geef een bericht mee met de omschrijving van deze veranderingen.

```
$ git commit -m 'Bericht naar keuze'
```

- Link de remote repository aan het project met een duidelijke naamgeving (meestal origin) met “git remote add”
    - De remote-naam kan vrij gekozen worden, maar dient voor verdere commando’s. de naamgeving hiervan moet dus heel duidelijk zijn.
    - De repository-URL heeft de volgende stijl: https://github.com/”github-gebruikersnaam”/”repository-naam”)

```
$ git remote add remote-naam repository-url
```

- Stuur de voorbereide bestanden van de lokale repository door naar de remote repository
    - De repository-naam is dezelfde als die is gegeven in de vorige stap.

```
$ git push -u repository-naam master
```

Na een korte tijd is het initiële project beschikbaar op de remote repository. Dit kan enkele minuten duren, afhankelijk van de internetconnectie en de grootte van het initiële project.

Het connecteren met een reeds bestaande Github-repository

Na het aanmaken en initialiseren van een repository moeten andere teamleden ook het project kunnen aanpassen. Hiervoor moet er een lokale repository gekoppeld aan de remote repository aangemaakt worden. Dit proces behoort onder “Clone”, en is het aanmaken van een nieuw project aan de hand van een bestaande remote repository.

CONNECTEREN VIA ANDROID STUDIO

Als in Android Studio reeds een project openstaat moeten de volgende stappen ondernomen worden:

- Onder “Files” → “New” → “Project from Version Control” → klik op “Git”
- Vul de URL van de repository in (format: https://github.com/”Github-gebruikersnaam”/”repositorynaam”
- Kies een bestandslocatie en naam voor de map voor het gekloonde project aan te maken
- Klik op “Clone”.

Als echter nog geen project geopend staat, zijn de beginstappen iets anders. Maar het algeheel proces is bijna hetzelfde als hierboven. De stappen hiervoor zijn:

- Klik op “Check out existing project from Version Control”.
- Klik op “Git”.
- Vul de URL van de repository in (format: https://github.com/”Github-gebruikersnaam”/”repositorynaam”.
- Kies een bestandslocatie en naam voor de map voor het gekloonde project aan te maken.
- Klik op “Clone”.

Het nieuw aangemaakt lokaal project is een kopie van het project op de remote repository en kan naar believen aangepast en doorgestuurd worden naar de remote repository.

CONNECTEREN VIA GITBASH

Voor een Clone van een repository met Gitbash moeten de volgende stappen ondernomen worden:

- Navigeer met cd naar de locatie waar het project aangemaakt moet worden

```
$ cd c:/foldernaam
```

- Voer “git clone” uit naar de URL van de remote repository en geef eventueel een alternatieve mapnaam mee.

```
$ git clone https://github.com/github-gebruikersnaam/repository-naam mapnaam
```

Het aanpassen en opslaan van de code op een Github-repository

Als de lokale repository aan de remote repository is gelinkt kunnen collaborators het project aanpassen. Hier zijn voor een toevoeging vier stappen en een aanpassing drie stappen nodig. Voor de uitleg van deze stappen in Android Studio en Gitbash, komt er eerst een korte uitleg van de globale concepten “Add”,”Commit”, “Pull”, “Push”.

Add

Met een Add-commando wordt een bestand in een project aan de lokale repository toegevoegd. Iedere verandering aan een bestand na een Add wordt bijgehouden in een Changelist, een lijst met alle veranderingen na een Commit of Pull. Als er een bestand wordt toegevoegd aan het project dat niet enkel voor lokaal gebruik is, moet dit met het Add-commando toegevoegd worden aan de lokale repository. Anders zal er met dit bestand geen rekening gehouden worden bij de synchronisatie.

Commit

Een Commit is het (selectief) opslaan van de veranderingen in de bestanden in een lokale repository. Deze veranderingen zullen bij de volgende opwaartse synchronisatie (lokaal→remote) doorgestuurd worden. Deze stap is vereist voor de opwaartse synchronisatie uit te voeren. Het is belangrijk om bij elke Commit ook een boodschap over de aard van de veranderingen bij te voegen

Pull

Dit is een neerwaartse synchronisatie (remote→lokaal). Alle veranderingen op de remote repository worden afgehaald en toegepast op de bijhorende bestanden in de lokale repository. Indien er een overlapping van persoonlijke veranderingen en veranderingen op de remote repository plaatsvindt, kan er een Merge conflict gebeuren (Zie een Conflicten bij een Merge). Veranderingen die voor een Pull niet met een Commit opgeslagen zijn, worden overschreven.

Push

Dit is de opwaartse synchronisatie (lokaal→remote). Hierbij worden alle veranderingen die in een Commit opgeslagen zijn, toegepast op de remote repository. De voorheen lokale veranderingen zijn nu beschikbaar op de remote repository. Het is mogelijke om de veranderingen van meerdere Commits in een keer te synchroniseren.

Een heel belangrijke ongeschreven regel van Github is dat de bovenstaande concepten in deze volgorde worden uitgevoerd:

(Add) → Commit → Pull → (Merge) →Push

De stappen die tussen haakjes staan moeten niet alijd uitgevoerd worden. Add dient enkel uitgevoerd te worden bij het aanmaken van een bestand. Een Merge moet alleen uitgevoerd worden indien er een overlapping tussen veranderingen in de code in een lokaal bestand en de code op hetzelfde bestand op de remote is. Om een Merge te vermijden is het best om afspraken te maken wie in welke bestanden van een project werkt.

ADD, COMMIT, PULL EN PUSH MET ANDROID STUDIO

Add

- Maak een nieuw bestand/nieuwe bestanden aan in het project.
- Er verschijnt een pop-up menu “Add file(s) to git”
- Indien er meerdere bestanden tegelijk worden aangemaakt: vink de bestanden aan die toegevoegd moeten worden aan de lokale repository
- Klik op “Add”

De bestand(en) staan nu in de lokale repository.

Commit

- Onder “VCS”→ Klik op “Commit…”
- Selecteer in het pop-menu de bestanden waar je de veranderingen van wil doorsturen.
- Vul onder “Commit message” een omschrijving in van de geselecteerde veranderingen voor deze Commit.
- Druk op “Commit”
    - Indien er fouten of waarschuwingen zijn verschijnt er een pop-up die de aantallen hiervan zegt. Hier zijn drie knoppen: “review” geeft een overzicht van waar de fouten en meldingen zich bevinden. “Commit anyway” voert de Commit met de fouten en meldingen uit en “Cancel” breekt de commit af.

Het is mogelijk om meerdere Commits tegelijk klaar te hebben staan. Dit kan voor een beter overzicht zorgen van welke veranderingen er precies gebeurd zijn. Ook kan dit helpen om een foute Commit makkelijker te overschrijven of terug te draaien (Zie "Enkele behulpzame commando's voor Gitbash"). De laatst uitgevoerde Commit op een bestand zal het resultaat zijn voor de Push naar de remote repository.

Pull

- Onder "VCS" → "Git" → klik op "Pull…"
- In het pop-menu, klik op de knop "Pull"
- Indien er zich een Merge conflict voordoet, zie "Conflicten bij een Merge"

Push

- Onder "VCS" → "Git" → klik op "Push…"
- In het pop-menu, selecteer de Commit(s) die doorgestuurd moeten worden naar de remote repository.
- Druk op "Push".

ADD, COMMIT,PULL EN PUSH MET GITBASH

Alle commando's moeten uitgevoerd worden in Gitbash op de locatie van het project. Het navigeren naar het project kan met het "cd" commando

```
$ cd c:/foldernaam
```

Add

- Voer het "git add" commando met vermelding van het pad naar het toe te voegen bestand in het project.
    - Om bestanden in het project te vinden die niet in de lokale repository zitten kan gebruik gemaakt worden van "git status" zoals omschreven in "Enkele behulpzame commando's voor Gitbash".
    - Bestanden kunnen geautomatiseerd toegevoegd worden bij een commit, maar er alle bestanden in het project die niet in de repository zitten worden dan automatisch bijgevoegd.

```
$ git add app/src/main/java/com/example/tester/test.java
```

Commit

- Voer het "git commit" commando uit met een boodschap.

```
$ git commit -m "boodschap"
```

    - Commit met automatisch toevoegen en boodschap

```
$ git commit -a -m "boodschap"
```

    - Commit van een specifieke file met boodschap

```
$ git add app/src/main/java/com/example/tester/test.java -m "boodschap"
```

Doe na elke Commit een statuscheck met “git status” om zeker te zijn dat de Commit goed verlopen is (zie “Enkele behulpzame commando’s voor Gitbash”).

Pull

- Voer het “git pull” commando uit

```
$ git pull
```

Push

- Voer het “git push” commando uit

```
$ git push
```

Conflicten bij een Merge

Indien er geen overlappingen zijn van veranderingen in bestanden bij een Pull zal de synchronisatie automatisch deze veranderingen toepassen in de lokale repository. Dit heet een Merge.

Indien er bij een Pull veranderingen binnenkomen die lokale veranderingen in een niet gesynchroniseerde Commit overlappen gebeurt er een Merge conflict. Het samenvoegen van de lokale code met de code die van de remote repository moet nu handmatig gebeuren.

MERGE CONFLICT OPLOSSEN MET ANDROID STUDIO

Indien er zich een Merge conflict voordoet zal een pop-up “conflict” verschijnen met links een selecteerlijst van bestanden met Merge conflicten en rechts vier knoppen:

- “Accept Yours” negeert alle overlappende veranderingen in het geselecteerd bestand die van de remote repository komen en neemt de lokale veranderingen als de juiste.
- “Accept Theirs” negeert alle lokale veranderingen in het geselecteerde bestand en overschrijft ze met veranderingen op de remote repository
- “Merge…” opent een nieuwe pop-up “Merge Revisions for ’bestandsnaam’” voor het geselecteerde bestand.
- “Close” sluit de dialoog, maar zorgt later voor problemen aangezien het Merge Conflict nog steeds bestaat.

Bij het klikken op “Merge…” verschijnt de pop-up voor het handmatig samen voegen van de twee versies van het bestand (lokaal en remote). Links staat de lokale versie, rechts staat de remote versie, in het midden staat de samengevoegde versie.

De gemarkeerde lijnen in het pop-up venster, zijn de lijnen waarop een conflict gedetecteerd werd. Door het klikken op de pijltjes en kruisjes tussen de drie bestandsvensters kun je het samengevoegde bestand aanpassen. Verkeerde aanpassingen kunnen ongedaan gemaakt worden met “Ctrl+z”.

Voor de Merge succesvol uit te voeren zijn er drie opties:

- Het middelste bestand aanpassen tot er een compromis is bereikt bij elk conflict en dan rechtsonder op “Apply” te klikken.
- De remote veranderingen toch nog verwerpen door linksonder op “Accept left te klikken”.
- De lokale veranderingen toch nog verwerpen door linksonder op “Accept right te klikken”.

Indien op “Abort” geklikt wordt, of de dialoog gesloten wordt, zal de Merge niet gebeuren en komen er hierdoor later problemen omdat het Merge conflict nog steeds bestaat.

MERGE CONFLICT OPLOSSEN MET GITBASH

Indien er zich een Merge conflict voordoet zal bij de meldingen van “git push” het volgende staan.

```
Auto-merging app/src/main/java/com/example/tester/test.java
CONFLICT (content): Merge conflict in app/src/main/java/com/example/tester/test.java
Automatic merge failed; fix conflicts and then commit the result.
```

Om dit op te lossen moeten de volgende stappen ondernomen worden.

- “git status” om te kijken welke bestanden een conflict hebben (Meer uitleg bij “Enkele behulpzame Commando’s voor Gitbash”)

```
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 3 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add ..." to mark resolution)

        both modified:  map/project/bestanden/mergeconflict

no changes added to commit (use "git add" and/or "git commit -a")
```

- Het bestand openen in een editor
    - Conflicten worden in dit bestand omringd door “<<<<” en “>>>>>”
    - De lokale veranderingen en de remote veranderingen worden gescheiden door “==========” met de lokale veranderingen boven en de remote veranderingen onder
- Het bestand herschrijven tot alle conflicten zijn weggewerkt.
- Het bestand opnieuw toevoegen aan de lokale repository met “git add”.

```
$ git add map/project/bestanden/mergeconflict
```

- Het bestand in een Commit zetten.

```
$ git commit map/project/bestanden/mergeconflict -m boodschap
```

Nu is de Merge compleet en kan er verder gewerkt worden.

Enkele behulpzame commando’s voor Gitbash

In dit deel worden een paar commando’s toegelicht die Gitbash overzichtelijk houden en lokale fouten met Commits mogelijk kunnen oplossen. Ook deze commando’s moeten op de project locatie uitgevoerd worden.

Status

Bij het uitvoeren van het “git status” commando krijg je een overzicht van nieuwe bestanden en veranderingen in het project.

Onder “Changes to be committed” komen de paden te staan van nieuwe bestanden en bestanden met veranderingen. Deze files zullen na een Commit en een push opwaarts gesynchroniseerd worden met de database. Bestanden die veranderd zijn na een vorig Commit zonder push staan ook onder deze header. Dit komt omdat Commit alleen reeds gedane veranderingen kan klaarzetten voor de Push, en dus geen rekening kan houden met nieuwe veranderingen aan een bestand.

Bij “Unmerged paths” staan de bestanden waar zich een Merge conflict voordeed die nog niet zijn opgelost.

De bestanden onder “Untracked files” zijn bestanden in het project die nog niet in de lokale repository zitten. Zij zullen dus na een Commit en Push niet opwaarts gesynchroniseerd worden met de remote repository.

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD ..." to unstage)

        new file:   app/src/main/java/com/example/tester/testers.java

Unmerged paths:
  (use "git add ..." to mark resolution)

        both modified:   app/src/main/java/com/example/tester/test.java

Untracked files:
  (use "git add ..." to include in what will be committed)

        .idea/.name
```

Kijk na elke Commit na of elk gewenst bestand niet meer voorkomt bij “git status”.

Reflog, log

Met het “git reflog” commando krijg je een overzicht van alle uitgevoerde Commits, Resets en Reverts die reeds gedaan zijn op de lokale repository. Dit commando geeft de nodige informatie voor een “git revert” en “git reset” die verderop beschreven zijn.

- “git reflog” toont een verkort referentie nummer, het aantal Commits verwijderd van de HEAD (laatste Commit) en de Commit boodschap.

```
$ git reflog
b929ad2 (HEAD -> master) HEAD@{0}: commit: boodschap4
1cb0bbb (origin/master, origin/HEAD) HEAD@{1}: commit: boodschap3
655a32d HEAD@{2}: commit: boodschap2
c134c6d HEAD@{3}: commit: boodschap1
```

Met “git log” krijg je alleen maar informatie te zien van alle Commits die nog van toepassing zijn op de lokale repository. Dit wil zeggen dat bij een Reset de Commits die ongedaan gemaakt worden ook uit deze lijst verwijderd worden.

- “git log” geeft iets meer uitleg over de Commits, zoals de datum en de auteur alsook het volledige referentienummer van de Commit.

