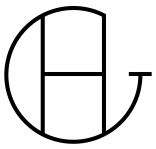

HoGent

Faculteit Bedrijf en Organisatie

Opslag en synchronisatie van offline data bij mobiele applicaties

Simon Jang

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Stefaan De Cock
Co-promotor:
Sam Verschueren

Instelling: Pridiktiv.care - Into.care

Academiejaar: 2016-2017

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Opslag en synchronisatie van offline data bij mobiele applicaties

Simon Jang

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Stefaan De Cock
Co-promotor:
Sam Verschueren

Instelling: Pridiktiv.care - Into.care

Academiejaar: 2016-2017

Tweede examenperiode

Samenvatting

Mobiele applicatieontwikkeling beschouwt toegang tot internet vaak als vanzelfsprekend wanneer een mobiele applicatie wordt gebruikt. Dit is echter niet het geval in elke (werk)omgeving. Daarom moet de ontwikkelaar ervoor zorgen dat de applicatie de data ook lokaal kan opslaan indien de wijzigingen niet meteen kunnen worden doorgevoerd naar de achterliggende infrastructuur. Dit probleem vormt voor ontwikkelaars een uitdaging omdat de data integriteit moet worden gewaarborgd wanneer het toestel terug verbonden is met internet. Het gebruik van een performante en betrouwbare methode voor de data die offline wordt ingegeven te synchroniseren met de online cloud-based databank is dus essentieel. Hierbij is het belangrijk om een onderscheid te maken tussen use cases waarbij de developer gebruik maakt van fully-managed databases zoals onder andere DynamoDB van Amazon Web Services en use cases waarbij men zelf de database beheert op verschillende virtual machines. In het onderzoek is synchronisatie specifiek onderzocht voor de fully-managed database DynamoDB die gebruikt van een serverless microservices architectuur en een Angular applicatie als client applicatie.

Na onderzoek is gebleken dat er verschillende manieren van synchronisatie zijn. Bij de synchronisatie van de offline data is het belangrijk om de verschillende use cases onder te verdelen volgens synchronisatie methode en zo verder te werken. Deze methodologie werd ook gehanteerd bij de opbouw van van het prototype. Er zijn scenario's waar er geen conflicten zijn maar performantie de belangrijkste factor is bv. Read-Only Optimised. Wanneer er conflicten optreden kan men dan kiezen voor Last Write Wins waarbij er dan onvermijdelijk data verloren gaat of andere strategieën van conflict resolution. Terwijl het onderzoek zich voornamelijk focust op data synchronisatietechnieken, kan het interessant zijn om bijvoorbeeld de kost in kaart te brengen om over stappen van een fully-managed database naar een andere provider die synchronisatie toelaat zonder dat men zelf veel rekening moet houden met synchronisatie.

Voorwoord

Mijn interesse in mobiele- en web applicaties was voor mij de reden om de opleiding Toegepaste Informatica aan de Hogeschool Gent te starten. Deze bachelorproef vormt het sluitstuk in mijn opleiding en de keuze van het onderwerp is tot stand gekomen door de samenwerking met mijn stageplaats Pridiktiv.care - Into.care. Er was nood aan een onderzoek naar offline data opslag en synchronisatie bij hun mobiele applicatie. Met de groei van IoT en de digitalisering binnen verschillende sectoren, lijkt offline / online synchronisatie relevanter dan ooit.

Het schrijven van een bachelorpaper is geen eenvoudige opdracht en ik zou daarom enkele personen willen bedanken voor hun ondersteuning en expertise. Eerst en vooral wil ik mijn co-promotor en stage-mentor Sam Verschueren bedanken voor zijn inzet en geduld bij de talloze vragen die ik het gesteld in verband met web applicaties. Daarnaast ben ik ook zeer dankbaar voor de intense begeleiding en feedback die ik heb ontvangen van mijn promotor Stefaan De Cock. Tenslotte wil ook mijn partner en vrienden bedanken voor de hulp die ze hebben aangeboden bij het lezen van mijn bachelorproef en de morele ondersteuning.

