2015•2016
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef

Ontwikkeling van een API voor het stellen van dynamische vragen met behulp van de smartphone

Promotor :
dr. Kris AERTS

Promotor :
Prof. dr. LARS GRIETEN

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

Michiel Deprez
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

KU LEUVEN | universiteit hasselt

KU LEUVEN | universiteit hasselt

2015•2016

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterproef

Ontwikkeling van een API voor het stellen van dynamische vragen met behulp van de smartphone

Promotor :
dr. Kris AERTS

Promotor :
Prof. dr. LARS GRIETEN

Michiel Deprez

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

universiteit hasselt | KU LEUVEN

Woord vooraf

Na mijn opleiding tot bachelor, behaald aan de KHLim, rond ik mijn opleiding tot industrieel ingenieur Elektronica-ICT af aan de gemeenschappelijke opleiding van UHasselt en KU Leuven. Het slot van deze opleiding vormt een masterthesis voor de Future Health afdeling binnen het ZOL.

Tijdens deze zeer leerrijke en interessante periode heb ik heel wat technische ervaring opgedaan binnen de bedrijfswereld.

Graag zou ik de volgende personen willen bedanken voor hun hulp en goede begeleiding om deze masterthesis tot een goede einde te brengen:

- Mijn bedrijfspromotor Prof. Dr. Lars Grieten, voor de begeleiding en het vertrouwen in mij;
- Mijn interne promotor Dr. Kris Aerts, voor het opvolgen van mijn project en het beantwoorden van al mijn vragen;
- De medewerkers van Mobile Health Unit, in het bijzonder Thijs Vandenberk, Wilco Waaijer en Thomas Reyskens. Dankzij hun verdiepende kennis en inzicht kreeg ik voldoende feedback om deze masterproef te realiseren;
- De mede-stagiairs met wie ik heel wat prettige en leerrijke momenten heb mogen beleven en vlot heb samengewerkt;
- Mijn ouders en vrienden voor hun onvoorwaardelijke steun.

Michiel Deprez

Zonhoven, juli 2016

Inhoudstabel

Woord vooraf ... 1
Lijst van figuren ... 5
Verklarende woordenlijst en afkortingen ... 7
Abstract ... 9
Summary ... 11
1. Inleiding ... 13
1.1. Situering ... 13
1.2. Probleemstelling ... 13
1.3. Doelstelling ... 13
1.4. Methode ... 14
2. Literatuurstudie ... 17
2.1. Basic authentication met TLS: ... 17
2.2. API Keys: ... 19
2.3. Custom protocols: ... 20
2.4. API Token Authentication: ... 20
2.5. OAuth: ... 22
2.5.1. OAuth1.0: ... 22
2.5.2. OAuth2.0: ... 23
2.5.3. OAuth1.0 vs. OAuth2.0: ... 24
3. Ontwikkelingsfase ... 27
3.1. PHP framework: Laravel ... 27
3.1.1. Composer ... 27
3.1.2. Artisan ... 27
3.1.3. MVC: Model View Controller ... 27
3.1.4. Routing ... 29
3.2. Database ... 31
3.2.1. Migrations ... 31
3.2.2. Foreign Keys ... 32
3.2.3. Databasemodel dynamische vragenlijst ... 34
3.3. API voor DHARMA ... 35
3.3.1. RESTful resource controllers ... 35
3.3.2. Routes ... 35
3.3.3. QuestionController: show ... 37
3.3.4. QuestionController: store ... 39
3.3.5. MeasurementController ... 40

3.3.6. RequestController 41
3.3.7. AuthenticatieController 41
3.4. Pushnotifications 43
3.4.1. Android: GCM 44
3.4.2. iOS: APNs 44
3.5. Schedules 45
3.6. Beveiliging van API: OAuth2.0 47
3.6.1. Keuze voor OAuth2.0 47
3.6.2. Library 47
3.6.3. Grant-types 47
3.6.4. Set-up 48
3.6.5. Authenticatie 49
3.6.6. Aanroepen API 50
4. Resultaten 51
4.1. API voor DHARMA 51
4.2. Verzenden van pushnotificaties 51
4.3. Gebruik van schedules 51
4.4. Beveiliging met OAuth2.0 51
5. Besluit 53
5.1. Persoonlijke ervaring 53
5.2. Conclusie en mogelijke uitbreiding 53
Bibliografie 55