```
$ git log
commit b929ad2b7dc251e5828ed40d5b4798e102d1554d (HEAD -> master)
Author: kernino
Date:   Thu Jan 9 15:09:46 2020 +0100

    Boodschap2

commit 1cb0bbbb59999c328f6f2042c86a4777568f00ca (origin/master, origin/HEAD)
Author: kernino
Date:   Thu Jan 9 15:06:14 2020 +0100

    Boodschap
```

Zoals je dus kan is het belangrijk om duidelijke boodschappen bij Commits te zetten zodat deze later makkelijk terug te vinden zijn.

Reset

Met het “git reset” commando kun je het project terugzetten naar een oudere Commit indien er een fout in het project is geslopen.

Dit doet alle veranderingen die na deze Commit zijn gebeurd te niet, zowel in Commits als in het project. Dit kan niet ongedaan gemaakt worden.

Je kan op twee manieren refereren naar een Commit.

- Voor een “git revert” met (verkort) referentie nummer, zoek je het referentienummer op met “git reflog” of “git log”.
    - Voor een Reset naar een Commit met referentie “171ceb6”→

```
$ git reset 171ceb6
```

- Voor een “git revert” met verwijzing naar de laatste Commit, zoek je het aantal stappen terug vanaf de HEAD (laatste Commit) op met “git reflog”.
    - Reset van een Commit op vijf stappen van de HEAD

```
$ git reset HEAD@{5}
```

Revert

Met “git revert” kan een specifiek Commit ongedaan gemaakt worden. Hiervoor wordt er een Commit aangemaakt die exact het tegenovergestelde is van de Commit die een Revert ondergaat. Makkelijker gezegd worden de veranderingen in een Commit terug veranderd.

- Zoek het referentienummer van de Commit die ongedaan gemaakt moet worden op met “git reflog”.
    - Revert van een Commit met referentie “171ceb6”

```
$ git revert 171ceb6
```

Besluit

Hoewel dit verslag niet de werking van git en Github ten gronde uitlegt, geeft het wel een basis voor mensen die pas beginnen met Github.

Dit is echter nog maar het begin van de functies die git en Github ter beschikking stellen. Mensen die de concepten in dit verslag onder de knie hebben kunnen de verdere functies ontdekken zoals bijvoorbeeld Branching dat het mogelijk maakt om meerdere versies van hetzelfde project te werken en deze later samen te voegen.

Git, Github en andere versiecontrolesystemen zijn een noodzaak in de moderne programmeerwereld. Indien men zich hier niet in wil verdiepen is het toch belangrijk om alvast de basis in dit verslag te kennen.

SQFlite

Wat is SQFlite

Er wordt steeds meer en meer met Cloud data gewerkt in software development. Omdat deze data ook offline gebruikt wordt, wordt deze opgeslagen in een lokale database. Omdat wij met Flutter een cross platform app maken, maken wij gebruik van de SQFlite database.

SQFlite is een plug-in speciaal gemaakt voor Flutter en die gebaseerd is op SQLite. Het voordeel van SQFlite is dat dit beide platforms ondersteund en dat het ook alle voordelen van de gewone SQLite heeft. Deze plug-in is ontworpen door takrtik.com.

SQFlite installeren

Om met SQFlite te werken moet er in het flutter project een dependency toegevoegd worden. Dit doe je door naar het build.gradle bestand te gaan van het project en hier de volgende lijn toe te voegen.

```
dependencies:
  ...
  sqflite: ^1.1.8
```

Als de dependency is toegevoegd moet er in het bestand waar SQFlite wordt gebruikt een import worden gedaan naar het package.

```
import 'package:sqflite/sqflite.dart';
```

SQFlite gebruiken

Database openen / sluiten

Een SQFlite database is een bestand opgeslagen op de gsm. Het bestand is bij Android te vinden in de default database map en bij iOS is dit te vinden in de documenten map. Om deze te open kan je de openDatabase(String database) methode gebruiken.

```
var db = await openDatabase('my_db.db');
```

In een normale app wordt er meestal maar met één lokale database gewerkt en is er dus geen nood aan het sluiten van de database. Moest je toch meerdere databases hebben kan je deze ook sluiten.

```
await db.close();
```

Raw SQL queries

Transacties maken

Net zoals bij een gewone SQL-database kan je met SQFlite ook transacties maken. Dit doe je door de transaction methode van het database object te gebruiken. Hierna kan je al de queries uitvoeren na elkaar.

```
// Insert some records in a transaction
await database.transaction((txn) async {
  int id1 = await txn.rawInsert(
      'INSERT INTO Test(name, value, num) VALUES("some name", 1234,
456.789)');
  print('inserted1: $id1');
  int id2 = await txn.rawInsert(
      'INSERT INTO Test(name, value, num) VALUES(?, ?, ?)',
      ['another name', 12345678, 3.1416]);
  print('inserted2: $id2');
});
```

Records updaten

Een update query voer je hetzelfde uit als in een gewone SQL database. Hierbij gebruik je de rawUpdate methode waarbij je als eerste argument de query meegeeft met ? als waardes van variabelen en als 2de argument de waardes van de variabelen.

```
// Update some record
int count = await database.rawUpdate(
    'UPDATE Test SET name = ?, VALUE = ? WHERE name = ?',
    ['updated name', '9876', 'some name']);
print('updated: $count');
```

Records opvragen

Een record opvragen gaat met de methode rawQuery waarmee je de query meegeeft. In de meeste gevallen is dit een SELECT query.

```
// Get the records
List<Map> list = await database.rawQuery('SELECT * FROM Test');
List<Map> expectedList = [
{'name': 'updated name', 'id': 1, 'value': 9876, 'num': 456.789},
{'name': 'another name', 'id': 2, 'value': 12345678, 'num': 3.1416}
];
print(list);
print(expectedList);
assert(const DeepCollectionEquality().equals(list, expectedList));
```

Records deleten

Records deleten werkt hetzelfde als deze te updaten. Enkel ga je nu in plaats van de rawUpdate methode de rawDelete methode uitvoeren.

```
count = await database
    .rawDelete('DELETE FROM Test WHERE name = ?', ['another name']);
assert(count == 1);
```

SQL datatypes

Integer

- Dart type: `int`
- Supported values: from -2^63 to 2^63 – 1

REAL

- Dart type: `num`

TEXT

- Dart type: `String`

BLOB

- Dart type: `Uint8List`
- Dart type `List<int>` is supported but not recommended (slow conversion)

SQFlite in ons project

Voor het makkelijk te maken zullen we een klein voorbeeld geven van elke toepassing

1:Tabellen maken:

```
db = await openDatabase(
  join(await getDatabasesPath(), 'data.reizentechnogie.db'),
  onCreate: (db, version) async {
    await db.execute(
      "CREATE TABLE trips ("
          "id INTEGER PRIMARY KEY,"
          "name TEXT,"
          "start_date TEXT,"
          "end_date TEXT,"
          "destination TEXT,"
          "transportation_info TEXT)"
    );
```

2: We kunnen seeden voor als we vaste data manueel in de database willen zetten

```
var dayPlanning1 = DayPlanning(
    id: 1,
    date: '17/05',
    highlight: 'vertrek naar ...',
    description: 'descriptionTest',
    location: 'Location1');


db.insert("day_planning", dayPlanning1.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace);
```

3: We hebben ook CRUD in SQFlite met Queries :

a) Create (insert)

```
db.insert("day_planning", dayPlanning1.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace);
```

b) Read (by id of all)

```
activitiesData = await globals.database.query('plannings', where: '"day_id" = ?', whereArgs: [id]);
```

of

```
dayPlannings = await globals.database.query("days");
```

c) Update

```
await globals.dbHelper.db.rawUpdate(
    "UPDATE remote_update SET update_time = ? WHERE id = ?",
    [result.data['latest'], dbDate[0]["id"]]);
```

d) Delete

```
await globals.dbHelper.db.rawDelete("DELETE FROM travellers");
```

GraphQL

Wat is GraphQL

GraphQL is eigenlijk twee dingen, enerzijds een query language voor Api's en anderzijds een runtime engine om de query's te verwerken. Bij GraphQl is het zo dat er een query naar de server wordt gestuurd. In deze query worden de velden opgegeven waar de eindgebruiker iets mee wil doen. Dit is meteen het voordeel van GraphQL t.o.v. een traditionele Api, er wordt enkel data gestuurd waar de eindgebruiker iets mee is i.p.v. alle data. Hierdoor verloopt de communicatie sneller, zelfs over trage mobiele connecties. De hoeveelheid data is ook beperkt omdat de cliënt bepaalt welke data hij wil en niet de server. Verder heeft een uitbreiding van de velden geen invloed op de gegevens die de client vraagt.

GraphQL gebruiken

Serverside

Er zijn verschillende serverside programmeertalen waarop GraphQL geïnstalleerd kan worden. PHP, Java, Python, C# / .Net zijn hier een aantal van maar er zijn meer programmeer talen beschikbaar die GraphQL ondersteunen.

Clientside

Ook voor clients zijn er verschillende talen waarbij GraphQL geïnstalleerd kan worden. Het kan bv. Gebruikt worden met Android/Java, Javascript en C# / .Net, ook dit is maar een deel van de programmeertalen die de client ondersteunen, er zijn nog andere talen die het ook kunnen.

Wij gaan GraphQL gebruiken om api-requests op te doen vanuit de nieuwe reizentechnologie-App naar de reizentechnologie-website die op een server draait. De website is geschreven in het php-framework Laravel. We gaan de bestaande website (in Laravel) verder uitbreiden en GraphQL hierin implementeren. De App wordt geschreven in Flutter, om GraphQL te kunnen gebruiken zullen we in Flutter een library moeten gebruiken om de request te sturen en daarna te parsen.

GraphQL op de server

De website voor het beheer van de reizen is geschreven in het Laravel, dit is een php-framework dat gebruik maakt van een MVC-structuur. Als we op internet gaan zoeken hoe we GraphQl moeten implementeren in Laravel, dan vinden we meerdere softwarepakketten terug die dit kunnen. Hieronder worden een aantal pakketten opgesomd die je kan gebruiken met het Laravel-Framework.

- nuwave/lighthouse
- rebing/*graphql-laravel*
- ardani/*laravel-graphql*
- thecodingmachine/*graphqlite*
- mll-lab/*laravel-graphql*-playground
- ...

Zoals je ziet zijn er al veel mensen die een graphql integratie hebben gemaakt voor Laravel, we gaan dus zeker niet opnieuw het warm water uitvinden. In ons project hebben we twee pakketten geïntegreerd in Laravel. Als eerste hebben we geprobeerd om het softwarepakket van

“rebing/graphql-laravel” te installeren, deze installatie gaat relatief goed alleen is het gebruik wat omslachtig. In een later stadium hebben we dan een tweede softwarepakket bij geïnstalleerd, namelijk: “nuwave/lighthouse”. Deze laatste is in installatie en gebruik zelfs gemakkelijker. Beide installatie zijn volledig op Github te vinden maar worden verder ook in dit document uitgeschreven.

In GraphQL zijn er verschillende onderdelen nodig die er samen voor zorgen dat je GraphQL kan gebruiken. De eerste, de query, is eigenlijk de entry point van GraphQL voor het opvragen van data. Het type daarentegen bepaald welke velden er zijn die je kan opvragen.

Voor beide pakketten gaan er gelijkenissen en verschillen zijn. Voor beide systemen gaan we queries en types moeten aanmaken. Verderop zullen we zien dat bij “nuwave/lighthouse” het heel eenvoudig is om een querie of type aan te maken en dat dit bij “rebing/graphql-laravel” een iets grotere klus is.

Nuwave/lighthouse

Als eerste zullen we de installatie van graphql met “Nuwave/lighthouse” uit de doeken doen aangezien deze het gemakkelijkste is. Voordat we kunnen beginnen moeten we eerst een Laravel project hebben en moet composer geïnstalleerd zijn. Hoe je composer installeert en een Laravel project maakt kan je terugvinden op de website van Laravel zelf.

Installatie

Als eerste moeten we “Nuwave/lighthouse” installeren in ons Laravel project. Hiervoor gaan we met een terminal navigeren tot in de project map.

```
// bv. In een Linux Ubuntu installatie

cd /var/www/<<laravel-project>>/

composer require nuwave/lighthouse
```

Vervolgens moeten we de serviceprovider van lighthouse publiceren. Dit bestaat uit twee stukken, als eerste het configuratie bestand en als tweede het schema. Beide bestande maken we als volgt aan:

```
// voor het configuratie bestand

php artisan vendor:publish --provider="Nuwave\Lighthouse\LighthouseServiceProvider" --tag=config

// voor het schema

php artisan vendor:publish --provider="Nuwave\Lighthouse\LighthouseServiceProvider" --
tag=schema
```

Het configuratie bestand kan je dan terugvinden in de “config” map van je Laravel project. Het heeft de naam “lighthouse.php” gekregen. Het schema vind je in de map “graphql” en heeft als naam “schema.graphql”.

Querie & type

Zowel de query als het type worden in het “schema.graphql” bestand uitgeschreven. Voor Lighthouse is de query eigenlijk ook een type, zie als volgt:

```
type Query {
        me: User @auth
        Roles: [Role!]!
}
```

Een normaal type ziet er dan als volgt uit:

```
type User {
        Name: String!
        Email: String
        Role: String!
}
Type Role {
        Name: String!
        Users: [User!]
}
```

Zoals jullie zien is er praktisch geen verschil tussen het query type en een ander type. Lighthouse gaat zelf een aantal types standaard voor zijn rekening nemen, “Query”, “Mutation” zijn hier voorbeelden van. Al de overige types gaat lighthouse ‘koppelen’ aan de modellen in ons project. Het type “User” wordt dus gekoppeld aan het model “User”. Lighthouse (en graphql dus ook) kan alleen velden opvragen die ook in ons model zitten.

Zowel in de query, als de types zijn we een aantal speciale tekens voorkomen. De “[ ]” rond een type wijst erop dat we een lijst van dit type verwachten, in dit voorbeeld verwachten we een lijst met alle rollen (en per rol een lijst van alle gebruikers met die rol). Een “!” wijst erop dat het type niet nullable kan zijn.

Als laatste kunnen we, inden dit gewenst is, de URL nog aanpassen waarnaar we onze query moeten sturen. Dit gebeurt in het configuratie bestand van lighthouse, dit is te vinden in het bestand config/lighthouse.php. Onder de configuratie van de “route” vinden we de “Uri” en “name” terug. Beide moeten aangepast worden naar de gewenste Uri.

Aangezien dat eerste de versie van rebing/graphql geïnstalleerd was hebben we deze URL dus moeten aanpassen. In ons voorbeeld werd de Uri “graphql2” i.p.v. “graphql”, dit moet zowel bij “Uri” als bij “name” aangepast worden.

Ervan uitgaande dat de modellen “User” en “Role” bestaan en dat er correcte dat in de database zitten, kunnen we dit gaan testen. Hiervoor schrijven we een query naar de server. De URL waarnaar we de query sturen is (bv. Binnen een testomgeving op localhost) “http://localhost/graphql2?query=query+Fetch{users{name, email}}”. Je kan dit uittesten met je browser, sommige browsers kunnen overweg met json data en gaan dit op een mooie manier weergeven. Een ander alternatief is bv. Postman, dit is een platform waarmee je api’s kan uittesten. Daar waar je bij een browser enkel een GET request kan maken, kan je met programma’s gelijk Postman ook POST, PUT, DELETE, … requesten maken. Verder laten ze ook toe om een vorm van authenticatie toe te passen.

Er bestaan ook playgrounds voor het gebruik van graphql bv. mll-lab/laravel-graphql-playground.

Op moment van schrijven is de huidige versie van lighthouse versie 4.7. Wanneer er nieuwere versies van lighthouse uitkomen zou het kunnen dat dit document niet meer juist is. Raadpleeg best de website of github pagina van lighthouse voor de laatste versie van lighthouse.

Rebing/graphql-laravel

“rebing/graphql-laravel” is een andere GraphQl engine die we kunnen implementeren in laravel. Het gebruik van deze engine is iets moeilijker aangezien alle types en queries een eigen bestand hebben en ze geregistreerd moeten worden. Als je de installatie van “Nuwave/lighthouse” hebt gevolgd dan heb je al een laravel project, je kan deze versie van graphql daar gewoon bij installeren. Als je dit nog niet hebt gedaan moet je eerst zorgen dat composer geïnstalleerd is en dat je een laravel project aanmaakt. Hoe je dit doet kan je terugvinden op de website van Laravel.

Installatie

Als eerste moeten we met de composer package manager de “Rebing/graphql-laravel” package toevoegen in installeren in ons project. Hiervoor navigeren we eerst naar onze project map.

```
// bv. In een Linux Ubuntu installatie

cd /var/www/<<laravel-project>>/

composer require rebing/graphql-laravel
```

Daarna moeten we het configuratie bestand nog beschikbaar maken. Bij deze versie van graphql is er geen appart schema nodig gelijk dit is bij “Nuwave/lighthouse”. Het “schema” wordt mee in het configuratie bestand opgenomen.

```
php artisan vendor:publish --provider="Rebing\GraphQL\GraphQLServiceProvider"
```

Je kan nu het configuratie bestand terugvinden in de map config, het heeft de naam “graphql.php”.

Queries en types

Nu we de installatie volbracht hebben kunnen verder gaan met het maken van queries en types. Daar waar we bij “Nuwave/lighthouse” in het schema gewoon moeten zeggen hoe een type en een query eruit ziet, moeten we deze bij de “rebing/graphql-laravel” eerst nog zelf aanmaken.

Query

Net als alle andere dingen binnen Laravel moeten we een query aanmaken met artisan. Dit doen we met:

```
php artisan make:graphql:query <<query_name>> // bv. UserQuery
```

Nu wordt een query aangemaakt in de map “/app/GraphQL/Queries”, de bestandsnaam is “<<query_name>>.php”. Wanneer we in het query bestand gaan kijken zien we drie functies terugkomen, een type, een args en een resolve.

De functie type bepaald wat de query zal terugsturen. Dit kan een gewoon GraphQl type zijn, of een Type lijst van een type. Hierbij kan je elk type dat je al hebt aangemaakt, of je kan een nieuw type kiezen maar dan moet je het hierna nog wel aanmaken.

De args functie bevat een array van alle mogelijk argumenten die we moet een query kunnen meesturen. Hierbij moeten we de naam van het argument opgeven en het type dat we verwachten. Het type kan een string, int, boolean, … zijn.

De laatste functie is de resolve. Hierin wordt de query effectief afgehandeld. Als je argumenten gebruikt ga je deze op hier implementeren. Je kan dus op modellen een “where” gaan toepassen van één of meerdere argumenten, het resultaat hiervan zal dan via graphql teruggestuurd worden. Bij het gebruik van parameters voorzie je best ook een return waarde voor als er geen parameter is opgegeven. Zo vermijd je dat er exceptions worden geworpen.