Inhoudsopgave

1 Glossarium 9

2 Inleiding 11

2.1 Business Case 11

2.2 Terminologie 12

2.2.1 Huidige Web APIs voor lokale opslag 12

2.2.2 Scalable Angular Architecture 13

2.2.3 Ngrx store: een state container 14

2.2.4 Reactive programming paradigm 14

2.2.5 Offline First 14

2.2.6 Amazon Web Services 15

2.3 Stand van zaken 15

2.4 Probleemstelling en Onderzoeksvragen 16

2.5 Opzet van deze bachelorproef 16

3 Methodologie 19

3.1 Research 19

3.1.1 Beperkingen door business case 19

3.2 Protoyping 20

3.3 Testen van prototype 20

3.4 Conclusie 20

4 Opstelling 21

4.1 Client side 21

4.2 Server side 21

5 Synchronisatie patterns 23

5.1 Read-Only Data 23

5.2 Read-Only Data Optimized 24

5.3 Read/Write Data Last Write Wins 24

5.4 Read/Write with Conflict Detection 24

6 Onderzoek 27

7 Conclusie 29

Bibliografie 29

1. Glossarium

Single-Page Application, SPA : Een single-page application (SPA) is een web applicatie of website waarbij noodzakelijke HTML, CSS en JavaScript dynamisch wordt ingeladen of bij de eerste page load, afhankelijk van de acties van de gebruiker. De volledige pagina wordt bij een SPA nooit volledig herladen. Het is wel mogelijk dat onderdelen van de pagina dynamisch wordt gewijzigd. De data van de SPA wordt normaal dynamisch opgevraagd van de web server.

Single Source Of Truth, SSOT : In de context van het ontwerp van informatica systemen is de single source of truth een techniek om data op een bepaalde manier te structureren zodat die niet gedupliceerd wordt binnen de applicatie en alle referenties verwijzen naar dezelfde 'bron'. Alle informatie wordt opgehaald in een centraal punt en dat is voor de applicatie en haar componenten de enige plaats waar die informatie kan worden opgehaald. Ngrx store vervult die rol in een Angular applicatie.

Angular CLI : Angular CLI is een command line interface tool waarbij een volledige Angular project wordt gebouwd met minimale configuratie. Er wordt een basis structuur, tests, root module en root component aangemaakt. Met behulp van Webpack worden alle files dan gebundeld in enkele static files. Dankzij Angular CLI wordt er heel wat tijd uitgespaard omdat een groot deel van de configuratie wegvalt. Angular CLI wordt gebruikt de business case van Pridiktiv en in het prototype.

localForage : localForage is een library van Mozilla waarbij het mogelijk is om data lokaal te cachen. De library maakt gebruik van asynchrone storage mogelijkheden als IndexedDB en WebSQL met een localStorage-achtige syntax. De localStorage DOM Storage wordt wel gebruikt indien de webbrowser geen ondersteuning biedt voor IndexedDB of WebSQL. In dit geval is de opslagruimte om te cachen wel beperkter.

Progressive enchancement : Progressive enhancement is een ontwikkel strategie bij web development waarbij de focus wordt gelegd op de belangrijkste business require-

ments die de web applicatie of website moet invullen. Afhankelijk van de end-user's browser en connectie, kunnen er 'lagen' van functionaliteit (features) worden toegevoegd aan de applicatie of website. Op die manier kan er altijd een minimum worden aangeboden aan de end-user.

Conflict resolution : Een applicatie kan data van een remote databank lokaal opslaan voor offline functionaliteit aan te bieden aan de gebruiker. Wanneer de applicatie terug online gaat en de applicatie of remote databank een verschil opmerkt, dan is er sprake van een conflict. Conflict resolution duidt op de methode(s) die worden gebruikt voor het synchroniseren van de data tussen de verschillende databanken.

Event-driven Architectuur : Een software architectuur waarbij de verschillende componenten events genereren, detecteren en op een gepaste manier reageren. De event-driven architectuur vormt de basis van ReactiveX en serverless computing.

Asynchroon : Asynchroon of 'Asynchronous' in de context van web development wil zeggen dat een bepaalde handeling niet real-time maar periodiek van aard is. Dit is belangrijk wanneer gebruikers geen stabiele verbinding hebben tot een netwerk, wanneer men optimaal gebruik wenst te maken van de beschikbare bandbreedte of eenvoudigweg bij het uitvoeren van een HTTP request. Een voorbeeld is RxJs dat gebaseerd is op de ReactiveX library en volledig gebouwd rond asynchroon programmeren.