Lijst van figuren

Figuur 1: Basic authentication met TLS (Badri, 2012) 18
Figuur 2: Api token authentication (Sevilleja, 2015) 21
Figuur 3: werken Oauth1.0 (Mayko, 2015) 22
Figuur 4: werking OAuth2.0 (Mayko, 2015) 24
Figuur 5: Laravel model deel 1 28
Figuur 6: Laravel model deel 2 28
Figuur 7: MVC met routing (Architecture of Laravel Applications, sd) 29
Figuur 8: migratie database toevoegen 31
Figuur 9: Lijst van migraties 31
Figuur 10: Tabel aanmaken via migratie 31
Figuur 11: Migratie down functie 32
Figuur 12: Foreign keys toevoegen 32
Figuur 13: Down functie voor foreign keys 33
Figuur 14: Database model vragenlijst 34
Figuur 15: API routes 36
Figuur 16: API routes group 36
Figuur 17: RequestController show 37
Figuur 18: voorbeeld lijst van vragen 38
Figuur 19: voorbeeld van antwoorden verstuurd naar server 39
Figuur 20: QuestionController store 40
Figuur 21: voorbeeld van metingen verstuurd naar de server 40
Figuur 22: MeasurementController store 41
Figuur 23: RequestController show 41
Figuur 24: gegevens voor aanvraag access_token 41
Figuur 25: AuthenticateController oauth 42
Figuur 26: verzenden push notificatie 43
Figuur 27: config file push notificaties 43
Figuur 28: werking GCM (Kiran V S, sd) 44
Figuur 29: werking APNs (Apple Push Notification Service, sd) 44
Figuur 30: Set-up OAuth2.0 48
Figuur 31: resource owner credentials grant-type (Degasperi, sd) 49
Figuur 32: gegevens voor aanvraag access_token met resource owner credentials grant-type 49
Figuur 33: verkregen access_token en refresh_token na authenticatie 50
Figuur 34: refresh token grant-type 50
Figuur 35: gegevens voor aanvraag nieuw access_token met een refresh_token 50

Verklarende woordenlijst en afkortingen

Protocol: legt de regels vast tussen communicatie van 2 systemen. Zo zijn er meerdere beveiligingsprotocollen beschikbaar voor het beveiligen van de API.

Base64 encoded: data wordt omgezet naar een ASCII string formaat. De omgezette data is niet leesbaar zonder deze te decoderen.

data integriteit: synoniem voor betrouwbaarheid. Hoe beter de integriteit hoe veiliger de data.

API: application programming interface, is een verzameling definities waarmee een computerprogramma of applicatie kan communiceren met een ander programma.

Hash: een opeenvolging van random karakters zoals verschillende letters.

Request: een verzoek naar de API voor bijvoorbeeld een vragenlijst te ontvangen.

ZOL: ziekenhuis Oost-Limburg.

DHARMA: Digitaal Health Research platform Mobile health unit.

TLS: Transport Layer Security.

H-MAC: Keyed-hash message authentication code

HTTPS: HyperText Transfer Protocol Secure.

PHP: Hypertext Preprocessor.

REST: representational state transfer.

ORM: object-relational mapper.

GCM: Google cloud messaging.

APNs: Apple push notification service.

Abstract

De afdeling Future Health binnen het ziekenhuis Oost-Limburg (ZOL) is gespecialiseerd in onderzoek en ontwikkeling naar nieuwe en innoverende technologieën voor de medische sector. Een project binnen deze afdeling is DHARMA, Digitaal Health Research platform Mobile health unit. Eén van de geïmplementeerde studies op DHARMA is Premom, waarbij zwangere vrouwen met risico op pre-eclampsie in de thuissituatie worden opgevolgd door middel van medicale devices, zoals: een bloeddrukmeter en weegschaal. Bij een overschrijding van vooringestelde grenswaarden is het belangrijk dat onderzoekers extra vragen kunnen stellen aan de patiënt om na te gaan wat de oorzaak kan zijn. Het doel van deze masterproef is een API te ontwerpen en realiseren die bovendien geïntegreerd is in DHARMA. De vragenlijsten die online worden samengesteld en verstuurd kunnen met behulp deze koppeling worden ingevuld van via de smartphones. Bijkomend evalueert de masterproef methodes voor de beveiliging van het systeem.

Het technisch project van de masterproef is opgedeeld in twee delen. Eerst is er met Laravel een API ontwikkeld voor de communicatie tussen de mobiele applicatie en de server. Vervolgens is de beveiliging van het systeem geanalyseerd en uitgebreid zodat de data van de patiënt niet door iedereen kan worden opgehaald.