```php
class InfoQuery extends Query
{
  protected $attributes = [
    'name' => 'info',
    'description' => 'A query to retrieve information'
  ];

  public function type(): Type
  {
    return Type::listOf(GraphQL::type('info'));
  }

  public function args(): array
  {
    return [
      'info_name' => ['name' => 'info_name', 'type' => Type::string()],
    ];
  }

  public function resolve($root, $args, $context, ResolveInfo
$resolveInfo, Closure $getSelectFields)
  {
    if (isset($args['info_name'])) {
      return Information::where('info_name', "=",
$args['info_name'])->get();
    }
    return Information::all();
  }
}
```

Type

Op dezelfde manier als de query maken we types ook aan met artisan:

php artisan make:graphql:type <<type_name>> // bv. UserType

Types worden aangemaakt in de map “/app/GraphQL/Types”, de bestandsnaam is “<<type_name>>.php”. Als je het bestand gaat bekijken zie je een protected array en een functie fields.

In de protected array vinden we standaard twee elementen namelijk ‘name’ en ‘description’. De naam geef je best dezelfde naam als de klasse naam (zonder de ‘Type’ eraan) bv. User. Best geef je ook nog een extra element mee, namelijk ‘model’. Hiermee geef je aan welke model er bij het type hoort, aangezien we met het UserType bezig zijn is het model “User::class”. Dit zorgt er ook voor dat Graphql in staat is om relaties te volgen binnen de verschillende types. Stel dat je bv. Van alle users wil weten welke vakken ze volgen. Dan voorzie je in het usertype een lijst van het type ‘course’. Graphql vindt dan de relatie terug omdat deze normaal gezien is uitgeschreven in de modellen.

In de functie fieds wordt ook een array opgebouwd. Elke element is een field dat we via GraphQl kunnen opvragen. Voor elk element is er opnieuw een array met daarin het ‘type’ en ‘description’. Het type kan een string, int, boolean, ... zijn maar ook een ander GraphQl type.

```
class TripType extends GraphQLType
{
  protected $attributes = [
    'name' => 'Trip',
    'description' => 'A type',
    'model' => Trip::class,
  ];

  public function fields(): array
  {
    return [
      'name' => [
        'type' => Type::string(),
        'description' => 'trip name',
      ],
      'hotels' => [
        'type' => Type::listOf(GraphQL::type('hotel')),
        'description' => 'a list of all hotels on this trip',
      ],
      'travellers' => [
        'type' => Type::listOf(GraphQL::type('traveller')),
        'description' => 'a list of all travellers on this trip',
      ],
      'day_plannings' =>[
        'type' => Type::listof(GraphQl::type('day_planning')),
        'description' => 'a list of all the days on this trip',
      ],
      'transport' => [
        'type' => Type::listOf(GraphQL::type('transport')),
        'description' => 'a list of all cars/busses on this trip',
      ],
      'emergency_numbers' =>[
        'type' => Type::listOf(GraphQL::type('emergency_number')),
        'description' => 'a list of all the emergency numbers on this
trip',
      ],
    ];
  }
}
```

Config

Nu we de query en het type hebben aangemaakt moeten we deze nog registeren in het package. Dit gebeurt in “config/graphql.php”. Hierin staat ook alles weer weergegeven als een array. Hierin vinden we ook een schema terug, standaard staat hier al een default schema in. Je kan hier perfect in verder werken. Je vindt hierin de queries, mutations maar ook de middleware en methodes die moeten gebruikt worden. De nieuw gemaakte query gaan we toevoegen in de array van queries.

Iets daaronder zien we opnieuw een array komen met daarin alle types. Het nieuw gemaakte type moet hier toegevoegd worden. Pas wanneer alle types en queries zijn toegevoegd kunnen we ze ook gebruiken.

Ook vind je in dit bestand de prefix die gebruikt moet worden in de route naar deze package van graphql. De prefix is hetzelfde als de Uri die we geconfigureerd hebben in “Nuwave/lighthouse”. Als je de installatie van “Nuwave/lighthouse” hebt gevolgd dan zou daar de Uri “graphql2” zijn en moet je aan deze prefix (gelijk het hier genoemd wordt) niets moeten veranderen.

Je kan dit nu op dezelfde manier gaan testen als we hiervoor gedaan hebben. Voor de eenvoudige GET requests kan je de browser gebruiken, maar je kan net zo goed programma’s zoals postman gebruiken. Deze zijn speciaal gemaakt om api’s te testen. Als je echt iets speciaal zoekt voor het testen van graphql kan je bv. mll-lab/laravel-graphql-playground gebruiken.

Ook hier is het weer zo dat de uitleg die hier staat na verloop van tijd out-dated is. Je kan dus best de stappen volgen gelijk ze beschreven staan op de github pagina van rebing.

GraphQL op de client

De app waar we al de informatie over de reizen kunnen raadplegen is gemaakt met Flutter en Dart. Hier gaan we bij dit hoofdstuk niet te ver op ingaan omdat dit al ergens anders wordt besproken. Om graphql te kunnen gebruiken in Flutter maken we gebruik van de library “graphql_flutter 2.1.0”. Hiermee kunnen we via aan api-request de benodigde data ophalen voor onze app.

Installatie

Om de graphql library te installeren op flutter moeten we in het “pubspec.yaml” bestand een dependecy toevoegen. De “pubspec.yaml” bestand is kort samengevat een bestand waar instaat welke libraries en versies er gebruikt worden in het project. Voeg volgende regel toe bij dependecies:

```
graphql_flutter: ^2.0.0
```

Elk bestand in het project waar u gebruik van wilt maken moet u bovenaan de library importeren zodat de functies en methodes ervan beschikbaar zullen zijn.

```
import 'package:graphql_flutter/graphql_flutter.dart';
```

Gebruik

Om de client te gebruiken, moet deze eerst worden geïnitialiseerd met een link en cache. Voor ons project gebruiken we een HttpLink als onze link en InMemoryCache als onze cache. Als uw eindpunt authenticatie vereist, wat in ons geval zo is, kunt u een AuthLink samenvoegen, het lost de referenties op met behulp van een “future” (een asynchrone functie in Flutter), zodat u asynchroon kunt authentiseren.

Normaal wordt de library zo gebruikt dat alles in één widget terechtkomt maar omdat wij gebruik maken van een MVVM-structuur hebben wij de widget opgedeeld in meerdere functies die zo in onze viewmodel terechtkomen. Maar omdat Flutter eist dat sommige widgets verplicht in één bestand moeten zijn er kleine deeltjes die wel verplicht in de view moeten staan. Als voorbeeld om de werking van graphql in Flutter met MVVM-structuur te demonstreren neem ik het gedeelde waar men de overeenkomsten van de app moet accepteren vooraleer hij/zij de app kan gebruiken.

Gebruik in de view

De eerste widget van de view ziet er als volgt uit:

```
class VoorwaardenConnection extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final ValueNotifier<GraphQLClient> client = ValueNotifier<GraphQLClient>(
      GraphQLClient(
        link: setConnection(),
        cache: OptimisticCache(
          dataIdFromObject: typenameDataIdFromObject,
        ),
      ),
    );
    return GraphQLProvider(
      child: Voorwaarden(),
      client: client,
    );
  }
}
```

Deze widget wordt gebruikt om de connectie met de server op te zetten. Er is een link met een functie die in het viewmodel is geplaatst waar we later dieper in zullen gaan, en er is een cache waar je kan instellen welke soort cache je wilt gebruiken. Maar omdat wij de offline cache niet gebruiken gaan we hier ook niet dieper over ingaan. Het is enerzijds wel verplicht om een cache soort in te vullen hebben we de standaard cache “optimisticCache” genomen sinds deze toch nog mutaties ondersteund voor eventuele uitbreidingen in de toekomst.

De child in de return verwijst naar een statefulWidget die we gaan overslaan omdat deze niet meer informatie geeft over ons gebruik van graphql in Flutter. Als laatste is er een client die verwijst naar een eerder aangemaakte variabele die gewoonweg zegt dat we de graphql client gebruiken.

De tweede widget in de view:

```
class _VoorwaardenState extends State<Voorwaarden> {
  bool _isButtonDisabled = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title:
            Text("Algemene voorwaarden", style: TextStyle(color: Colors.white)),
        backgroundColor: Color.fromRGBO(224, 0, 73, 1.0),
      ),
      body: Scrollbar(
        child: Center(
          child: Padding(
            padding: const EdgeInsets.all(8.0),
            child: SingleChildScrollView(
                child: Column(
              mainAxisSize: MainAxisSize.min,
              mainAxisAlignment: MainAxisAlignment.start,
              children: <Widget>[
                SizedBox(
                    height: MediaQuery.of(context).size.height - 215,
                    child: getData()),
                Column(
                  children: <Widget>[
                    Row(
                      children: <Widget>[
                        Checkbox(
                            value: _isButtonDisabled,
                            onChanged: (bool newValue) {
                              setState(
                                  () => _isButtonDisabled = !_isButtonDisabled);
                            }),
                        Text('Gelezen en goedgekeurd')
                      ],
                    ),
                    RaisedButton(
                      padding: const EdgeInsets.all(10.0),
                      onPressed: !_isButtonDisabled
                          ? null
                          : () {
                              acceptConditions(context);
                            },
                      color: Color.fromRGBO(224, 0, 73, 1.0),
                      textColor: Colors.white,
                      child:
                          Text('Ik ga akkoord', style: TextStyle(fontSize: 20)),
                    ),
                  ],
                )
              ],
            )),
          ),
        ),
      ),
    );
  }
}
```

Deze widget is eigenlijk de echte view waar alle opmaak en dergelijke wordt gedaan maar om de data die we via graphql gaan ophalen hebben we er de in blauw gemarkeerde functie “getData()” in gezet die weer verwijst naar een functie in het viewmodel.

Gebruik in het viewmodel

Nu gaan we in het viewmodel de eerder tegengekomen functies bespreken. De eerste functie is de “setConnection()”.

```
Link setConnection(){
  var bearer = globals.loggedInUser[0]["token"];
  final HttpLink httpLink =  HttpLink(uri:
"http://171.25.229.102:8222/graphql?query=");
  final AuthLink authLink = AuthLink(getToken: () async => 'Bearer ' + bearer);
  final Link link = authLink.concat(httpLink);
  return link;
}
```

Deze functie initialiseert eigenlijk de link zodat deze gebruikt kan worden om de data op te halen. De functie begint met een bearer token op te halen die al eerder verkregen is door in de app in te loggen en er wordt ook een nieuwe HttpLink aangemaakt met de link van de server met erachter “graphql?query=" om tegen de server te zeggen dat er een graphql request binnen zal komen. Hierna wordt de aangemaakte link gekoppeld met de bearer token zodat we toegang krijgen tot de data van de server en wordt de volledige Link teruggegeven aan de widget die in de view staat.

De volgende functie die noodzakelijk is, is de “getData()” functie.

```
Query getData() {
  return Query(
      options: QueryOptions(document: getAlgemeneData()),
      builder: (QueryResult result, {
        BoolCallback refetch,
        FetchMore fetchMore,
      }) {
        if (result.data == null) {
          return Center(child: CircularProgressIndicator());
        }
        return ListView.builder(
          itemBuilder: (BuildContext context, int index) {
            return Text(result.data['info'][index]['info_value']);
          },
          itemCount: result.data['info'].length,
          scrollDirection: Axis.vertical,
        );
      });
}
```

Deze functie geeft eigenlijk een volledige QueryWidget (omdat flutter altijd met widgets werkt) terug naar de view.
Bij de QueryOptions ziet u een blauwe functie die weer verwijst naar een andere functie in het viewmodel. Die functie is eigenlijk de query zelf die naar de server wordt verstuurd, maar daar zullen we ons dadelijk in verdiepen.
De builder geeft het resultaat van de server terug in de variabele “result” van het type QueryResult wat eigenlijk een soort van array is. Met deze variabele kunnen we zeggen dat wanneer er geen data is of wanneer deze nog niet is binnengekomen kunnen we een laad icoontje teruggeven. Maar wanneer er wel data is wordt er een Listview.builder widget teruggegeven.

Deze widget gaat elke lijn af van de variabele result en hier kunnen we zeggen welke data we moeten hebben. De velden van de array in de variabele result komen overeen met de tabellen en records die de server terugstuurt.

De laatste functie in het viewmodel is de query zelf:

```
String getAlgemeneData() {
  String data ="""
    query{
      info(info_name:"algemene_voorwaarde")
      {
        info_value
      }
    }
    """;
  return data;
}
```

Deze functie geeft gewoonweg een string terug van wat de query zou moeten zijn die naar de server wordt gestuurd.
In dit geval hebben we de info_value nodig die in de tabel info staat met als info_name “algemene_voorwaarden”.

Om de query te verduidelijken geef ik ook nog een andere query opstelling weer:

```
String data ="""
  query{
    info(info_name: "algemene_info")
    {
      info_value
    }
    trip(trip_id: 1) {
      name
      travellers {
        traveller_id
        first_name
        last_name
        phone
      }
  }
"""
```

In deze query worden twee tabellen van de server aangesproken. De ene tabel info die de info_value van de algemene info ophaalt (niet verwarren met de info_value van de vorige query, sinds deze tabel een andere info_name heeft). En een andere tabel trip waar je meegeeft over welke trip het gaat en hier dan de name en de travellers van deze trip opvraagt. Je wil wel niet alle data van de travellers hebben dus vraag je alleen hetgeen op wat je nodig hebt.

Zo zie je maar hoe gemakkelijk het is om data uit meerdere tabellen in één keer op te vragen met graphql en hoe flexibel het kan zijn.

Sprint 1

Startdatum: 01-10-2019

Sprint backlog

User Story	Taken	Requirements/Toelichting
Als gebruiker **wil ik** mij (eenmalig) kunnen aanmelden via mijn inloggegevens indien ik netwerkverbinding heb **zodat ik** algemene voorwaarden kan bevestigen waarna ik toegang krijg tot de app	- SQFlite opzetten - Inloggegevens valideren - Inlogview maken - Voorwaardenview maken - Algemene Voorwaarden downloaden van server (MVM) - Check netwerkstatus (Foutmelding indien geen netwerk , warning bij 4G) - Check inlogstatus (kijken bij SQFlite) - GraphQL opzetten	- Foutmelding zonder netwerk - Verificatie aanmeldstatus - Inloggen met r-nummer en geregistreerd wachtwoord - Aanvaarden voorwaarden (Van server downloaden, GraphQL) pas mogelijk na lezen - Foutmelding bij foute inloggegevens
Als geregistreerde gebruiker **wil ik** alle noodnummers via een klik kunnen raadplegen **zodat ik** in geval van nood de informatie snel ter beschikking heb	- Dropdown met noodnummers - Appbar noodnummer icoon - Instant call	- Knop noodnummers altijd zichtbaar (navbar) - Dropdown list met nummers leerkrachten/personeel - Instant Call door op nummer te klikken

SQFlite opzetten

- Toelichting/analyse
    - Voordat we met data kunnen gaan werken moet er eerst een database komen. Dus in sprint 1 gaan we deze opzetten en een basisconfiguratie toepassen.

- Code
  - Eerst wordt een klasse DatabaseHelper aangemaakt waarin alle code komt die met de database te maken heeft. Hierna gaan we een Database instantie aanmaken db. Als laatste wordt er een asynchrone functie geschreven en openen we een database met als path ‘data.reizentechnologie.db’.
  - ```
    class DatabaseHelper {

      Database db;

      Future initializeDatabase(BuildContext context) async {

        db = await openDatabase(
          join(await getDatabasesPath(), data.reizentechnogie.db'),
          onCreate: (db, version) async { })
    ```

Inloggegevens valideren

- Toelichting
  - Voordat de gebruikers de app kunnen gebruiken moeten ze ingelogd zijn. De inloggegevens worden vanuit de inlogview naar de server gestuurd voor validatie. Op de server worden de gegevens vergeleken met deze in de database. Als alles in orde is geven we een json object terug met daarin de bearer_token, de voor- en achternaam, de traveller- en trip-id.
  - Voor het opvragen van alle traveller gerelateerde info maken we gebruik van de “travellerRepository”.
  - De bearer_token die we terugkrijgen is nodig voor de verdere authenticatie die gaat gebeuren bij de opvragingen met GraphQl.
    ```