Polling : Bij polling (of pulling) wordt op regelmatige intervallen de status van applicatie, databank of datastructuur opgevraagd. Bij polling vraagt de ontvanger voor nieuwe data of updates.

Pushing : Bij push technologie wordt de request voor een transactie geïnitialiseerd door de server.

Microservices Architecture : Microservices is een architectuur voor het opstellen van server-side enterprise applicaties. Daarbij worden alle onderdelen opgebouwd als een set van losgekoppelde en samenwerkende services. Elke service heeft een beperkte functionaliteit die kan worden aangeroepen door andere microservices. Wanneer een microservice dan data nodig heeft die het zelf niet beschikt, dan kan wordt er een andere microservice aangeroepen met een andere verantwoordelijkheid.

Serverless computing : Een andere naam voor Function as a Service (FaaS) is een cloud-computing executie model waarbij de cloud provider het opstarten en stoppen van een functie volledig zelf beheert. Het opstarten van een functie is op basis van een event. een mogelijk event is bijvoorbeeld een API call naar een HTTP endpoint van een functie. Met serverless computing hoeft een developer zich niet bezig te houden met het configureren en beheren van verschillende virtual machines.

2. Inleiding

Het onderzoek zal verschillende methodes voor offline opslag en synchronisatie analyseren en onderzoeken. Het onderzoek zal van elke methode de voor- en nadelen overlopen en de toepassing tonen met behulp van een prototype. In de sectie 'Business Case' wordt de business case van Pridiktiv.care - Into.care toegelicht. Daarna worden in 'Terminologie' de verschillende relevante termen overlopen. In 'Stand van zaken' komt de context en noodzaak van het onderzoek aan bod. Tenslotte volgt de probleemstelling, de onderzoeksvraag en de opzet van de bachelorproef.

2.1 Business Case

De business case van Pridiktiv vormt het vertrekpunt van deze studie en wordt nu nog wat verder toegelicht. De applicatie draait op een smartphone in een woonzorgcentrum. Het woonzorgcentrum beschikt over draadloos internet maar de dekking is niet het volledige gebouw. Hierdoor moet de applicatie ook offline werken. Wanneer verschillende waarden worden geregistreerd met de applicatie (zoals bloeddruk, gewicht, inname medicatie) worden die offline opgeslagen in een IndexedDB. Wanneer het toestel terug online komt, moeten deze waarden worden doorgestuurd naar de backend. Het is essentieel dat er geen data verloren gaat aangezien het gaat om medische data. Het huidige algoritme doet een HTTP call wanneer een waarde wordt ingegeven. Wanneer dit niet lukt, veronderstelt het algoritme dat de applicatie offline is en wordt de data lokaal opgeslagen. Wanneer het toestel terug online komt, dan worden de offline data gesynchroniseerd met de server. De realiteit wijst echter uit dat betrouwbaar internet eerder uitzonderlijk is in een woonzorgcentrum. In deze context is er dus nood aan een 'offline first' - oplossing. De huidige applicatie is een Angular (het vroegere Angular 2) met Ngrx als state container

als een Cordova applicatie. Voor lokale opslag wordt momenteel gebruik gemaakt van Mozilla's localForage library. De backend maakt gebruik van Amazon Web Services met DynamoDB voor de persistentie van de data. Het is de intentie van dit onderzoek om te werken binnen het kader en de restricties van de business case.

2.2 Terminologie

In deze sectie komen de verschillende begrippen met betrekking tot de componenten van het onderzoek, het prototype en tools aan bod. De volledige lijst met begrippen en termen kan u terugvinden onder het hoofdstuk 'Glossarium'. Er is een minimum kennis in verband met software ontwikkeling vereist van de lezer om deze sectie volledig te begrijpen.