Met behulp van dit systeem wordt het voor de onderzoekers mogelijk om patiënten op een uitgebreidere manier op te volgen. Wanneer de patiënt klaar is met deze lijst in te vullen, wordt deze terug verstuurd naar de onderzoeker voor analyse.

Summary

The Future Health department within the hospital East-Limburg (ZOL) specializes in research and development for new and innovative technologies for the medical sector. A project within this research group is DHARMA, Digital Health Research Platform Mobile health unit. One of the implemented studies within Dharma is Premom, where pregnant women with risk of pre-eclampsia are followed at home through medical devices, such as: a blood pressure monitor, a step counter and an scale. When exceeding pre-set limited values, it is important that researchers can ask extra questions at the patient in order to verify what can be the cause. The purpose of this master's thesis is to design and implement an API which is also integrated into DHARMA. The questionnaires, which are submitted and sent online, can be completed with this link using smartphones. Additionally the master's thesis evaluates and integrate methods for the security of the system.

The technical project of the thesis is divided into two parts. First up an API is developed with Laravel for communication between the smartphones and the server. Next, the security of the system is analyzed and expanded so that the data of the patient cannot be retrieved by anyone.

With the aid of this system it is possible for researchers to monitor patients on a more extensive way. When the patient has completed the list it is sent back to the investigator for analysis.

1. Inleiding

1.1. Situering

De afdeling Future Health binnen het Ziekenhuis Oost Limburg (ZOL) is gespecialiseerd in onderzoek naar nieuwe en innoverende technologieën voor de medische sector met als doel de kwaliteitszorg van de patiënt binnen het ziekenhuis te garanderen en te verbeteren. Eén van deze projecten is DHARMA. Dit project heeft als doel om gegevens van patiënten te verzamelen in de thuiszorg door middel van wearable devices ban het merk Withings. Een van deze toestellen is de Withings bodyscale. De patiënt krijgt een Withings bodyscale ter beschikking die wordt geïnstalleerd via een computer en wordt gekoppeld met een Wifi-netwerk. Wanneer metingen worden gedaan wordt de meetdata, via de Wifi-connectie, online opgeslagen. De onderzoeker of dokter kan deze meetdata online raadplegen.

Binnen deze thesis wordt als doelstelling gesteld het uitbreiden van het online platform genaamd DHARMA. DHARMA is ontwikkeld binnen de afdeling Future Health waar reeds meerdere studenten aan hebben gewerkt (Pelssers, juni 2015). Hierin is het mogelijk voor dokters en onderzoekers om patiënten op te volgen via medicale devices.

1.2. Probleemstelling

Het is noodzakelijk dat onderzoekers de patiënten op een gebruiksvriendelijke en vlotte manier kunnen volgen zodat het mogelijk is om de patiënt een kwalitatieve zorg te verlenen en op te treden indien nodig. Zo is het bijvoorbeeld mogelijk om zwangere vrouwen op te volgen in hun dagelijkse leven en de verzamelde gegevens te verwerken. Om dit op een snelle en gebruiksvriendelijke manier te doen werd een online platform ontwikkeld genaamd DHARMA. Hierdoor kunnen de gegevens online worden verzameld en moeten vragenlijsten niet meer met pen en papier worden ingevuld. Het is echter noodzakelijk voor de correcte interpretatie van patiëntgegevens dat de onderzoekers deze vragenlijsten op een vlotte manier kunnen samenstellen en verwerken. Zo is het nodig dat wanneer de patiënt een bepaald antwoord geeft er vervolgvragen komen om zo per patiënt de juiste informatie te verzamelen.

1.3. Doelstelling

Door middel van medicale devices wordt meetdata van patiënten (gewicht, totaal aantal stappen van de dag, hartslag) online opgeslagen. Dokters en onderzoekers kunnen zo patiënten opvolgen buiten de muren van het ziekenhuis. Door de meetdata te analyseren wordt gecontroleerd hoe de patiënt zich stelt en indien nodig tijdig in te grijpen. Het doel van dit project is DHARMA uit te breiden zodat het mogelijk is voor dokters en onderzoekers om een vragenlijst te versturen, wanneer er een overschrijding is van vooraf ingestelde paramaters, naar een patiënt. De patiënt moet deze vragenlijst op een applicatie kunnen invullen waarna deze wordt teruggestuurd naar de server.