- Code back-end:

```
class AuthController extends Controller
{
    private $travellerRepository;


    public function __construct(TravellerRepository $travellerRepository) {
            $this->travellerRepository = $travellerRepository;
    }


    public function authenticate(Request $request)
    {
            $credentials = $request->only('username', 'password');
            if(Auth::attempt($credentials)) {
                    return Array(
                            "status" => "succedded",
                            "api_token" => Auth::user()->api_token,
                            "first_name" => $this->travellerRepository-
                    >getFirstNameByUserId(Auth::id()),
                            "last_name" => $this->travellerRepository-
                    >getLastNameByUserId(Auth::id()),
                            "traveller_id" => $this->travellerRepository-
                    >getTravellerIdByUserId(Auth::id()),
                            "trip_id" => $this->travellerRepository-
                    >getCurrentTripByUserId(Auth::id()),
                    );
            }
            return Array(
                    "status" => "failed",
                    "message" => "username or password wrong"
            );
    }
}
```

- Code front-end:

```
void FetchJSON(String username, String password, BuildContext context)
async {
    username = username.trim();
    if (username == "" || password == ""){
      showAlertDialog(context, "Geen gebruikersnaam en/of paswoord
ingegeven.");
    }
    else{
      String url = 'http://171.25.229.102:8222/api/login';
      Map<String, String> headers = {"Content-type":
"application/json"};
      String json = '{"username":"' + username + '","password":"' +
password + '"}';
      var Response = await http.post(url, headers: headers, body: json);

      if (Response.statusCode == 200) {
        String responseBody = Response.body;
        var responseJSON = jsonDecode(responseBody);

        status = responseJSON['status'];
        message = responseJSON['message'];
        api_token = responseJSON['api_token'];
        first_name = responseJSON['first_name'];
        last_name = responseJSON['last_name'];
        traveller_id = responseJSON['traveller_id'];
        trip_id = responseJSON['trip_id'];

        isData = true;
/*      setState(() {
        print('UI Updated');
      });*/
      } else {
        print('Something went wrong. \nResponse Code :
${Response.statusCode}');
      }
```

```
if (status != null && api_token != null && message == null && first_name
!= null && last_name != null && traveller_id != null){
        //Navigator.of(context).pushReplacementNamed('/screen2');

        var user = User(
            id: 1,
            firstName: first_name,
            lastName: last_name,
            acceptedConditions: 0,
            token: api_token,
            traveller_id: traveller_id,
            trip_id: trip_id);

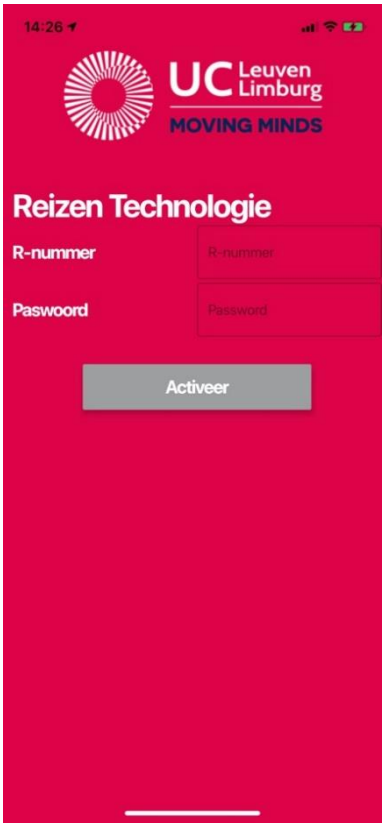


        await globals.dbHelper.db.insert("users", user.toMap());

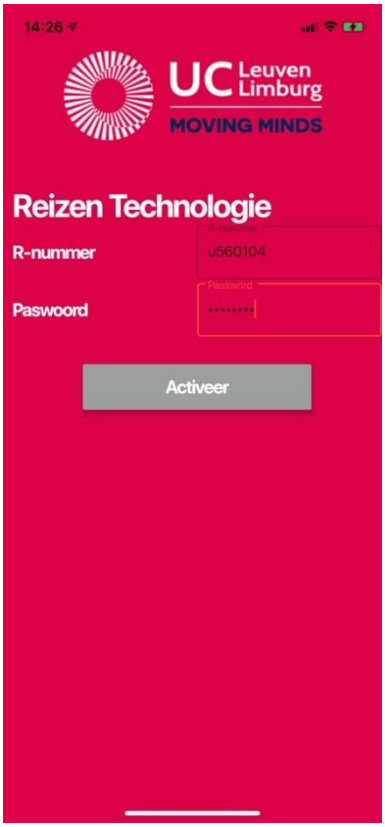
        Navigator.pushReplacement(
            context,
            new MaterialPageRoute(
                builder: (BuildContext context) =>
                new MainDart()));
      }
      else if (status != null && api_token == null && message != null){
        showAlertDialog(context, "Onjuiste inloggegevens gebruikt.");
      }
    }
```

Inlogview maken

- Als je de app voor de eerste keer opstart kom je in een inlogpagina terecht. Hierin moet je je R-nummer en wachtwoord ingeven om zo de rest van de app te kunnen zien.
- Voorbeeld:



Figuur 1 inlogscherm zonder tekst (ios)



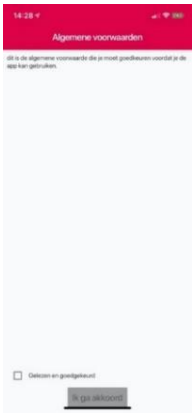
Figuur 2 inlogscherm met tekst (ios)

Voorwaardenview maken

- Wanneer de gebruiker ingelogd is, moet hij/zij alvorens toegang te krijgen tot de app eerst de algemene voorwaarden lezen en goedkeuren. Deze taak bestond er dus uit om een venster te maken dat verschijnt na het inloggen. Belangrijk was dat de gebruiker eerst de check box “Gelezen en goedgekeurd” moet aanvinken voor hij/zij naar de app kan navigeren. Daarom blijft de knop “Ik ga akkoord” uitgeschakeld tot de check box wordt aangevinkt.
- De moeilijkheid in deze taak zat in het uitschakelen van de knop. Dit is opgelost door bij de check box een state in te stellen. Deze state is vervolgens gekoppeld aan de onPressed property van de knop.

```
Column(
  children: <Widget>[
    Row(
      children: <Widget>[
        Checkbox(
            value: _isButtonDisabled,
            onChanged: (bool newValue) {
              setState(
                  () => _isButtonDisabled = !_isButtonDisabled);
            }),
        Text('Gelezen en goedgekeurd')
      ],
    ),
    RaisedButton(
      padding: const EdgeInsets.all(10.0),
      onPressed: !_isButtonDisabled
          ? null
          : () {
              acceptConditions(context);
            },
      color: Color.fromRGBO(224, 0, 73, 1.0),
      textColor: Colors.white,
      child:
          Text('Ik ga akkoord', style: TextStyle(fontSize: 20)),
    ),
  ],
)
```

- Voorbeeld:



Figuur 3: Voorwaardenview (iOS)

Algemene voorwaarden downloaden van server

- Om de Voorwaardenview op te vullen, wordt de data die hier tot toebehoort opgehaald vanuit de server. Om dit te kunnen doen gebruiken we het eerder uitgelegde GraphQl. Wanneer we de data hebben aangekregen, slaan we deze ook op in de lokale database van de app zodat we deze altijd kunnen raadplegen.

- De code hiervoor staat in het viewmodel. Twee van deze functies zijn verbonden met de view namelijk “setConnection” om de connectie met de externe database op te stellen via api. En “getData” die een widget naar de view terugstuurt om de data weer te geven. “getAlgemeneData” is de query die in de widget wordt gebruikt, deze wordt met de api naar de server gestuurd.

```
String getAlgemeneData() {
  String data ="""
    query{
      info(info_name:"algemene_voorwaarde")
      {
        info_value
      }
    }
    """;
  return data;
}


Link setConnection(){
  var bearer = globals.loggedInUser[0]["token"];
  print(bearer);
  final HttpLink httpLink =  HttpLink(uri: "http://171.25.229.102:8222/graphql?query=");
  final AuthLink authLink = AuthLink(getToken: () async => 'Bearer ' + bearer);
  final Link link = authLink.concat(httpLink);
  return link;
}

Query getData() {
  return Query(
      options: QueryOptions(document: getAlgemeneData()),
      builder: (QueryResult result, {
        BoolCallback refetch,
        FetchMore fetchMore,
      }) {
        if (result.data == null) {
          return Center(child: CircularProgressIndicator());
        }
        return ListView.builder(
          itemBuilder: (BuildContext context, int index) {
            return Text(result.data['info'][index]['info_value']);
          },
          itemCount: result.data['info'].length,
          scrollDirection: Axis.vertical,
        );
      });
}
```

Check netwerkstatus

- Toelichting
    - Bij elke synchronisatie met de remote database moet er nagegaan worden welk type verbinding er is: Wifi, 4G of geen. Dit is een vereiste van de app omdat deze een oogmerk heeft om in het buitenland te werken, waar 4G duur kost. Het doel van deze taak is om bij een verbinding met wifi de synchronisatie met de database meteen te starten. Bij 4G moet er een bericht komen dat vraagt om bevestiging voor 4G te gebruiken. Enkel bij een positief antwoord begint de synchronisatie. Bij een gebrek aan verbinding komt er hier een melding van en gaat de synchronisatie niet door.

- Code
    - Voor deze functionaliteit te bekomen is er een klasse Connection gemaakt. Voor het aanmaken van de klasse Connection is er alleen de context van de app nodig, zodat een instantie van de klasse berichten kan laten zien.

      Verder zijn er in deze klasse drie functies beschreven: checkConnectivity(), _using4GDialog en dbSync().

      Bij een aanvraag tot synchronisatie wordt een instantie van Connectivity aangemaakt en checkConnectivity() opgeroepen. Deze functie vraagt het verbindingstype van de telefoon op. Bij Wifi wordt de functie dbSync aangeroepen, bij 4G de functie _Using4GDialog en bij geen verbinding komt er melding op het scherm met het bericht dat er geen verbinding is.

      _Using4GDialog laat een alert op het scherm verschijnen die vraagt of de gebruiker wil synchroniseren op 4G. Bij een positief antwoord wordt dbSync() aangeroepen. Bij een negatief antwoord zal er melding op het scherm verschijnen dat de synchronisatie is afgebroken.

      dbSync() roept een functie van de database aan die de synchronisatie van start laat gaan.

```dart
class Connection
{
  BuildContext _context;

  Connection(BuildContext context):_context=context;

 checkConnectivity() async{
    var connection = await(Connectivity().checkConnectivity());
    if (connection == ConnectivityResult.mobile) {
      _using4GDialog(_context);
    }
    else if (connection == ConnectivityResult.wifi) {
      dbSync();
    }
    else if (connection == ConnectivityResult.none) {
      await globals.getEmergencyNumbers();
      Fluttertoast.showToast(
          msg: "no connection to data network",
          toastLength: Toast.LENGTH_SHORT,
          gravity: ToastGravity.BOTTOM,
          timeInSecForIos: 1,
          backgroundColor: Colors.red,
          textColor: Colors.white,
          fontSize: 16.0
      );
    }
  }
```

```
_using4GDialog (BuildContext context) {
  showDialog(
      barrierDismissible: false,
      context: context,
      builder: (BuildContext context) {
        return SimpleDialog(
          title: const Text('Continue with mobile data?'),

          children: <Widget>[
            SimpleDialogOption(
                onPressed: () {
                  Navigator.of(context, rootNavigator: true).pop();
                  dbSync();
                },
                child: const Text('Yes')
            ),
            SimpleDialogOption(
                onPressed: () {
                  Navigator.of(context, rootNavigator: true).pop();
                  Fluttertoast.showToast(
                      msg: "Sync cancelled",
                      toastLength: Toast.LENGTH_SHORT,
                      gravity: ToastGravity.BOTTOM,
                      timeInSecForIos: 1,
                      backgroundColor: Colors.red,
                      textColor: Colors.white,
                      fontSize: 16.0
                  );
                },
                child: const Text('no')
            )

          ],
        );
      });
}

void dbSync() async {

  await checkUpdate();
}
```

Check inlogstatus

- Een van de vereisten van de app is dat de gebruiker zich maar 1 keer moet aanmelden en de voorwaarden accepteren. Het doel van deze taak is dan ook om dit te verwezenlijken.

- Deze code gaan we in de main.dart schrijven. Enkel deze code en de connectie met de database zal worden geschreven in de main.dart. We gaan kijken of het veld isLoggedIn van de database voor de gebruiker true is en of de voorwaarden geaccepteerd zijn. Is dit het geval, dan komt de gebruiker uit op de home pagina, anders komt de gebruiker uit op de login. Als de gebruiker de voorwaarden nog niet heeft geaccepteerd, komt hij/zij op de voorwaarden pagina terecht. De code zie je hieronder.

```
builder: (BuildContext context, AsyncSnapshot snapshot) {
  switch (snapshot.connectionState) {
    case ConnectionState.none:
      return new Container();
    case ConnectionState.active:
      return new Container();
    case ConnectionState.waiting:
      return new Container();
    case ConnectionState.done:
      if (globals.isLoggedIn != true) {
        return new Inlog();
      }
      else if (globals.loggedInUser[0]["accepted_conditions"] == 0) {
        return new VoorwaardenConnection();
      }
      else {
        return new Navbar();
      }
  }
  return new Container();
},
```

Appbar noodnummer icoon

- Deze taak bestaat eruit dat in de appbar die bovenaan in de app geplaatst is, er altijd een icoon van de noodnummers zichtbaar is. Belangrijk is dus dat de gebruiker te allen tijde en vanuit iedere pagina van de app hierop kan klikken, zodat de noodnummers altijd beschikbaar zijn.
- In de code gebruiker we de klasse *Icons* om het icoontje in te laden. De klasse *Icons* is onderdeel van het material design, zoals uitgelegd bij “Flutter & Dart”. Het enige dat we hiervoor moeten doen is de parameter “uses-material-design” op true zetten in de *pubspec.yaml* file.

```
PopupMenuButton<String>(
  icon: Icon(Icons.call),
```

- Voorbeeld:

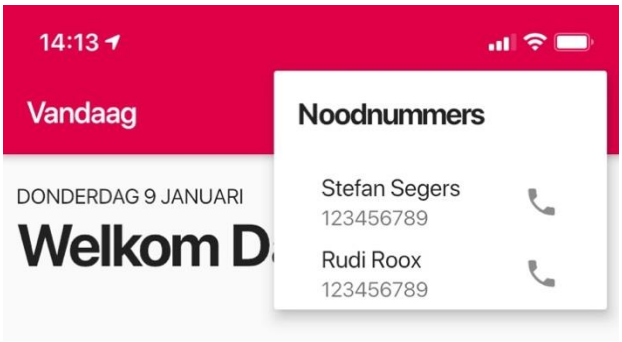
Figuur 4: Appbar met noodnummericoon (iOS)

Dropdown met noodnummers

- Wanneer de gebruiker op het icoon van noodnummers klikt, komt er een dropdown tevoorschijn met hierin de namen en telefoonnummers van de noodcontacten. De telefoonnummers moeten clickable zijn zodat hier een actie aan gekoppeld kan worden (bellen naar dit noodnummer).
-

```
PopupMenuButton<String>(
  icon: Icon(Icons.call),
  itemBuilder: (BuildContext context) => <PopupMenuEntry<String>>[
    PopupMenuItem<String>(
      value: 'Noodnummers',
      child: Text('Noodnummers',
          style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),
    ),
    for (var number in globals.emergencyNumbers)
      PopupMenuItem<String>(
        value: 'phone1',
        child: ListTile(
          trailing: Icon(Icons.call),
          title: Text(number['first_name'] + ' ' + number["last_name"]),
          subtitle: Text(number['number']),
          onTap: () =>
              launch("tel://" + globals.emergencyNumbers[0]['number']),
        ),
      ),
  ],
),
```

- Voorbeeld:



Figuur 5: Dropdown met noodnummers (iOS)

Instant call

- Wanneer men op een contact in de dropdown van de noodnummers klikt, moet dit noodnummer gebeld worden. Dit doen we met een instant call: een functie die ervoor zorgt dat de telefoonapp van de gsm wordt geopend en het nummer automatisch wordt ingevuld.
- We dachten dat deze opdracht moeilijker zou worden door de cross-platformcompatibiliteit en dat we voor Android en iOS telkens een andere functie zouden moeten gebruiken. Dit bleek niet het geval. Volgende functie blijkt compatibel te zijn met beide platformen:

```
launch("tel://" + globals.emergencyNumbers[0]['number']),
```

Sprint 2

Startdatum: 22-10-2019

Sprint backlog

User Story	Taken	Requirements/Toelichting
Als geregistreerde gebruiker **wil ik** de knop overzicht kunnen bedienen **zodat ik** een chronologisch overzicht krijg van de reis gegroepeerd per bestemming	- View planning opmaken - Data lokaal ophalen voor planning - Bottom navigation	- Navigatie naar gedetailleerde pagina (lege pagina) - Naam van de stad en per stad een lijst van de dagen. gegevens uit lokale db
Als geregistreerde gebruiker **wil ik** de lijst van alle hotels kunnen raadplegen **zodat ik** kan zien in welke hotels ik ga verblijven.	- View hotel opmaken - Dummy lokale tabellen aanmaken - Bottom navigation	- Hotels in grid lijst (vierkantjes) uit lokale db - Als ik een hotel selecteer navigeer ik naar een gedetailleerde pagina hierover (voorlopig lege pagina)
Als geregistreerde gebruiker **wil ik** in de lijst van hotels op een hotel kunnen klikken **zodat ik** informatie krijg van het betreffende hotel en de indeling van alle kamers voor dat hotel kan raadplegen	- Data lokaal ophalen voor hotel - View: info hotel + kamerindeling opmaken - Bottom navigation	- Carousel die de indeling van alle kamers bevat - Info over het hotel - Foto van het hotel

View planning opmaken

- Het eindpunt van deze taak is dat de planning pagina in de app volledig af is en voldoet aan een aantal criteria. De pagina moet scrollable zijn en de Appbar moet zichtbaar zijn. Scrollable betekent dat vanaf het moment dat er info buiten het scherm valt, de gebruiker moet kunnen scrollen. Verder moet er voor elke bestemming een tussentitel zijn. Onder elke tussentitel staan meerdere cards die de dagen op de bepaalde locatie voorstellen. Als de gebruiker op deze card drukt navigeert hij/zij naar een lege pagina. Deze pagina wordt in een volgende sprint opgevuld met meer info over deze bepaalde dag. En de laatste vereiste is dat de opmaak van de pagina past bij de opmaak van de app. De code die hiervoor geschreven wordt, komt in de view van deze pagina.