2.2.1 Huidige Web APIs voor lokale opslag

Er zijn momenteel 4 APIs voor lokale opslag die ondersteund worden door verschillende browsers. Deze APIs worden ondersteund door de Web Hypertext Application Technology Working Group, aangegeven door WHATWG en de specification van de API wordt dan gestandardiseerd door de World Wide Web Consortium, aangegeven door W3C. Dit proces is belangrijk omdat de APIs dan door de populaire browsers (Chrome, Firefox, Safari en Opera, Internet Explorer en Edge) worden geïntegreerd. Op die manier kunnen webapplicaties gebruik maken van de verschillende APIs. Het is belangrijk om deze APIs kort te overlopen omdat ze steeds gebruikt worden bij de verschillende caching technieken om data lokaal op te slaan.

localStorage en sessionStorage

Deze 2 APIs vallen onder Web Storage of DOM Storage. Wanneer de browser ondersteuning biedt voor Web Storage, zijn beide beschikbaar als een globaal object. Web Storage wordt vaak vergeleken met cookies. Terwijl die vergelijkbaar zijn in functie, verschilt web storage in volgende aspecten:

- Opslag ruimte: Afhankelijk van browser maar meestal 5 MB beschikbare opslagruimte in vergelijking met maar 4 kb voor cookies.
- Client-side interface: Cookies kunnen zowel door server als client side worden gebruikt. Web storage valt exclusief onder client-side scripting.
- Twee verschillende storage areas: localStorage en sessionStorage.
- Een eenvoudigere programmeerbare interface in vergelijking met cookies.

localStorage

Data die in localStorage wordt opgeslagen is persistent tenzij die manueel wordt verwijderd door de gebruiker of applicatie. localStorage is dus een belangrijke kandidaat om data lokaal op te slaan in geval een applicatie offline moet kunnen worden gebruikt. Bij

gevoelige data zoals medische data is het belangrijk om de data te verwijderen uit de localStorage wanneer die niet meer moet worden gecached.

sessionStorage

Het grote verschil met localStorage is dat sessionStorage een vervalttijd heeft en de inhoud van de sessionStorage wordt verwijderd wanneer de sessie vervalt. Een sessie vervalt bijvoorbeeld hij het openen van een nieuw tabblad of browser venster. Refreshen van de browser heeft geen impact op de sessionStorage. sessionStorage laat toe om instances van een webapplicatie te runnen in verschillende browser windows, zonder dat er conflicten optreden.

IndexedDB

IndexedDB is een Web API die gebruikt wordt het opslaan van relatief grote data structuren in browsers. Dankzij indexering is het mogelijk om snel en performant te zoeken in de databank. Net zoals SQL-databanken is IndexedDB een transactional database system. Het grote verschil is echter het gebruik van JSON objecten in plaats van fixed columns tables om data op te slaan, vergelijkbaar met andere NoSQL databanken zoals MongoDB of CouchDB. Onder impuls van Mozilla wordt IndexedDB waarschijnlijk de alternatieve storage standaard voor het web in de nabije toekomst.

Web SQL

Net zoals bij IndexedDB biedt Web SQL toegang tot een databank waar data structuren kunnen worden opgeslagen. Met een SQL variant is het dan mogelijk om queries uit te voeren op de Web SQL database. Momenteel biedt enkel SQLite een database systeem voor Web SQL. W3C werkt momenteel niet meer verder aan de specificatie van Web SQL omdat er te weinig onafhankelijke implementaties zijn van Web SQL.

2.2.2 Scalable Angular Architecture

Bij de start van de ontwikkeling van een Angular applicatie is het belangrijk om na te denken over de architectuur van de applicatie. Moderne SPA-technologieën zoals React en Angular maken gebruik van components. Een component is combinatie van HTML, JavaScript en optioneel CSS. Door alles in componenten onder te verdelen is het eenvoudiger om de applicatie te onderhouden omdat alle relevante data word gegroepeerd. Componenten zelf kunnen 'dumb' zijn als ze enkel data voorstellen of 'smart' wanneer te data opvragen of verwerken. De dataflow tussen componenten is ook belangrijk om een beheersbare applicatie te bouwen. Zo communiceren child components enkel maar met hun parent en interageert een parent met een model dat op zijn beurt communiceert met een 'store' (.cfr 'ngrx store: een state container') aan de hand van acties. Wanneer de state wordt gewijzigd in de store, dan wordt de volledige component tree opnieuw geëvalueerd. Een ander belangrijk aspect van een scalable SPA is het inperken van de com-

municatiemogelijkheden van smart components. Door het gebruik van een extra abstractie layer, zoals een model of sandbox, is het mogelijk om de communicatie te verwerken in de abstractielaag en op die manier een microservice te genereren die gemakkelijk kan worden refactored indien er aanpassingen moeten gebeuren.