Vooropgestelde doelstellingen voor deze thesis zijn: ten eerste, in overleg met andere ontwikkelaars, een databasemodel voorstellen voor de uitbreiding van DHARMA. Hierbij moet rekening worden gehouden welke tabellen nodig zijn om aan de vereisten van de onderzoekers te voldoen, zodat ze op een juiste manier een dynamische vragenlijst kunnen samenstellen en deze koppelen aan een patiënt. Wanneer deze patiënt een vragenlijst invult en terugstuurt wordt deze in de database in eerste instantie gekoppeld aan de vooropgestelde vragenlijst. Verder worden de beantwoorde vragen gekoppeld met het antwoord dat de patiënt doorstuurde. Het koppelen wordt gedaan aan de hand van id's. Elke vragenlijst, vraag, antwoord, … worden aangeduid met een id binnen de kolom.

Ten tweede wordt er een keuze gemaakt met welk framework de code wordt geschreven voor deze database te gebruiken. Hiermee wordt rekening gehouden dat er een online interface nodig is voor de lijsten samen te stellen. Collega student Maarten Gregoire heeft uitgebreid onderzoek gedaan welk framework de beste keuze is (Gregoire, juni 2016).

Vervolgens is het nodig om de communicatie tussen de interface en de database op een correcte en gebruiksvriendelijke manier te programmeren met PHP en het gekozen framework.

Ten slotte is het nodig om de communicatie met de applicatie op een snelle en beveiligde manier te ontwikkelen. Hiervoor moet er een keuze worden gemaakt met welk framework de API's worden geschreven en met welk protocol deze worden beveiligd. Deze beveiliging is belangrijk omdat de antwoorden van de patiënten privé zijn. De patiënt moet zich eerst authentiseren vooraleer toegang tot de applicatie wordt verkregen. Wanneer de patiënt een request doorvoert moet te allen tijde geweten zijn welke patiënt de request verstuurt en of deze patiënt machtiging heeft tot deze request. Patiënten mogen geen gegevens van andere patiënten kunnen ophalen of vragen van andere patiënten kunnen invullen en versturen naar de database.

1.4. Methode

Voor de keuze van het gebruikte framework zijn er twee mogelijkheden onderzocht, namelijk CodeIgniter welk reeds gebruikt wordt en Laravel. Na een onderzoek is er gekozen voor Laravel, hiermee is een keuze gemaakt voor de toekomst (Gregoire, juni 2016). Dit wil zeggen dat het reeds ontwikkelde project volledig wordt herschreven door medewerkers binnen de afdeling Future Health. Door de combinatie van MVC(model-view-controller) en de verschillende functies en libraries waarover Laravel reeds beschikt is er minder code nodig voor het project te programmeren en blijft alles zeer gebruiksvriendelijk en overzichtelijk. Dit zorgt ervoor dat het uitbreiden van het project op een snellere manier gebeurt.

Doordat de communicatie via de API's op een zeer veilige manier moet gebeuren wordt er onderzoek gedaan naar de beste manier om dit te doen (Stormpath, 2013):

- Basic authentication met TLS 2.1: dit is een makkelijk implementeerbaar protocol doordat het in de meest voorkomende gevallen kan worden uitgevoerd zonder extra bibliotheken.
- Application programming interface key (API key) 2.2: een code wordt doorgegeven door het computerprogramma waarmee connectie met de API wordt aangevraagd. Hierdoor kan de patiënt worden geïdentificeerd en wordt er nagegaan hoe en waarvoor de API wordt gebruikt. Zo kan er misbruik van de API worden voorkomen.

- Custom protocols 2.3: tenzij er een volledige kennis is en alle fijne kneepjes van ‘cryptographic digital signatures’ gekend zijn kan dit protocol best vermeden worden.
- API Token Authentication 2.4: bij deze methode voegt de patiënt een api-token toe aan het einde van de URL om zo hun verzoek te authentiseren.
- OAuth protocol 2.5: Is een Token-based protocol welk voorkomt dat de informatie van de patiënt wordt blootgesteld aan iedereen. Er zijn twee versies van OAuth, OAuth1.0 en OAuth2.0, elk met zijn sterke en zwakke punten.

2. Literatuurstudie

In deze moderne tijden hebben hackers de vaardigheid om privégegevens, handelsgeheimen en zelfs persoonlijke data bloot te stellen. Hierdoor is het noodzakelijk om de beveiliging van de API te versterken volgens 4 fundamenten van de API-beveiliging namelijk: authenticatie, autorisatie, federatie, en de delegatie (Sandoval, 2015).