- De code die we gaan gebruiken om deze pagina op te bouwen is vergelijkbaar met de code voor alle andere pagina's. In het viewmodel wordt er een functie geschreven om alle data uit SQFlite te halen. Dit is een andere taak, en hier moeten we ons dus niets van aantrekken. We moeten natuurlijk wel gaan kijken hoe we de velden moeten noemen, zodat dit overeen komt met het viewmodel. De grootste moeilijkheid van deze pagina is dat we niet boven elke dag een tussentitel van de locatie willen. We moeten dus gaan zorgen dat dit alleen gebeurt vanaf het moment dat de volgende dag in een nieuwe stad plaatsvindt. Dit gaan we doen met de onderstaande code.

```
if (location != content[index]['location']) {
  titelWidget = Visibility(
    child: Padding(
      padding: EdgeInsets.only(top: 20),
      child: Text(
          content[index]['location'],
          style: TextStyle(fontSize: 35)
      ),
    ),
    visible: true,
  );
}
location = content[index]['location'];
```

Data lokaal ophalen voor planning

- We moeten de view van de planning natuurlijk ook kunnen vullen met data. Dit doen we met enkele functies die in het viewmodel staan vermeld.

- In het viewmodel zijn drie functies beschreven om de gewenste data op te halen per situatie. Alle drie functies zijn volgens het principe van Future, dit zorgt ervoor dat de functies asynchroon gebeuren en de app niet zou vasthangen. "GetDayPlannings" haalt ALLE planningen op uit de lokale database uit de tabel genaamd "days". "GetDayPanningData" haalt ÉÉN planning op door het meegeleverde id wanneer men in de view op een planning drukt. De if-else statements dienen voor debugging om in de console van Android Studio te bekijken of de data wel werkelijk is opgehaald. "GetAllData" is een functie die bovenstaande twee functies combineert wanneer je ze beide nodig hebt.

```
Future<List> GetDayPlannings() async {
  List<Map> dayPlannings = await globals.database.query("days");

  if(dayPlannings != null) {
    print("data dayplanning ophalen gelukt: " + dayPlannings.toString());
  }
  else{
    print("error data dayplanning ophalen");
  }

  return dayPlannings;
}


Future<List> GetDayPlanningData(int id) async {
 List<Map> dayPlanningData =
                 await globals.database.query('days', where: '"id" = ?', whereArgs: [id]);

 if(dayPlanningData != null) {
  print("data day planning met id " + id.toString() + " ophalen gelukt: " +
    dayPlanningData.toString());
 }
 else{
  print("data day planning met id " + id.toString() + " ophalen NIET gelukt: ");
 }

 return dayPlanningData;
}


Future<List> GetAllData(int planningId) async {
  List<Map> allData = new List();
  List<Map> DayPlanning = await GetDayPlanningData(planningId);
  List<Map> DayPlannings = await GetDayPlannings();

  allData.add({'day_planning': DayPlanning, 'day_plannings': DayPlannings});

  return allData;
}
```

Bottom navigation

- Voor te kunnen navigeren tussen de verschillende pagina's van de app komt er onderaan een navigatiebar.
- Eerst om onze navbar te gebruiken moeten we in onze main.dart zetten dat als de voorwaarde in orde zijn dat we naar de navbar widget "navigeren". Vanuit deze widget gaan we kijken welke pagina gedrukt is en vervolgens naar deze navigeren.

```
int selectedIndex = 0;
final widgetOptions = [
  new VandaagPage(),
  new PlanningPage(),
  new CarsPage(),
  new HotelsPage(),
  new ContactPage(),
];

void onItemTapped(int index) {
  setState(() {
    selectedIndex = index;
  });
}
```

- Voorbeeld:

Figuur 6 navbar (android)

View hotel opmaken

- Deze taak bestond eruit de view op te maken voor het overzicht van alle hotels. Op die manier kan de gebruiker, wanneer hij/zij via de appbar naar "Hotels" navigeert, in één overzicht alle hotels en wat korte info bekijken. Vervolgens moeten de hotels clickable zijn, om door te navigeren naar de volledige info van het geselecteerde hotel.
- Een probleem dat opdook bij deze taak was dat de titel van een hotel vaak te lang was voor de voorziene ruimte, en daardoor een overflow error gaf. Dit hebben we opgelost door het gebruik van een FittedBox die we de property "fitWidth" hebben gegeven.
-

```
FittedBox(
  fit: BoxFit.fitWidth,
  child: Text(
    content[index]['name'],
    style: TextStyle(
       fontSize: 23, fontWeight: FontWeight.bold),
  ),
),
```

- Verder wordt er voor de algemene opmaak gebruik gemaakt van een “GridView.count”. We halen op hoeveel hotels er in deze reis zitten en maken vervolgens met “List.generate” evenveel kolommen aan.
- ```
  GridView.count(
    crossAxisCount: 2,
    padding: EdgeInsets.all(10.0),
    mainAxisSpacing: 10.0,
    crossAxisSpacing: 10.0,
    // Generate 100 widgets that display their index in the List.
    children: List.generate(content.length, (index) {
      ...
  ```

Data lokaal ophalen voor hotel

- Deze taak is weer voor het ophalen van de data uit de lokale database, deze keer voor de view van hotel. Dit is weer mogelijk gemaakt met een viewmodel die in de view wordt gebruikt.

- In het viewmodel worden twee functies gebruikt voor de gewenste situatie. “GetHotels” haalt alle hotels uit de lokale database op, en “GetHotelData” haalt één hotel op d.m.v. het meegeleverde id samen met de kamers die bij dit hotel horen. De kamers worden ook gesorteerd van klein naar groot via het kamernummer. Voor de rooms wordt ook elke reiziger die tot deze room behoord opgehaald.
  ```

```
Future<List> GetHotels() async {
  List<Map> hotels = await globals.database.query("hotels");
  if(hotels != null) {
    print("data hotels ophalen gelukt: " + hotels.toString());
  }
 return hotels;
}


Future<List> GetHotelData(int id) async {
  List<Map> hotel = await globals.database.query('hotels', where: '"id" = ?', whereArgs: [id]);
  List<Map> rooms = await globals.database.query('rooms',
    orderBy: "room_number ASC",where: '"hotel_id" = ?',whereArgs: [id]);

  List<Map> travellers = new List();
  if(rooms.isNotEmpty) {
    for (int i=0; i<rooms.length; i++) {
      int room_id = rooms[i]['id'];
      List<Map> travellersRoom = await globals.database.query(
          'room_traveller', where: '"room_id"=?', whereArgs: [room_id]);
      if(travellersRoom.isNotEmpty) {
        for (int a = 0; a < travellersRoom.length; a++) {
          List<Map> traveller = await globals.database.query(
              'travellers', where: '"id"=?',
              whereArgs: [travellersRoom[a]['traveller_id']]);
          travellers.add({
            'room': room_id,
            'traveller': traveller[0]['first_name'] + ' ' +
                traveller[0]['last_name']
          });
        }
      }
    }
  }
  List<Map> hotelData = new List();
  hotelData.add({'hotel':hotel[0], 'rooms':rooms, 'travellers':travellers});
  return hotelData;
}
```

Dummy lokale tabellen aanmaken

- Toelichting
  - Om de views van data te kunnen voorzien moeten we, tot dat we kunnen synchroniseren met de backend, dummy data aanmaken. Dit gebeurt door models te maken van de gevraagde data. Hierna gaan we dit model toevoegen in de database zodat deze gebruikt kan worden in de views.

- Code
  - Eerst moet er in de OnCreate methode van OpenDatabase een tabel gecreëerd worden. Dit moet je doen voor elke tabel die je nodig hebt. Je kan dit gewoon doen met een SQL-query die je meegeeft bij de execute methode. In deze query geef je de naam mee en ook alle velden van de tabel. Als je een relatie moet leggen kan dat met FOREIGN KEY – REFERENCES.
  - ```
    await db.execute(
      "CREATE TABLE travellers ("
          "id INTEGER PRIMARY KEY,"
          "first_name TEXT,"
          "last_name TEXT,"
          "major_name TEXT,"
          "phone TEXT,"
          "driver INTEGER,"
          "car_id INTEGER,"
          "FOREIGN KEY(car_id) REFERENCES cars(id))"
    );
    ```
  - Hierna kan je een model maken zodat er een instantie gemaakt kan worden. In dit model moeten alle velden gedeclareerd worden en moet je het object instantiëren. Ook moet je een toMap() methode maken zodat je de insert functie van het database object kunt gebruiken.
  - ```
    class Traveller {
      final int id;
      final String first_name;
      final String last_name;
      final String major_name;
      final String phone;
      final int driver;
      final int car_id;

      Traveller({this.id, this.first_name, this.last_name,
    this.major_name, this.phone, this.driver, this.car_id});

      Map<String, dynamic> toMap() {
        return {
          'id': id,
          'first_name': first_name,
          'last_name': last_name,
          'major_name': major_name,
          'phone': phone,
          'driver': driver,
          'car_id': car_id
        };
      }
    }
    ```

- Als je het model hebt kan je hier een object van maken en in de database zetten.
- ```
  var traveller1 = Traveller(
      id: 1,
      first_name: 'a',
      last_name: 'test',
      car_id: 1,
      phone: "555"
  );
  ```
- Tot slot moet je de insert functie van het database object uitvoeren. Hier geef je de tabel naam mee en de map van het object. Verder kan je ook nog het confictAlgorithm instellen hiermee stel je in of een object met hetzelfde ID overschreven moet worden in de database of niet.
- ```
  db.insert("travellers", traveller1.toMap(),
      conflictAlgorithm: ConflictAlgorithm.replace);
  ```

View: info hotel + kamerindeling opmaken

- Wanneer men zich op het scherm van “Hotel” bevindt en hier op een hotel naar keuze klikt, dan navigeert men naar de “Info hotel” pagina. Op deze pagina krijgt men zowel de kamerverdeling als de informatie over het hotel te zien. Tussen deze 2 delen bevindt zich een licht grijze lijn zodat deze duidelijk gescheiden zijn van elkaar. De kamerverdeling wordt met behulp van een “carrouselview” getoond. Dit is een middel waardoor de gebruiker naar links en rechts kan scrollen om telkens een andere kamer te zien. Op de afbeelding bij het voorbeeld wordt het duidelijk waar het om gaat. Deze pagina bevat bovendien ook nog een afbeelding van het hotel dat de gebruiker heeft aangeklikt. Tenslotte is deze pagina gemaakt met een “scrollview”. Dit wil dus zeggen dat indien de pagina te groot is voor je scherm, je verticaal naar beneden kunt scrollen.

- De carousel voor de kamers werd aangemaakt met een horizontale SingleChildScrollView met een gegenereerde lijst. De horizontale Scrollview zorgt ervoor dat de Row die erin zit genoeg virtuele plaats krijgt en dat er een horizontale scrollbeweging mogelijk is. De gegenereerde lijst wordt verder uitgelegd bij de Autoview.

```
SingleChildScrollView(
    scrollDirection: Axis.horizontal,
    child: Row(
      children: List.generate(content[0]['rooms'].length,
          (index) {
        return Container(
          width: MediaQuery.of(context).size.width - 30,
          decoration: BoxDecoration(
              border: Border.all(
                  width: 5, color: Colors.black26),
              borderRadius: const BorderRadius.all(
                  const Radius.circular(20))),
          child: makeWidget(content[0], index, max),
          padding: EdgeInsets.all(5.0),
          margin: EdgeInsets.all(5.0),
        );
      }).toList(),
    )),
```

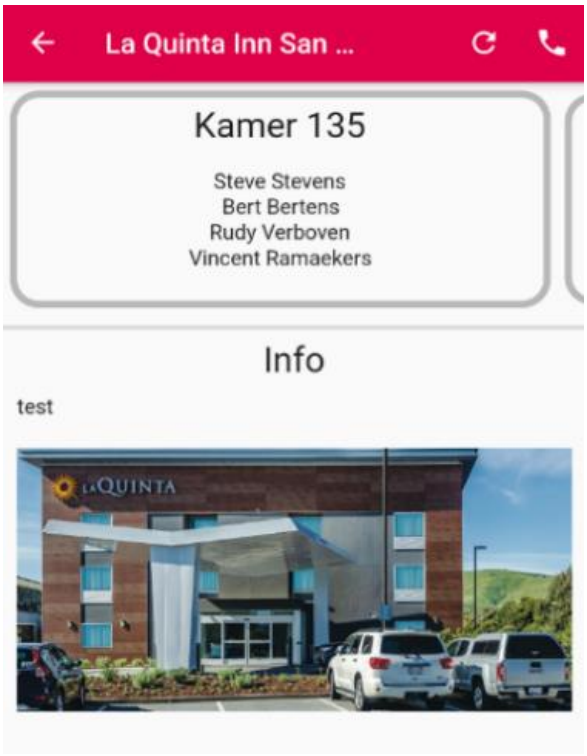
- Voor de grijze lijn werd onderstaande code gebruikt. De “thickness” staat voor de dikte van de lijn. Deze is op 3 gezet zodat dit het beste resultaat opleverde.

```
Row(children: <Widget>[
  Expanded(
      child: Divider(
    thickness: 3,
  )),
]),
```

- Voor de afbeelding heeft men gebruik gemaakt van een “AspectRatio”. De verhouding van de afbeelding die gebruikt is, is 2 (in de breedte van het scherm) op 1 (lengte van het scherm). De afbeelding gaat dus altijd in dit formaat getoond worden. Foto’s die verticaal genomen zijn, kan men dus beter niet gebruiken.