2.2.3 Ngrx store: een state container

Ngrx store is een container die de state van de applicatie bijhoudt waarbij performantie en consistentie centraal staan. De ngrx store is de 'single source of truth', verder benoemd als SSOT, voor de applicatie en moet ervoor zorgen dat de data van alle componenten consistent is met elkaar en er geen duplicate data is. Alle updates aan de data worden eerste weggeschreven naar de store en daarna gecachet. In het geval dat de gebruiker toegang heeft tot internet wordt de data verstuurd naar de achterliggende systemen. Indien er geen of slechte internetverbinding is, krijgt de user de gecachte data te zien dankzij de SSOT. De ngrx store werkt met volgende principes:

- De state is een single immutable data structure.
- Actions initialiseren veranderingen in de state.
- Pure functions verwerken actions om een 'nieuwe' state te creëen.
- De state van de store wordt geraadpleegd als een Observable van state die op zijn beurt een observer is van actions.

2.2.4 Reactive programming paradigm

Het reactive programming paradigm is gebouwd rond veranderende data flows, waarbij een applicatie kan reageren op nieuwe of andere input. Dankzij de ReactiveX library is het mogelijk om in verschillende programmeertalen componenten van reactive programming te gebruiken. De library laat toe om asynchronous en event-based applicaties te ontwikkelen, die in real-time data kunnen manipuleren en tonen. Het breidt het Observer design pattern waarbij het mogelijk is om verschillende operators te 'chainen' met elkaar zonder rekening te houden met low level concerns zoals threading, synchronisatie, thread safety, concurrent data structures en blocking I/O. De RxJs library van ReactiveX is volledig geïntegreerd in Angular en ngrx store. Met Promises was het reeds mogelijk om asynchroon te programmeren in JavaScript dus wat zijn de voordelen van RxJs ten opzicht van standaard JavaScript? Een Promise kan maar een single event afhandelen en kan niet worden gecancelled. Een Observable kan worden beschouwd als een stream van objecten waarop verschillende operatoren zoals map, reduce en filter kunnen worden op toegepast en is lazy van natuur.

2.2.5 Offline First

Een stroming binnen web development waarbij offline gebruik als de basis wordt beschouwd van de applicatie. Net zoals bij 'Progressive Enhancement' wordt online functionaliteit beschouwt als een extra laag van features die de applicatie kan aanbieden.

Het idee is gegroeid vanuit de teleurstelling dat 'always online' omwille van technische, geografische, financiële en praktische redenen nog niet haalbaar is voor de nabije toekomst.

2.2.6 Amazon Web Services

Om zo het onderzoek zo dicht mogelijk te laten aansluiten bij de use case van Pridiktiv, wordt er ook gebruikt gemaakt van bepaalde onderdelen binnen AWS. Volgende componenten komen aan bod verder in de bachelor paper:

- Amazon SQS: Simple Queue Service: SQS is een fully managed message queue service die toelaat om te communiceren tussen verschillende componenten en microservices. SQS voorziet 2 varienten. De standard queue biedt de meest snelste en meer performante oplossing maar hanteert geen FIFO-principe. Een FIFO-queue, zoals de naam reeds aangeeft, hanteert wel het FIFO-principe en elke message in de queue wordt altijd maar 1 keer geleverd. Andere AWS services pollen de queue voor nieuwe data.
- Lambda: Met AWS Lambda is het mogelijk om 'functies' uit te voeren in de cloud dus zonder een specifieke server structuur op te bouwen en vormt dus de basis voor de serverless architectuur. De input voor de functie kan worden geleverd door HTTP endpoints maar ook door andere AWS services en is een voorbeeld van een event-driven architectuur. De output kan dan bijvoorbeeld worden gepersisteerd door DynamoDB of teruggestuurd worden naar de client. De verschillende Lambda functies vormen de microservices architectuur in de backend.
- Amazon SNS, Amazon Simple Notification Service: Een push notificatie service die toelaat om berichten te sturen naar mobile clients, email adressen of andere services. Met SNS is het onder andere mogelijk om polling van andere services te vermijden.
- DynamoDB: DynamoDB is een NoSQL key-value database van AWS en wordt gebruikt door de business case voor de data opslag. DynamoDB voorziet geen automatische synchronisatie met clients.