- **Authenticatie:** In de praktijk kan dit beveiligingsniveau verschillende vormen aannemen. Zo kan een gebruiker zich aanmelden met een authenticatie service die alleen een gebruikersnaam en wachtwoord vereist. Dit kan echter worden uitgebreid door bijvoorbeeld een eenmalig gegenereerde token mee te sturen;
- **Autorisatie:** terwijl authenticatie de gebruiker controleert, bepaalt autorisatie het toegangsniveau dat ze moet worden verleend. Doordat de ontwikkelaar een hoger toegangsniveau vereist dan de consument, kan een consument slechts een klein deel van de oproepen of functies gebruiken. Dit niet toepassen maakt het systeem kwetsbaarder voor hacks en ongeautoriseerde toegang;
- **Federatie:** hierbij wordt de gebruiker de mogelijkheid gegeven om dezelfde set credentials over meerdere diensten te gebruiken. Doordat de authenticatie plaats vindt in één enkel domein, kunnen andere veiligheidsdiensten dit primaire domein vertrouwen en de authenticatie hergebruiken. Authenticatie en identiteitsgegevens worden doorgegeven tussen deze partijen met behulp van tokens. Wanneer er een breuk optreedt in de Identity Provider, kunnen de vertrouwende partijen het eerder geplaatst vertrouwen in trekken;
- **Delegatie:** toegang en rechten worden gegeven aan geautoriseerde gebruikers waar wordt ingesteld welke handelingen deze gebruiker kan uitvoeren en tot welke data ze toegang hebben.

Voor de systeembeheerder zijn er verschillende protocollen die de implementatie van deze vormen van beveiliging beschikbaar maken. Deze protocollen worden geanalyseerd om tot een conclusie te komen welk het beste kan worden gebruikt binnen dit systeem.

2.1. Basic authentication met TLS:

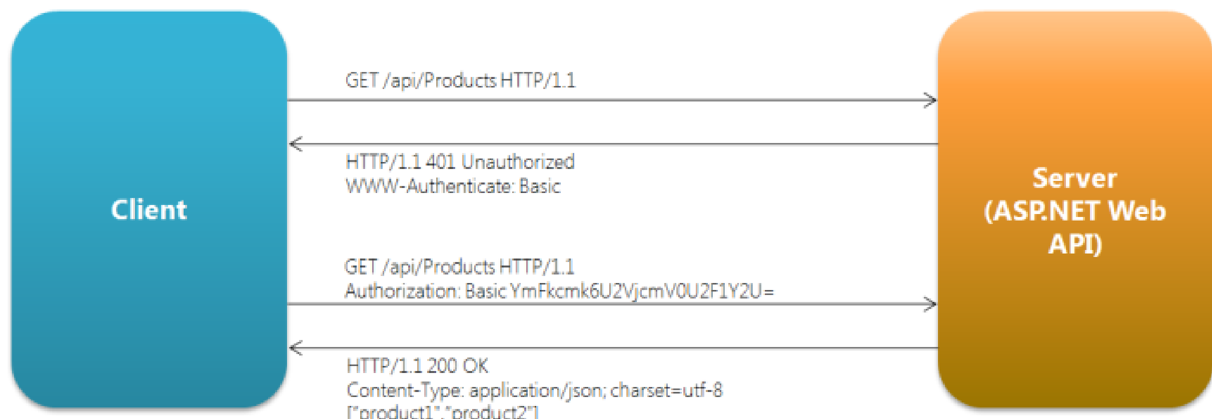
Het Transport Layer Security protocol (TLS) is een evolutie van het Secure Sockets Layer protocol (SSL) en heeft het grotendeels vervangen. Beide zijn encryptie-protocollen die de communicatie tussen computers beveiligen (Rouse, sd). Het grote verschil tussen TLS en SSL is dat TLS een gestandaardiseerd MAC (H-MAC) implementeert waardoor het in tegenstelling tot SSL met elke hash-functie werkt. Keyed-hash message authentication code (H-MAC) wordt gebruikt voor zowel de data integriteit en het authentiseren van het bericht waarmee bevestigd wordt of het bericht afkomstig is van de vermelde afzender (McKinley, 2003).

TLS is samengesteld uit 2 delen, het TLS Record protocol en het TLS Handshake protocol. Het Record protocol zorgt voor een veilige connectie terwijl het Handshake protocol ervoor zorgt dat er authenticatie mogelijk is tussen cliënt en server (Badri, 2012).