```
new AspectRatio(
  aspectRatio: 2 / 1,
  child: Container(
    padding: new EdgeInsets.all(10.0),
    child: new Image.network(
      content[0]['hotel']['photoUrl'],
      fit: BoxFit.cover,
    ),
  ),
)
```

- Voorbeeld:



Sprint 3

Startdatum: 05-11-2019

Sprint backlog

User Story	Taken	Requirements/Toelichting
Als geregistreerde gebruiker **wil ik** de refreshknop kunnen bedienen **zodat ik** de nieuwe data ophaal met een waarschuwing indien ik mobiel ben	- Refresh icon toevoegen	- Bij klikken begint sync (dummy functie) - Permanente refreshknop zichtbaar
Als geregistreerde gebruiker **wil ik** de telefoonnummers van mijn medereizigers kunnen bekijken **zodat ik** deze kan contacteren	- Opmaak contactenview	- De ListView moet scrollable zijn - ListView met alle contacten uit de lokale database
Als geregistreerde gebruiker **wil ik** de auto-indeling kunnen raadplegen **zodat ik** kan zien wie bij wie in de auto zit	- Data lokaal ophalen voor auto - Carouselview auto	- Per auto zichtbaar maken - View opvullen met auto's uit lokale db - Carouselview maken - Passagiers per auto uit lokale db halen - Variabele aantal passagiers tot +-50 zichtbaar maken
Als geregistreerde gebruiker met netwerkverbinding **wil ik** telkens als ik de app open de nieuwe info van de remote database ophalen **zodat ik** de meest recente gegevens kan raadplegen in de app en deze opgeslagen worden op het toestel.	- Trip Type GrapQL - Traveller Type GraphQL - Vervoer Type GraphQL - Planning Type GraphQL - Hotel Type GraphQL	- Begint bij startup - Waarschuwing 4g - Lokale database synct met remote database (reizigers, auto's, hotels, planning, noodnummers, algemene info)

Refresh icon toevoegen

- Bovenaan de appbar is een permanente refreshknop geplaatst die een syncfunctie uitvoert. M.a.w. wordt de grote db gesynchroniseerd met de lokale db.

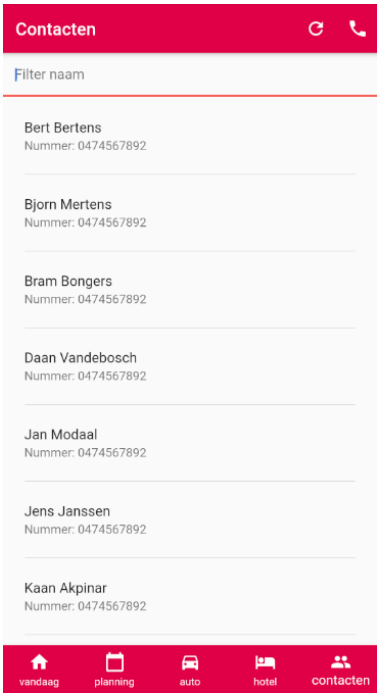
- Voorbeeld:

Figuur 7: Refresh icon in de Appbar (Android)

Opmaak contactenview

- Op de contacten pagina vindt kom je een lijst met travellers tegen met hun telefoonnummer

- Voorbeeld:

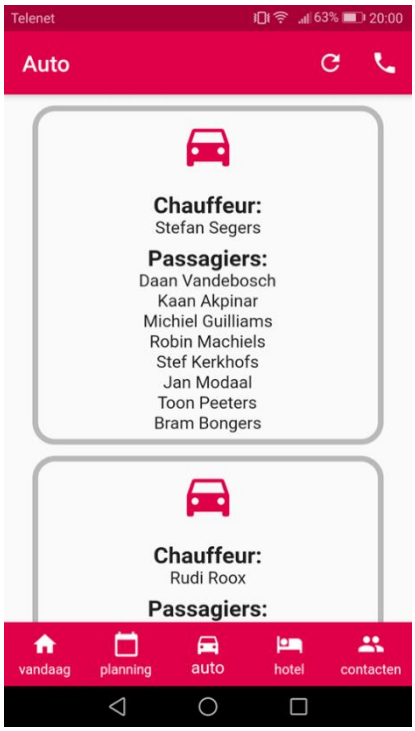


Figuur 8 contacten view(android)

Carouselview auto

- Toelichting
  Bij het openen van de auto-pagina is er een Scrollview zichtbaar die een lijst van alle wagens met hun chauffeur en passagier weergeeft.

- Voorbeeld:



- Code
  Voor de lijst op te vullen met gestandaardiseerde kaartjes is er gebruik gemaakt van een gegenereerde Listview. Hierbij worden de elementen in de Listview niet meegegeven maar aangemaakt tijdens het aanmaken van de Listview. Bij List.generate() moet er een lijst meegegeven met items en moet de index bijgehouden worden. In de functie wordt beschreven hoe deze lijstelementen er uit moeten zien. In dit geval dus een Container die wordt opgevuld met de functie makeWidget() die een Widget maakt van het meegegeven element van de lijst. In dit geval een Column die zo in de omliggende Container wordt aangemaakt.

```
ListView(
    padding: EdgeInsets.all(5.0),
    children:
    List.generate(content.length, (index){
            return
              Container(
               decoration: BoxDecoration(
               border: Border.all(width: 5, color: Colors.black26),
               borderRadius: const BorderRadius.all(const
Radius.circular(20))
             ),
             child: makeWidget(content[index],index),
             padding: EdgeInsets.all(5.0),
             margin: EdgeInsets.only(top:5.0,bottom: 5.0,left:20.0,right:
20.0)

           );
}).toList(),
)
```

Data lokaal ophalen voor auto

- Om de carouselview van de auto's te bekijken, moet er natuurlijk weer data ter beschikking zijn. In deze taak wordt uitgelegd hoe deze data uit de lokale database wordt gehaald m.b.v. het viewmodel dat door de carouselview wordt gebruikt.

- Voor de data van de auto's op te halen zijn er weer twee functies voor de gewenste situatie. "GetCars" haalt al de auto's op en ook al de reizigers die in deze auto zitten. "GetCarData" haalt dan weer één enkele auto op via het meegeleverde id.

```
Future<List> GetCars() async {
  List<Map> cars = await globals.database.query("cars");
  List<Map> data = new List();
  List<Map> travellers = await globals.database.query('travellers');
  if(cars != null){
  for(int i=0;i<cars.length;i++)
    {
      int counter = int.parse(cars[i]['id'].toString());
      List<Map> carTravellers = new List();
      for(int a=0; a<travellers.length; a++)
         {
           if(travellers[a]['car_id']==cars[i]['id']) {
             carTravellers.add({
               'naam': travellers[a]['first_name'] + ' ' +
                   travellers[a]['last_name']
             });
           }
         }
      if(carTravellers!=null) {
        List<Map> findDriver = await globals.database.query('travellers',where: '"id" =?',whereArgs:
[cars[i]['driver_id']]);
        String chauffeur = findDriver[0]['first_name']+' '+findDriver[0]['last_name'];
        data.add({'reizigers': carTravellers,'chauffeur':chauffeur});
      }
    }
  }
  return data;
}

Future<List> GetCarData(int id) async {;
List<Map> carData = await globals.database.query('cars', where: '"id" = ?', whereArgs: [id]);
if(carData != null) {
  print("data auto met id " + id.toString() + " ophalen gelukt: " + carData.toString());
}
else{
  print("data auto met id " + id.toString() + " ophalen NIET gelukt: ");
}
return carData;
}
```

Trip Type GraphQL

- Toelichting
    - Alle informatie over in reis wordt opgehaald via het trip type. Zoals hieronder te zien is bevat dit type lijsten van andere types. Het opvragen van deze data is beveiligd, je hebt hiervoor de bearer_token nodig.
    - De “name” is de naam van de trip waarin de gebruiker zit. De trip wordt opgehaald aan de hand van de traveller die is ingelogd.
    - Bij “travellers” krijgen we een lijst met alle personen die meegaan tijdens deze reis.
    - “Hotels” bevat een lijst met alle hotels waar we overnachten tijdens de reis.
    - Met “planning” krijgen we het volledige overzicht van alle dagen en de activiteiten die plaatsvinden tijdens de reis.
    - Voor noodgevallen zijn er de “emergencynumbers”, dit zijn de gsm-nummers van de docenten en begeleiders die meegaan tijdens de reis.
    - De laatste in de rij is “transports” dit bevat een lijst met alle voertuigen (auto’s, busjes, …) die gebruikt worden in de reis. Je vindt hierin de chauffeur en de personen die meerijden.

- Code

```
type Trip {
    name: String!
    travellers: [Traveller!]
    hotels: [Hotel!]! @belongsToMany
    days: [Day!]!
    emergencynumbers: [EmergencyNumber!]!
    transports: [Transport!]!
     }
```

Traveller Type GraphQL

- Toelichting
    - Het “traveller” type bevat al de nodige informatie over de reiziger.
    - Voornaam en achternaam spreken voor zich,
    - Verder krijgen we ook de reiziger zijn telefoon en studierichting
    - Als laatste krijgen we van deze reiziger ook de kamers waarin hij slaapt voor elk hotel tijdens de reis.

- Code

```
type Traveller {
   traveller_id: ID!
   first_name: String!
   last_name: String!

   phone: String!
   major: Major!
   rooms: [Room!]!
    }
type Major {
   major_id: ID!
   major_name: String!
    }
```

Vervoer Type GraphQL

- Toelichting
    - Het “vervoer” type bevat informatie over het vervoer van en naar de verschillende activiteiten en plaatsen. Het gaat hierbij niet over bv. Een vlucht naar Amerika.
    - Het “driver_id” verwijst naar een docent of begeleider zijn traveller_id. Deze persoon is (gedurende de ganse reis) de bestuurder van deze auto/bus.
    - De “Size” geeft aan hoeveel plaatsen er zijn in het voertuig, hierbij is de bestuurder niet meegerekend.
    - In “travellers” steekt een lijst met alle personen die in het voertuig zitten.

- Code

```
type Transport {
   transport_id: ID!
   driver_id: Int!
   size: Int!
   travellers: [Traveller!]!
    }
```

Planning Type GraphQL

- Toelichting
    - Hierin zit een lijst met alle dagen van de huidige reis. Per dag wordt een planning opgemaakt van meerdere activiteiten.
    - Elke dag heeft een datum en start-locatie,
    - “Highlight” is de titel van die dag en is het hoogtepunt van die dag,
    - “Description” geeft meer uitleg over die dag (bv. Speciale voorzieningen, …).
    - Een planning heeft een start- en einduur en een activitie.
    - Er zijn meerdere activiteiten op een dag mogelijk.
    - Een activity tot slot heeft een naam, omschrijving en locatie.
    - De locatie is de plaatst waar de activiteit doorgaat (bij bv. Een museumbezoek is dit het adres van het museum).
- Code

```
type Day {
    day_id: ID!
    date: String!
    highlight: String!
    description: String!
    location: String!
    plannings: [Planning!]!
}
type Planning {
    planning_id: ID!
    start_hour: String!
    end_hour: String!
    activity: Activity!
}
type Activity {
    activities_id: ID!
    name: String!
    description: String!
    location: String!
     }
```

Hotel Type GraphQL

- Toelichting
    - Op dit punt in de sprint bevatte het “hotel” type enkel de naam, adres en afbeelding link.
    - De relatie naar de kamers moest via dezelfde tussentabel gebeuren als van hotel naar trip. Dit zorgde voor problemen met de huidige versie van graphql. In de volgende sprint zijn we hiermee verdergegaan.

Sprint 4

Startdatum: 19-11-2019

Sprint backlog

User Story	Taken	Requirements/Toelichting
Als geregistreerde gebruiker **wil ik** wanneer de app opstart een overzicht krijgen van de dagplanning **zodat ik** weet wat ik die dag moet doen en de gegevens krijg van de auto en kamer indeling	- VandaagView	- Zichtbaar vanaf begin reis, daarvoor een pagina met welkombericht met countdown op plaats van planning en knop voor algemene info.
Als geregistreerde gebruiker **wil ik** specifieke contacten kunnen opzoeken door de naam van een persoon in te geven in een zoekbalk in de pagina "contacten" **zodat ik** een specifieke persoon kan contacteren indien nodig	- Contacten filteren	- Contacten kunnen filteren op naam - Bellen als je op contact drukt
Als geregistreerde gebruiker **wil ik** alle relevante info van de reis kunnen raadplegen **zodat ik** voor het vertrek weet waarmee er rekening gehouden moet worden	- Algemene info view	- Algemene info ophalen uit database - Algemene Info zichtbaar op vandaag pagina
Als organisator **wil ik** als ik in het menu op “algemene info” druk, naar een pagina word herleid **zodat ik** de algemene info kan zien	- View algemene info back-end - Controller algemene info back-end	- Er wordt een eloquent backend voor algemene info voorzien - Als organisator wil ik als ik in het menu op “algemene info” druk, naar een pagina word herleid zodat ik de algemene info kan zien
Als geregistreerde gebruiker met netwerkverbinding	- Hotel Type GraphQL - RoomType GraphQL	- Begint bij startup - Waarschuwing 4g

wil ik telkens als ik de app open de nieuwe info van de remote database ophalen **zodat ik** de meest recente gegevens kan raadplegen in de app en deze opgeslagen worden op het toestel. **(vervolg)**	- Noodnummers Type GraphQL	- Lokale database synct met remote database (reizigers, auto's, hotels, planning, noodnummers, algemene info)

Hotel Type GraphQL

- Probleemstelling
    - Graphql gebruikt de modellen van Laravel voor het volgen van relaties.
    - De relatie van trip naar hotel wordt hierin uitgeschreven en verloopt dus correct.
    - Om het hergebruik van hotels over verschillende reizen toe te staan was het de bedoeling dat de rooms in een hotel niet rechtstreeks aan een hotel gekoppeld worden maar aan de primary key gevormd door hotel_id en trip_id.
    - Deze laatste relatie kregen we niet duidelijk gemaakt aan Laravel en dus als gevolg ook niet aan graphql.
    - Om dit probleem op te lossen hebben we geprobeerd om een tweede graphql engine (“Nuwave/lighthouse”) te installeren, deze had hetzelfde probleem en vormde dus geen oplossing. We zijn er wel achter gekomen dat deze versie van graphql eenvoudiger is in gebruik. Er werd besloten om alles van de oude engine om te zetten naar de nieuwe.
- Toelichting
    - Het hotel type bevat informatie over het hotel zelf zoals naam, adres en afbeelding link
    - Via het type “hoteltrip” verkrijgen we de rooms in het hotel.
    - Hierin vinden we de kamernummer en de grote.
    - Voor elke kamer krijgen we ook een lijst met alle travellers die in de kamer slapen.

- Code

```
type Hotel {
    hotel_id: ID!
    hotel_name: String!
    address: String!
    picture1_link: String
    hoteltrips: [HotelTrip!]!
     }
type HotelTrip {
    id: ID!
    hotel_id: Int!
    start_date: String!
    end_date: String!
    rooms: [Room!]!
}
type Room {
    room_id: ID!
    hotel_trip_id: Int!
    room_number: Int
    size: Int!
    travellers: [Traveller!]!
     }
```

Noodnummer Type GraphQL

- Toelichting
    - In geval van nood moeten docenten te bereiken zijn.
    - Het “emergencyNumber” type bevat al de nodige info hiervoor zoals voornaam en naam van de docent en zijn gsm-nummer (van de simkaart die op die plaats te bereiken is bv. Belgisch nummer voor Duitsland en Amerikaanse nummer (van Amerikaanse provider) voor Amerika reis)

- Code

```
type EmergencyNumber {
    emergency_number_id: ID!
    number: String!
    first_name: String!
    last_name: String!
}
```

VandaagView opmaken

- De VandaagView is de eerste pagina die de gebruiker te zien indien de gebruiker is ingelogd. Op deze pagina krijgt de gebruiker in één oogopslag een overzicht van alle actuele data. Bovenaan staat de datum en wordt de gebruiker begroet met de eigen naam. Hieronder komt een aftelklok te staan die weergeeft over hoeveel dagen de gebruiker op reis vertrekt. De dag dat de reis begint, verdwijnt deze klok.

- Hieronder wordt de planning van de huidige dag weergegeven in een openklapbaar blokje. De gebruiker kan hier ook navigeren naar de echte PlanningView door op “toon alles” te klikken. Dan komt hij/zij automatisch uit op de dag van de weergegeven planning, met alle activiteiten voor die dag. Voor de reis is begonnen, wordt hier gewoon de planning van de eerste dag weergegeven.

  Onder de huidige planning wordt het hotel getoond waarin de reiziger die avond zal slapen. Hier staat een foto en de naam van het hotel. Ook hier kan het blokje weer open geklikt worden, in dit geval om de kamerindeling van het hotel van de huidige dag te bekijken. De gebruiker kan ook weer verder klikken op “toon alles” om de hotelinfo van het huidige hotel te bekijken.

  Onderaan de pagina krijgt de gebruiker een stukje van de algemene reisinfo te zien. Indien de gebruiker de volledige info wilt lezen, kan er weer worden doorgeklikt naar een aparte view met de volledige algemene info.

- Het probleem van deze opdracht was dat we enorm veel verschillende data moesten ophalen uit het viewmodel. Normaal wordt er echter telkens maar één tabel uit de database tegelijk opgehaald, afhankelijk van het karakter van de view (hotels voor hotelView, auto’s voor autoView,…). We hebben dit opgelost door in een asynchrone functie “GetAllData()” alles in een Future<List> te steken. In deze functie wordt ook gezocht naar de dayPlanning van de huidige dag met de functie “findDayPlanning()“ door alle planningen af te gaan tot de datum van de dagplanning overeenkomt met de huidige datum. Analoog wordt het huidige hotel gezocht.