2.3 Stand van zaken

Wanneer een ontwikkelaar de end user van zijn web -of mobiele applicatie wil voorstellen, dan denkt hij vaak aan een user met dezelfde eigenschappen als zichzelf (laatste smartphone, up-to-date besturingssysteem, snelle internet verbinding). De ontwikkelomgeving (snelle desktop/laptop met een betrouwbare en snelle internet verbinding) simuleert amper de omgeving van de end user. Ziet de realiteit er van de end user anders uit. Bepaalde omgevingsfactoren zoals bv. tunnels, trein en vliegtuig kunnen hebben een grote invloed op de betrouwbaarheid van de connectie. Het klassieke client - server model waarbij de client enkel maar als het ware een view is van de data die door de server wordt bijgehouden, is achterhaald, want elke onderbreking in de internet connectie zorgt ervoor dat de applicatie niet meer kan worden gebruikt. Dat is een scenario dat ten alle kosten moet worden vermeden. Meeste mobiele- en webapplicaties hebben twee momenten waarbij er problemen kunnen optreden door de status van de connectie:

1. Client push request naar de server
2. Client pull van server / server push naar client

Afhankelijk van wat de applicatie wil doen, zijn er verschillende opties:

- De gebruiker al dan niet op de hoogte brengen van de veranderingen in de status van de connectie. Bijvoorbeeld: Bij het versturen van bericht, de gebruiker op de hoogte brengen dat het bericht pas wordt verstuurd wanneer de applicatie terug online is.
- Offline client-side creatie en manipulatie van data toelaten aan de hand van caching en wanneer de applicatie terug online
- Uitschakelen of aanpassen van bepaalde features wanneer de applicatie offline is.

Daarnaast kan het zijn dat de server data wil pushen naar de applicatie van de gebruiker en dat verschilt met de data van de applicatie. De applicatie moet de gebruiker dan waarschuwen dat er nieuwere, recentere data beschikbaar zijn en indien nodig, een conflict-resolution tool aanbieden. Momenteel zijn er al enkele frameworks en databank systemen die eenvoudig toelaten om (offline) data te synchroniseren van een web- of mobiele applicatie met de achterliggende databank. CloudBoost, CouchBase en Firebase bieden een all-in-one oplossing. Het nadeel van die oplossing is dat het niet compatibel is met de bestaande backend. Deze opties komen dan ook niet verder aan bod in deze bachelorpaper.

2.4 Probleemstelling en Onderzoeksvragen

Het doel van dit onderzoek is om bij een bestaande Angular applicatie data synchronisatie zo vlot mogelijk te laten verlopen, zonder onderbreking bij de gebruiker terwijl de data consistent blijft. Hoe wordt conflict-resolution opgelost en moet de gebruiker daar zelf een keuze maken of kan de applicatie of achterliggende systemen zelf alle conflicten oplossen? Tenslotte wordt ook de performantie en de schaalbaarheid van de oplossingen onderzocht. Op basis van verschillende tests op het prototype, is het de intentie van het onderzoek om tot een betrouwbare oplossing te komen voor het probleem van de business case. Zoals reeds vermeld moet in het geval van de business case, de applicatie zonder onderbreking gemakkelijk kunnen overstappen van een offline status naar een online status en omgekeerd, zonder dat er daarbij belangrijke data verloren gaat. Het is cruciaal dat deze functionaliteit wordt verwerkt aangezien de dekking van draadloos internet in verschillende woonzorgcentra niet optimaal is.

2.5 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen. In hoofdstuk 4 wordt de opstelling voor het onderzoek van deze bachelorpaper voorgesteld. In hoofdstuk 5 worden de verschillende methodes voor synchronisatie en conflict resolution

toegelicht. In hoofdstuk 6 worden de oplossingen op de verschillende concrete user stories overlopen. In hoofdstuk 7, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein. Indien bepaalde begrippen onduidelijk zijn, vindt u een beknopte uitleg in het het hoofdstuk 1.