```
Future<List> GetAllData() async {

 //var now = DateTime.now();
 //var today = DateTime(now.year, now.month, now.day);
 String todayString='2020-5-21'; //testData
 var today = new DateFormat("yyyy-MM-dd").parse(todayString);//testdata

 List<Map> allData = new List();
 List<Map> Users = await GetUser();
 List<Map> Plannings = await GetDayPlannings();

 dynamic _findDayPlanning(){
  for(int i = 0; i<Plannings.length; i++)
    {
      String planningDateString = Plannings[i]['date'];
      var planningDate = new DateFormat("yyyy-MM-dd").parse(planningDateString);
      if(planningDate == today)
      {
        return Plannings[i];
      }
    }
  return -1;
}
var dayPlanning = _findDayPlanning();
var Planning;
if (dayPlanning != -1)
{
  Planning = dayPlanning;
}
else if (dayPlanning == -1)
{
  Planning = null;
}

List<Map> Hotels = await GetHotels();
List<Map> HotelData;
```

```
int _findHotelId(){
   for(int i = 0; i<Hotels.length; i++)
    {
      String hotelStartDateString = Hotels[i]['start_date'];
      String hotelEndDateString = Hotels[i]['end_date'];
      var hotelStartDate = new DateFormat("yyyy-MM-dd").parse(hotelStartDateString);
      var hotelEndDate = new DateFormat("yyyy-MM-dd").parse(hotelEndDateString);
      if((hotelStartDate == today || hotelStartDate.isBefore(today)) && hotelEndDate.isAfter(today))
      {
        return  Hotels[i]['id'];
      }
    }
   return -1;
 }
 int hotelId = _findHotelId();
 if(hotelId != -1)
  {
    HotelData = await GetHotelData(hotelId);
  }
 else if (hotelId == -1)
  {
    HotelData = new List<Map>();
  }

 List<Map> Info = await GetInfo();
 List<Map> Trips = await GetTrips();

 allData.add({'user':Users[0],'day_planning': Planning, 'hotels': Hotels, 'hotel_data': HotelData,
'trip_info': Info, 'trips': Trips, 'hotel_id': hotelId});

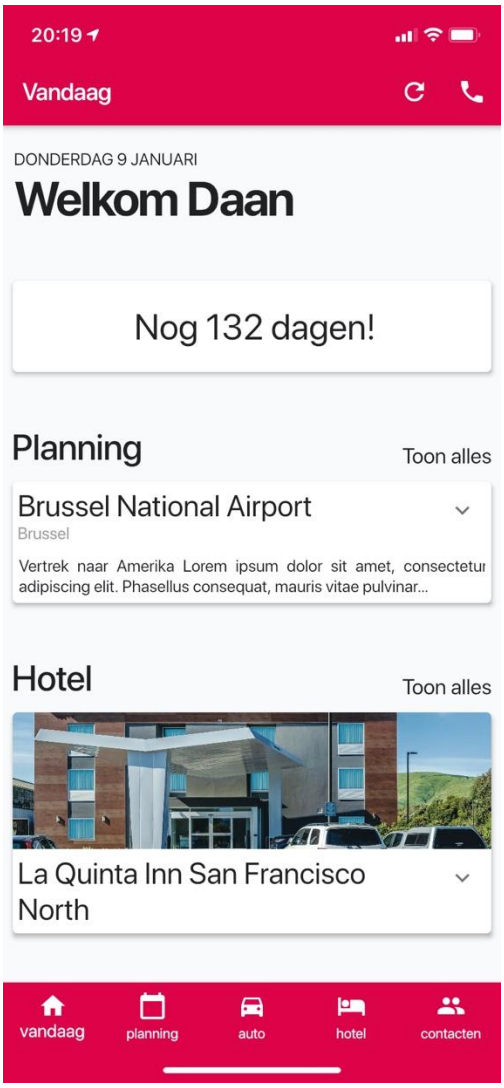
 return allData;
}
```

- De functie “_countDays” telt het aantal dagen tot de reis begint. Er wordt gekeken naar het verschil in dagen tussen de startdatum van het eerste hotel, en de huidige datum. Dit doen we met de functie “difference()”.

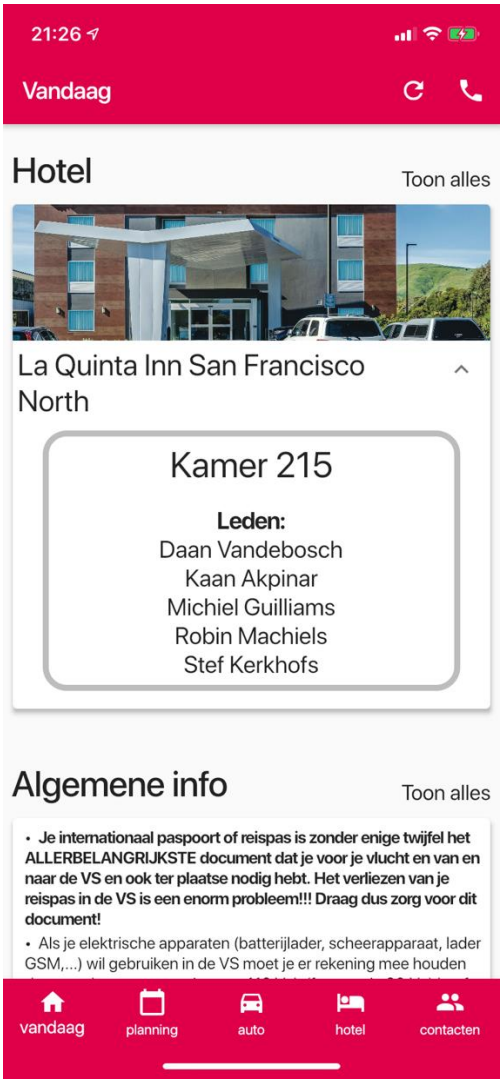
```
int _countDays(var content) {
 var today = DateTime(now.year, now.month, now.day);
 //String todayString='2018-05-20'; //testData
 //var today = new DateFormat("yyyy-MM-dd").parse(todayString);//testdata
 String leaveDateString = content[0]['hotels'][0]['start_date'];
 var leaveDate = new DateFormat("yyyy-MM-dd").parse(leaveDateString);
 dynamic counter = leaveDate.difference(today).inDays;

 return counter;
}
```

- Voorbeeld:



Figuur 9: VandaagView (iOS)



Figuur 10: VandaagView met hotelindeling (iOS)

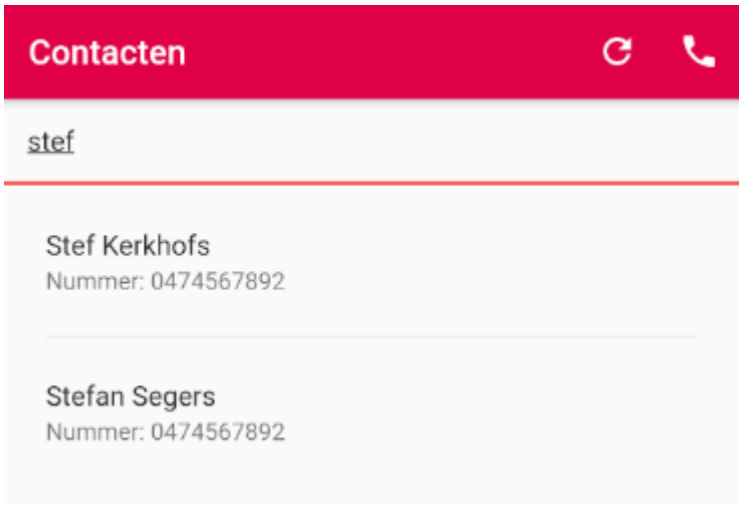
Contacten filteren

- In de contacten pagina hebben we een tekst balk bovenaan de pagina voor de contacten te kunnen filteren. Dit kan zowel op voornaam als op achternaam. Als men op 1 van de contacten drukt wordt men doorverwijst naar de naar de telefoon app van uw gsm met de nummer al ingevoerd zodat je rechtstreeks de gekozen persoon kan bellen.

- Voor te filteren halen we eerst alle users op en zetten deze in een list. Vervolgens filteren we deze list op de string dat we in de tekst vak hebben ingetypt en geven de gefilterde resultaat mee in een list.

```
TextField(
  decoration: InputDecoration(
    contentPadding: EdgeInsets.all(15.0),
    hintText: 'Filter naam',
  ),
  onChanged: (string) {
    setState(() {
      filteredUsers = filteredUsersNietOphalen
          .where((u) =>
      (u['first_name']
          .toLowerCase()
          .contains(string.toLowerCase()) ||
          u['last_name'].toLowerCase().contains(
              string.toLowerCase())))
          .toList();
    });
  },
),
```

- Voorbeeld:



Figuur 9 opzoeking contactenpagina (android)

Algemene info view

- De algemene info view is een simpele view waarin de algemene info in HTML-formaat wordt getoond. Dit zorgt ervoor dat er op de client side geen problemen met opmaak zullen voorkomen. De administrator kan deze pagina in de back-end opbouwen in HTML-formaat en op die manier gemakkelijk opmaak en foto's gebruiken.
- Om html te tonen maken we gebruik van het pakket "flutter_html.dart". Hierdoor kunnen we simpelweg de functie "Html()" gebruiken om de data te laten zien.

```
Html(
  data: algemeneInfoString,
),
```

- Voorbeeld:

Algemene info lokaal ophalen

- Deze taak is zodat de view van algemene info wordt opgevuld met data.

- Een simpele functie voor de algemene info uit de lokale database op te halen.

```
Future<List> GetInfo() async {
  List<Map> info = await globals.database.query("trip_info");
  if(info != null) {
    print("info ophalen gelukt: " + info.toString());
  }
  return info;
}
```

View algemene info back-end

- Het doel van deze taak is het schrijven van de eerste view van de back-end. De back-end is de website die we gaan gebruiken om de data toe te voegen aan de database, en dus ook aan de app. De website die we moeten gebruiken bestaat al, en is geschreven in Laravel. We gaan dus Laravel moeten gebruiken om deze pagina te schrijven. Voor Laravel zijn er meerdere programmeeromgevingen, voorbeelden hiervan zijn Visual Studio Code, NetBeans en PHP Storm. Deze pagina heeft 2 acceptatiecriteria. De pagina moet een tekst vak bevatten waarin de algemene info komt. En er moet een link bestaan waardoor ik via de navbar naar deze pagina kan navigeren. We gaan dus zelf een bestand schrijven en bestaande bestanden aanpassen.

- Om deze pagina te gaan schrijven moeten we een Blade-bestand aanmaken binnen het Laravelproject. Dit bestand maken we aan binnen de map resources/views/organizer. Om deze pagina te bekijken moeten we eerst een link leggen, binnen het project, naar deze pagina. Dit gaan we doen binnen het bestand web.php. We moeten ook gaan kijken waar we onze link toevoegen, omdat niet iedere gebruiker toegang mg hebben tot alle pagina's. De algemene info pagina mag enkel bereikbaar zijn voor gebruikers die de rol van organisator hebben. Daarom gaan we onze link leggen bij de link van Specific Organisator. De link ziet eruit zoals de onderstaande code.

  ```
  Route::get('info', 'Organiser\InfoController@getInfo')->name('info');
  ```

  Om naar de pagina te gaan moeten we nu deze link toevoegen in de navbar. Dit doen we in nav.blade.php. Ook hier zetten we de code zodat alleen organisatoren toegang hebben. Deze code ziet er als volgt uit:

  ```
  <a class="dropdown-item" href="{{ route('info') }}">Algemene info</a>
  ```

  De rest van de pagina is de opmaak van de view in info.blade.php. Dit is gewoon een textbox en twee knoppen toevoegen.

Controller algemene info back-end

- Om de view van de algemene info ook effectief te weergeven, is er een controller vereist. Deze controller wordt de "InfoController" genoemd en bevat 2 functies. Namelijk een construct functie (magic method) die wordt uitgevoerd elke keer wanneer de controller wordt opgeroepen en een andere functie genaamd "getInfo". In de construct functie stelt men de variabele "info", die men in het begin heeft geïnstantieerd, gelijk aan het model van de "infoRepository". Dit is nodig om in de functie "getInfo" gebruik te maken van functies uit deze infoRepository. In de functie "getInfo" gaat men namelijk deze repository gebruiken om de algemene info uit de remote database te halen. De onderstaande code is de code van de infoController:

```
class InfoController extends Controller
    {
        private $info;

        public function __construct(InfoRepository $info)
        {
            $this->info = $info;
        }

        public function getInfo(){
            $oInfo = $this->info->get('algemene_info');

            return view('organizer.info', array(
                'oInfo' => $oInfo,
            ));
        }
    }
```

- Zoals u ziet, roept men in de “getInfo” methode een methode “get” op. Deze methode bevindt zich in de InfoRepository. Deze InfoRepository is een interface die geïmplementeerd wordt door “EloquentInfo”. In “EloquentInfo” wordt de code van deze methode (get) geschreven. De get-methode zorgt er eigenlijk gewoon voor dat men de info uit de remote database haalt. Indien dit niet lukt, geeft deze een foutmelding terug. Vervolgens gaat de getInfo methode een object genaamd “oInfo” gelijk stellen aan de waarde die men juist heeft teruggekregen. En tenslotte kan men dan de view returnen samen met het object “oInfo”. In de view (info.blade) hoort men dan nog volgende code te plaatsen zodat men in het tekstveld de algemene info (oInfo) te zien krijgt:
  {{ Form::textArea('info_value', $oInfo->info_value, ['class' => 'form-control, html-editor']) }}

- De code van “EloquentInfo” ziet er bovendien als volgt uit:

```
class EloquentInfo implements InfoRepository
{
 public function get($sInfoName)
    {
        try {
            $info = Information::where('info_name', $sInfoName)->firstOrFail();
            return $info;
        }catch (ModelNotFoundException $ex) {
            return "Sorry but this link has no content";
        }
    }
}
```

Sprint 5

Startdatum: 03-12-2019

Sprint backlog

User Story	Taken	Requirements/Toelichting
Als organisator **wil ik** als ik in het menu op “Planning” druk **zodat ik** alle dagplanningen kan zien van de reis waar ik aan gekoppeld ben	- Planning view	- Een overzicht van alle dagen
Als organisator **wil ik** als ik op de knop “toevoegen druk” **zodat ik** deze dagplanning aan de reis kan toevoegen	- Planning view	- Knop nieuwe dagplanning - Pop-up om alle data in te geven - Annuleerknop - Save-knop - Nieuwe dag meteen tonen in lijst
Als organisator **wil ik** als ik op de pagina planning zit en bij een bepaalde planning op de knop “wijzig planning” druk **zodat ik** deze planning kan wijzigen	- Planning view	- Knop edit - Pop-up om alle data aan te passen - Annuleerknop - Save-knop - Verwijder-knop - Data aanpassen in lijst - Data tonen in pop-up
Als organisator **wil ik** op de pagina algemene info iets in het tekst vak type en op de knop “opslaan” drukken **zodat ik** de algemene info kan aanpassen	- Algemene info wijzigen	- HTML-editor om opmaak toe te voegen
Als organisator **wil ik** op de pagina planning op een knop drukken **zodat ik** aan deze planning een activity kan koppelen en wegschrijven	- Activitiesview maken	- Bouwstenen gebruiken - Navigeren vanuit planning - Toevoegen aan dag - Eventueel Drag and Drop
Als geregistreerde gebruiker **wil ik** op een planning kunnen drukken **zodat ik** alle activities in de planning kan zien	- Activity_widget	- Activities zichtbaar - Kalendermenubalk - Scrollable - Cards - Sorteren op begin uur

Planning view

- Bij deze opdracht is het einddoel om een dag te kunnen toevoegen, verwijderen en bewerken in de back-end. Dit gaan we implementeren in het Laravelproject, net zoals de andere pagina's van de back-end. Voor deze pagina zijn er ook weer enkele acceptatiecriteria. Deze pagina moet enkel bereikbaar zijn voor organisatoren via de navbar. Als de pagina wordt geopend moet er een lijst met alle dagen te zien zijn.

  Er is ook een knop voorzien om een nieuwe dag toe te voegen, wanneer er op deze knop gedrukt wordt is er een pop-up te zien met alle velden die ingevuld moeten worden. Boven aan de pagina is er ook de mogelijkheid om naar aan andere reis te navigeren, indien de organisator is ingeschreven bij meerdere reizen.

  In de lijst van dagen is er bij elke dag een knop om de info te zien en een knop om een de dag te wijzigen. Verder is er ook de mogelijkheid om een link te leggen naar de pagina oom activiteiten toe te voegen aan de geselecteerde dag. Deze pagina is een combinatie van meerdere User Stories. Maar omdat de afhankelijkheden groot zijn, hebben we besloten om dit samen te nemen binnen 1 taak.

- Om deze pagina te maken gaan we verschillende bestanden moeten aanmaken. We hebben een Blade-bestand nodig voor de view, een controller om alle functies aan te roepen vanuit de view, en een repository en een eloquent-bestand om de data uit de database te halen.

  In de repository gaan we de naam en de parameters van alle functies schrijven die we nodig hebben om data op te halen. In het eloquent-bestand gaan we al deze functies volledig uitschrijven.

  Deze functies gaan we dan weer gebruiken in de controller om alle data op de juiste manier te verwerken. Bijna elke functie in de controller gaat de data uit de database verwerken en in het juiste formaat steken.

  Een voorbeeld hiervan is de Index-functie. Deze functie gaat van elke dag de data, die nodig is in de lijst in de view, in een array steken en doorsturen naar de view. Deze array kunnen we dan gaan uitlezen in de view en laten zien in de lijst. Het opmaken van de lijst en de knoppen op de pagina gebeurt via Bootstrap, in opmaak zal er dus weinig tot geen werk gestoken worden.

  Het enige ingewikkeldere aan deze pagina zijn de pop-ups. Om deze te gaan schrijven gaan we gebruik maken van modals binnen de view. Bij elke knop gaan we een link leggen naar de juiste modal zodat de data kan worden meegestuurd. Hoe dit gedaan is zien we in de onderstaande regel code.