3. Methodologie

3.1 Research

De eerste stap in het onderzoek was het in kaart brengen van bestaande synchronisatie technologieën en technieken voor data synchronisatie. Synchronisatie technologieën zoals Firebase, CouchBase en Azure Mobile Apps laten toe om na wat configuratie een applicatie 'offline first' te maken waarbij conflict resolution nagenoeg automatisch verloopt. CouchDB en PouchDB zijn twee andere technologieën waarbij er geen 'all-in-one' oplossing wordt aangeboden. Ze werken enkel in tandem voor een oplossing te bieden naar synchronisatie. Wanneer een synchronisatie framework wordt gebruikt, dan gebruikt de client applicatie een specifieke library die de data lokaal beheert indien de gebruiker offline gaat of de client applicatie de data gewoon wil cachen. Wanneer de gebruiker terug online gaat of de client applicatie de gecachte data wil synchroniseren, dan gebeurt de synchronisatie en conflict resolution volledig automatisch. Voorwaarden om dit te kunnen realiseren is de implementatie van een specifieke library om de lokale data te beheren en het gebruik van de correcte backend frameworks of databases. Door de beperkingen van de business case (.cfr "Beperkingen door business case") was er ook nood aan onderzoek naar synchronisatie patterns. Deze patterns zijn vaak geïntegreerd in de vermelde synchronisatie technologieën.

3.1.1 Beperkingen door business case

Er zijn heel wat oplossingen voor synchronisatie van offline data out-of-the-box maar die hebben bepaalde voorwaarden die niet voldoen aan de constraints van de business case. De use case van Pridiktiv laat immers geen volledige refactor van de backend toe. Men heeft gekozen voor DynamoDB voor de persistentie van de data en helaas biedt DynamoDB

geen automatische synchronisatie toe. Hierdoor moeten ook synchronisatie methodes worden onderzocht.

3.2 Protoyping

Om de use case van Pridiktiv te simuleren, wordt er gebruik gemaakt van een prototype dat specifieke scenario's van gebruikers kan simuleren. Net zoals de applicatie van Pridiktiv, is het prototype een Angular applicatie met Ngrx state container. Configuratie van de offline data opslag staat niet vast, zodat het gemakkelijk kan worden aangepast in de testopstelling. Als backend wordt er gebruik gemaakt van een serverless architectuur met AWS Lambda.

3.3 Testen van prototype

Op basis van de research en het prototype, worden enkele test scenario's geselecteerd voor data synchronisatie. Het prototype bestaat uit 2 onderdelen. In de client-side applicatie worden verschillende scenario's overlopen die ook in de business case van Pridiktiv zitten. Het server-side gedeelte bevat net als de business case verschillende microservices die elk een scenario omvatten.

3.4 Conclusie

In het laatste onderdeel van het onderzoek wordt er een synthese gevormd met de resultaten van de testen op het prototype. Die synthese vormt de basis voor de aanbevelingen voor de business case en is het resultaat van het onderzoek naar verschillende synchronisatie opties.

4. Opstelling

In dit hoofdstuk bespreken we de opstelling van zowel de client als van de server. De gebruikte scenario's maken deel uit van verschillende use cases opgesteld door Pridiktiv bij het online en offline gebruik van de applicatie. Verder in het hoofdstuk 'Onderzoek' worden verschillende scenario's belicht en de resulaten van het onderzoek.

4.1 Client side

4.2 Server side

5. Synchronisatie patterns

In dit hoofdstuk worden enkele synchronisatie methodes en technieken besproken die data kan synchroniseren met de databank. Indien van toepassing, wordt telkens een concrete situatie van de use case van Pridiktiv gebruikt ter illustratie. Op het einde van dit hoofdstuk worden ook nog andere technieken besproken die niet meteen kunnen worden onderverdeeld onder een specifiek pattern.

5.1 Read-Only Data

Dit data synchronisatie patroon wordt gebruikt wanneer de end user enkel maar data moet kunnen opvragen terwijl de applicatie offline is en die data niet hoeft te manipuleren of de manipulaties op die data niet belangrijk genoeg zijn om te persisteren. Bij Read-Only Data is de richting van het dataverkeer unidirectioneel, van server naar end-user applicatie. Het Read-Only pattern hanteert volgende logica:

1. De end-user (client) vraagt de data op van de server. De server is de SSOT en houdt alle data bij. Die data kan wijzigen wanneer de end-user bijvoorbeeld offline is.
2. Server retourneert de data. De opgevraagde data wordt lokaal opgeslagen.
3. Alle manipulaties op de data worden geblokkeerd en worden nooit doorgegeven aan de server. De client kan dus enkel maar GET HTTP requests sturen voor de data op te vragen.
4. Bij synchronisatie wordt de oude data lokaal verwijderd en vervangen door de nieuwe data.
5. De applicatie controleert op regelmatige tijdstippen de data.

Een voorbeeld van het Read-Only pattern van in de use case van Pridiktiv is het opvragen

van de patiëntenlijst. Die lijst kan worden gewijzigd door de hoofdverpleger in het dashboard maar niet in de applicatie zelf. Wanneer de applicatie offline is, kan eenvoudig worden verder gewerkt met de applicatie. Indien er een nieuwe patiënt wordt toegevoegd, dan is die vanaf de volgende synchronisatie zichtbaar.

5.2 Read-Only Data Optimized

Het Read-Only Data Optimized pattern is identiek aan het Read-Only Data pattern maar met 1 verschil. Bij de synchronisatie wordt enkel de data opgevraagd die gewijzigd is en niet alle data. Dit kan op verschillende manieren worden geïmplementeerd. Er kan een timestamp worden bijgehouden van de laatste wijziging of een version number die incrementeert bij elke wijziging. Op basis van de vergelijking tussen de bestaande data en de data op de databank, kan de applicatie al dan niet beslissen om de lokale data te updaten. De richting van het dataverkeer is bidirectioneel, wat ook verschilt met de standaard implementatie van het Read-Only pattern. Omdat de server eerst moet worden gevraagd of er al dan niet data is gewijzigd, moet de applicatie ook kunnen communieren met de server.

5.3 Read/Write Data Last Write Wins

In dit pattern gaat de server er van uit dat writes altijd in de juiste volgorde worden uitgevoerd en de laatste write die naar de databank wordt gestuurd ook de werkelijke laatste wijziging is van de client applicatie. Er wordt geen conflict resolution uitgevoerd. Het pattern blinkt uit wanneer er enkel wordt toegevoegd en de data nooit wordt gemanipuleerd. In de use case van Pridiktiv kan dit worden toegepast bij bijvoorbeeld het toevoegen van notities bij een patient. De notities van een patient worden niet gewijzigd waardoor er geen data kan worden overschreven.

5.4 Read/Write with Conflict Detection

Het Read/Write with Conflict Detection pattern is het meest complexe waarbij verschillende end-users dezelfde data wijzigen op het moment dat de applicatie offline is. Bij dit pattern wordt er gesproken van multi-way synchronisatie waarbij een applicatie zowel de data van de server kan updaten en dat de server op zijn beurt alle andere apparaten moet updaten. Een mogelijke flow van het proces zou er als volgt kunnen uitzien:

1. De server database houdt alle data bij
2. De applicatie houdt lokaal een subset van de data bij die kan worden gewijzigd.
3. Bij synchronisatie wordt de aangepaste data die lokaal wordt opgeslagen naar de server en omgekeerd.
4. In de server wordt de data aangepast en alle conflicting changes worden geregistreerd voor verdere behandeling

5. Vraagt de gebruiker voor conflict resolution of de applicatie kan zelf het conflict oplossen en de data aanpassen.

Een voorbeeld uit de use case van Pridiktiv die gebruik zou kunnen maken van het Read/Write with Conflict Resolution pattern is de opvolging bij wondzorg. Wanneer een end-user een bepaalde wijziging aanbrengt in het dossier van de patient, is het belangrijk dat die data niet verloren gaat indien een andere end-user ook het wondzorg dossier van een patient aanpast. Het is belangrijk om conflict resolution 'achter de schermen' op te lossen om op die manier ervoor te zorgen dat er geen aanpassingen verloren gaan.

6. Onderzoek

In dit hoofdstuk wordt een overzicht gemaakt van de verschillende user stories, scenario's en oplossingen die worden onderzocht.

7. Conclusie

!!!!!! TODO aanvullen met relevante info na onderzoek, laatste onderdeel om te schrijven !!!!!!!

Lijst van figuren

Lijst van tabellen