```
<button type="button" class="btn btn-primary" data-toggle="modal" data-
target="#dayplanningeditPopup" data-
day='{{$oDayplanning}}'><i class="fas fa-edit"></i>edit</button>
```

Om een modal te maken, maken we gebruik van een div. Dit is een HTML-element dat wordt gebruikt meerdere elementen te groeperen. Aan deze div gaan we meegeven dat dit een modal is. Dit doen we op de volgende manier.

```
<div class="modal fade" id="dayplanningPopup" tabindex="-
1" role="dialog" aria-labelledby="dayPlanningPopupLabel">
```

Verder gaan we ook een div gebruiken om de elementen van de modal op te delen. Er komt zo een div voor de header, de body en de footer. Binnen de body gaan we alle velden zetten die we nodig hebben. We gaan dit doen op de volgende manier.

```
{{ Form::open(array('action' => 'Organiser\DayPlanningController@createD
ayPlanning', 'method' => 'post', 'files' => true)) }}
                {!! Form::hidden('Destination', $oCurrentTrip->name) !!}
                <div class="modal-body">
                    <div class="form-group">
                        {{Form::label('Highlight','highlight van de dag:
')}}
                        {{Form::text('Highlight', null, array('class' =>
 'form-control','required' => 'required'))}}
                        {{Form::label('Description','descriptie van de d
ag')}}
                        {{Form::text('Description', null,  array('class'
 => 'form-control','required' => 'required'))}}
                        {{Form::label('Date','Date:')}}
                        {{Form::text('Date', null, array('class' => 'for
m-control','required' => 'required')) }}
                        {{Form::label('Location','locatie van de dag:')}
}
                        {{Form::text('Location', null, array('class' =>
'form-control','required' => 'required'))}}
                        {{Form::hidden('Trip_id', $oCurrentTrip-
>trip_id)}}
                    </div>
                </div>
```

Boven de body gaan we ook meegeven welke functie van de controller er moet uitgevoerd worden door onze modal.
In de footer gaan we tot slot nog de knoppen zetten de nodig zijn voor onze modal.

Voor de modals binnen de lijst gaan we ook javascript code moeten toevoegen. Dit komt door het feit dat we moeten kunnen meegeven welke dag wordt geselecteerd. We gaan dus binnen de javascript code aangeven welke data er moet worden opgehaald. Deze code ziet er als volgt uit.

```
$('#dayplanningeditPopup').on('show.bs.modal', function(event){
        var button = $(event.relatedTarget);
        var data = button.data('day');
        var modal = $(this);
        modal.find('#Highlight').val(data.highlight);
        modal.find('#Description').val(data.description);
        modal.find('#Date').val(data.date);
        modal.find('#Location').val(data.location);
        modal.find('[name="Trip_id"]').val(data.trip_id);
        modal.find('[name="Day_id"]').val(data.day_id);
    });
```

We kunnen hier zien dat we de data voor elke veld gaan ophalen met de javascript code. Met deze code schrijven we ook de andere modal in de lijst.

Algemene info wijzigen

- Om de algemene info te wijzigen gaat men verder op hetgeen wat er in de vorige sprint is gedaan. In de info.blade plaatsen we eerst 2 knoppen. Eén voor het opslaan en één voor het annuleren van de wijzigingen:
  {{ Form::submit('Opslaan') }}
  {{ Form::button('Annuleren', array('type' => 'button', 'onclick' => 'history.go(0)', 'value' => 'Annuleren')) }}

  De “annuleren”-knop is hier al zo ingesteld dat deze de pagina refreshed naar de oorspronkelijke staat. Hierdoor worden alle wijzigingen dus ongedaan gemaakt.
- Nadat de view in orde is gebracht, kan men in de InfoRepository een nieuwe functie schrijven, genaamd “updateInfoPage”. Vervolgens schrijft men deze functie dan ook in “EloquentInfo”. Met behulp van deze functie kan men een meegestuurde waarde (hetgeen dat ingegeven is in het tekstveld) in de remote database plaatsen/updaten. De code van deze functie ziet er als volgt uit:

```
public function updateInfoPage($sInfoContent)
    {
        Information::where('info_name', 'algemene_info')->update(['info_value' =>
$sInfoContent]);
    }
```

- Het volgende dat moet gedaan worden, is het toevoegen van een functie genaamd “updateInfo” aan de InfoController. Deze functie ziet er zo uit:

```
public function updateInfo(Request $request){
        $sInfoContent = $request->post('info_value');
        if (strlen($sInfoContent) == 0){
            $sInfoContent = "";
        }
        $this->info->updateInfoPage($sInfoContent);
        return redirect()->back()->with('message', 'De info pagina is aangepast');
    }
```

  In de eerste regel van deze functie gaat men kijken wat er in het tekstveld is ingegeven. We stellen dit gelijk aan een string “sInfoContent”. Vervolgens stuurt men de waarde van deze string mee met de functie “updateInfoPage”. Deze functie zorgt er dan voor dat de waarde in de database wordt aangepast. Ten slotte refreshed men de pagina opnieuw en laat men een bericht zien dat aangeeft dat de pagina is aangepast.

Activities view (backend)

- Voor activities hebben we met react gewerkt. Het was de bedoeling om een “drag and drop” systeem te maken waarmee je activities kan overslepen dan een dagplanning. Deze pagina verkrijg je wanneer je op de pagina van dagplanningen op activities drukt.

- Om react te gebruiken moeten we de code hiervan onderaan de html pagina tussen <script> tag’s zetten. Er zijn twee render functies, de eerste render hieronder maakt een kaartje aan volgens de layout die in de tweede render wordt uitgelegd. Onderstaande render wordt uitgevoerd wanneer de pagina laadt. Hier wordt dan via blade in de linker kolom van de pagina alle activiteiten weergegeven, en in de rechterkolom komen de activiteiten die al in de dagplanning zitten.

```
render() {
    return (
        React.createElement("div", { className: "container" },
            React.createElement("div", { id: "links", className: "container" },
                @foreach($activities as $oActivity)
                React.createElement(Card, { h3: "{{$oActivity->name}}" , body:
"{{$oActivity->description}}",
                    activityId: "{{$oActivity->activity_id}}",
                    activityName:"{{$oActivity->name}}",
                    activityDescription: "{{$oActivity->description}}",
                    activityLocation: "{{$oActivity->location}}",
                    @foreach($plannings as $oPlanning)
                        @if($oActivity->activity_id == $oPlanning->planning_id)
                            timeBegin: "{{$oPlanning->start_hour}}",
                            timeEnding: "{{$oPlanning->end_hour}}"
                        @endif
                    @endforeach
                },),


                @endForeach
                //
            ),


            React.createElement("div", { id: "rechts", className: "container" },

                @foreach($dayActivities as $dayActivity)
                    @foreach($activities as $oActivity)
                        @if($dayActivity->activity_id == $oActivity->activity_id)
                            React.createElement(Card, { h3: "{{$oActivity->name}}" , body:
"{{$oActivity->description}}",
                                activityId: "{{$oActivity->activity_id}}",
                                activityName:"{{$oActivity->name}}",
                                activityDescription: "{{$oActivity->description}}",
                                activityLocation: "{{$oActivity->location}}",
                                @foreach($plannings as $oPlanning)
                                    @if($oActivity->activity_id == $oPlanning->planning_id)
                                timeBegin: "{{$oPlanning->start_hour}}",
                                timeEnding: "{{$oPlanning->end_hour}}"
                                @endif
                                @endforeach
                            },),
                        @endif
                    @endforeach
                @endforeach
                //
            )));
}}
```

- Deze render functie zorgt ervoor dat men een standaard kaart formaat kan maken die we vervolgens in zijn geheel meerdere keren kunnen creëren.

```
render() {
        return (
            React.createElement("div", { className: "card" },
                React.createElement("div", { className: "card-header" },
                    React.createElement("h3",{id: "activityHeader"},this.props.h3,
                        React.createElement("button", {type: "button", className:
"avtivityButton fas fa-edit float-right btn btn-primary",
                            'data-toggle': "modal", 'data-target': "#editActivityModal",
                            'data-activity-id': this.props.activityId,
                            'data-activity-name': this.props.activityName,
                            'data-activity-description': this.props.activityDescription,
                            'data-activity-location': this.props.activityLocation,
                        })),

                ),


                React.createElement("div", { className: "card-body" },
                    React.createElement("p", null, this.props.body),
                    React.createElement("b", null, 'begin:'),
                    React.createElement("input",{id:'begin', type: 'time', value:
this.props.timeBegin} ),
                    React.createElement("b", null, 'eind:'),
                    React.createElement("input",{id:'eind', type: 'time', value:
this.props.timeEnding} ))));
    }}


React.render(React.createElement(App, null), document.getElementById('container'));
```

Activities controller (Backend)

- Dit is de controller voor het model Activities.

- De controller werkt volgens het CRUD-systeem.

  Om alle activiteiten op te halen wanneer de pagina laadt, wordt de controller naar volgende functie herleid:

```
public function showAllActivities($dayId)
{
    $activities = $this->activities->getActivities();
    $plannings = $this->activities->getPlannings();
    $activitiesInDay = $this->activities->getActivitiesInDay($dayId);

    return view('organizer.activities', ['activities' => $activities, 'plannings' => $plannings,
'dayActivities' => $activitiesInDay]);
}
```

Create:

```
public function createActivity(ActivityForm $request)
{
    $iActivityId = $request->post('activity-id');
    $aData['name'] = $request->input('activity-name');
    $aData['description'] = $request->input('activity-description');
    $aData['location'] = $request->post('activity-location');

    $this->activities->addActivity($aData);
    return redirect()->back()->with('message', 'De activiteit is succesvol opgeslagen');
}
```

Update:

```
public function updateActivity(ActivityForm $request)
{
    $iActivityId = $request->post('activity-id');
    $aData['name'] = $request->input('activity-name');
    $aData['description'] = $request->input('activity-description');
    $aData['location'] = $request->post('activity-location');

    $this->activities->updateActivity($aData, $iActivityId);
    return redirect()->back()->with('message', 'De activiteit is aangepast');
}
```

Delete:

```
public function deleteActivity($id){
    $this->activities->deleteActivity($id);
    return redirect()->back()->with('message', 'Je hebt succesvol de activiteit verwijdert.');
}
```

Save:

```
public function saveActivities($dayId, $activityIds){
    if ($activityIds != 0) {
        $activityIds = str_replace(',', '', $activityIds);
        $activityIds = count_chars($activityIds, 3);
        $this->activities->saveOrDeleteActivities($dayId, $activityIds);

        return redirect()->back()->with('message', 'Je hebt succesvol de activiteiten aan de dag
gelinkt.');
    } else{
        $this->activities->saveOrDeleteActivities($dayId, -1);

        return redirect()->back()->with('message', 'Je hebt succesvol de activiteiten aan de dag
gelinkt.');
    }
}
```

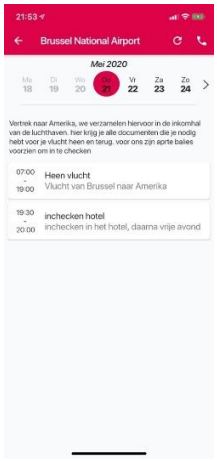
Activity_widget

Op deze pagina krijgt de gebruiker bovenaan de pagina eerst een kleine slidekalender te zien. Hij kan hier iedere dag van de reis selecteren in de kalender om het overzicht van die dag te zien. Iets lager staat de samenvattende beschrijving van de dag. Hieronder worden alle activities in cards weergegeven.

Speciaal aan deze view is de slidekalender. We gebruiken het package calendar_strip.dart. Hierdoor kunnen we de widget “CalendarStrip()” gebruiken. Deze heeft een heel aantal parameters. De belangrijkste zijn *startData* en *endDate* om te bepalen welke data zichtbaar zijn, *selectedDate* om de huidige dag weer te geven en *onDateSelected* om de actie te koppelen aan het klikken op een andere dag.

```
CalendarStrip(
  startDate: DateFormat("yyyy-MM-dd")
      .parse(content[0]['day_plannings'][0]['date']),
  endDate: DateFormat("yyyy-MM-dd").parse(content[0]
          ['day_plannings']
      [content[0]['day_plannings'].length - 1]['date']),
  selectedDate: DateFormat("yyyy-MM-dd")
      .parse(content[0]['day_planning'][0]['date']),
  onDateSelected: onSelect,
  dateTileBuilder: dateTileBuilder,
  iconColor: Colors.black87,
  monthNameWidget: _monthNameWidget,
  markedDates: markedDates,
  containerDecoration: BoxDecoration(),
)
```

Voorbeeld:



Figuur 10: Activities_widget (iOS)

Finetuning

In de laatste sprint hebben we ook de app nog getest op eventuele bugs / fouten en die er ook uitgewerkt. De weggewerkte fouten zijn:

- De grootte van de hotel kaders aanpassen
- Vandaagview database error
- Sync knop dialoog oplossen
- Kijken of de database aangepast is
- Algemene lay-out toepassen in de app
- Kamers sorteren op nummer
- Een checkbox bij algemene voorwaarden
- Contacten alfabetisch sorteren
- Verdere kleine bugs wegwerken

Realistische seeders

Tot nu toe was de data die gebruikt werd in de database (zowel lokaal als remote) dummy data, soms automatisch gegenereerd of gewoon opgebouwd in loops. Bij het gebruik van de app gaf dit vaak een vertekend beeld van wat we eigenlijk wilden laten zien? Sommige views hebben juist lange teksten nodig om hun volledige werken in aan te tonen.

Om dit probleem op te lossen hebben we alle database seeders onder handen genomen. Als voorbeeld hiervoor hebben we een programma bundel van een paar jaar geleden gebruikt. Als we nu de app laten synchroniseren zien we dat er nu deftige data in staat. Auto's/busjes bevatten nu alleen nog personen die mee gaan op de reis, voordien kan dat eender welke persoon zijn in de database. Hotels bevatten effectief adressen en niet random gegenereerde strings, hetzelfde geld voor de afbeeldingen in de app.

Ook is er een volledige planning uitgewerkt met activiteiten, deze was er voordien niet. Hierdoor kunnen we nu ook de planning laten zien in de app.

Besluit

In dit project hebben we veel bijgeleerd over de agile methodiek en hoe communicatie een heel belangrijke rol hierin speelt. Bij het begin waren hier moeilijkheden mee maar we zijn op de harde manier erachter gekomen dat zonder communicatie het niet werkt. De communicatie tijdens de lessen en op Discord heeft het ons hierbij geholpen om hulp aan elkaar te vragen en de nodige updates te geven. En na elke sprint een retrospective houden heeft ons ook geholpen om de werking van ons team te verbeteren.

Ook hebben we geleerd te werken met flutter en dart. Dit was iets nieuws voor ons en was daarom ook wel een uitdaging. Zoals bij elke uitdaging heeft het wel zijn ergernissen meegebracht maar dit is uiteindelijk ook tot een goed einde gebracht.

Dus conclusie dit was een interessant project met een handige methodiek wat fijn is om mee gewerkt te hebben en in de toekomst te gebruiken voor een project in teamverband tot een goed einde te brengen.