Het basic authentication protocol is een makkelijk implementeerbaar protocol doordat het in de meest voorkomende gevallen kan worden uitgevoerd zonder extra bibliotheken. Het probleem met dit protocol is, zoals “basic” doet vermoeden, dat de beveiliging heel basis is en de laagste beveiligingsopties biedt van de meest voorkomende protocollen. Er zijn geen geavanceerde opties bij het gebruik van dit protocol. Er wordt een username met een Base64 encoded passwoord doorgestuurd voor in te loggen (Stormpath, 2013).

De nadelen van dit protocol zijn (AviD, 2010):

- Het wachtwoord wordt Base64 encoded verzonden, dit kan makkelijk worden omgezet naar een gewoon tekstformaat;
- Voor elk request wordt het wachtwoord verzonden wat de kans op onderschepping vergroot doordat het wachtwoord zo veel wordt doorgestuurd;
- Het wachtwoord wordt gecached, op verzoek van de gebruiker kan het wachtwoord permanent worden opgeslagen in de browser wat tot extra risico’s kan leiden.
- Eens ingelogd kan er niet meer worden uitgelogd.



Figuur 1: Basic authentication met TLS (Badri, 2012)

Figuur 1 toont de werking van dit protocol. In eerste instantie verzendt de gebruiker een request zonder autorisatie. De server zal antwoorden met 401 Unauthorized. Hierop zal de gebruiker opnieuw een request versturen maar dit maal met een autorisatie header. In deze header zit het user id en het wachtwoord waarmee de gebruiker kan inloggen.

Door gebruik te maken van TLS over een HTTPS verbinding zal het eerste nadeel, namelijk het onbeveiligd doorsturen van een Base64 encoded wachtwoord, opgelost worden. HTTPS zal er voor zorgen dat het wachtwoord wordt beschermd in transit. Deze beveiliging alleen is echter niet genoeg. Wanneer de gegevens van een gebruiker worden gehacked kan de hacker elke request uitvoeren zonder hiervoor een extra veiligheidsmaatregel te overbruggen. Hierbij is een extra beveiliging nodig waardoor dit protocol niet geschikt is voor dit project.

2.2. API Keys:

Voor authenticatie bij een API wordt er normaal gebruik gemaakt van een combinatie van gebruikersnaam en wachtwoord voor het inloggen. Deze combinatie wordt vaak hergebruikt op meerdere sites, wat dus een risico geeft op het onderscheppen van de gegevens. Als ze op één site worden achterhaald kan dit ook op meerdere. Een alternatief hiervoor is het gebruik van een API Key.

Bij een API Key wordt een sleutel doorgegeven door het computerprogramma waarmee connectie met de API wordt aangevraagd. Hierdoor kan de gebruiker worden geïdentificeerd en wordt er nagegaan hoe en waarvoor de API wordt gebruikt. Zo kan er misbruik van de API worden voorkomen. De API Key wordt bij elke request meegegeven. Op basis van deze key weet de server wie de request uitvoert doordat de key slechts aan één patiënt is gekoppeld (Sandovel, 2015).

Voordeel van API Keys ten opzicht van username met passwoord (Hazlewood, 2012):

- **Entropie:** API Keys zijn willekeurig gegenereerde tekenreeksen. De entropie wordt aanzienlijk groter naarmate het aantal tekens dat wordt gebruikt stijgt. Hoeveel te meer tekens hoeveel te beter de entropie. Hierdoor zijn ze voor hackers veel moeilijker te achterhalen;
- **Reset van wachtwoord:** wanneer een patiënt het wachtwoord verandert zal elk stukje software dat communicatie maakte met de API door middel van dit wachtwoord mislukken. Dit kan een enorm onhandige en productie-bedreigende bijwerking zijn;
- **Onafhankelijkheid:** doordat API Keys onafhankelijk zijn van de master credentials van het account kunnen ze op wil herroepen of aangemaakt worden. Ook kunnen meerdere API Keys aan één account worden toegekend. Dit maakt het mogelijk om sleutels te verwijderen wanneer er een vermoeden is dat deze gekraakt zijn of om bijvoorbeeld iedere maand de sleutels te veranderen;
- **Beperkte blootstelling:** API Keys worden nooit blootgesteld in de user interface. De sleutel wordt één keer vrijgegeven wanneer deze wordt aangemaakt en opgeslagen in een bestand dat slechts leesbaar is door de patiënt zelf. Dit bevordert een no-sharing beleid en maakt het moeilijker om de sleutels te verwerven;
- **Traceerbaarheid:** een API Key wordt nooit in de user interface getoond. Voor de API Key te bemachtigen is er een download nodig die door een persoon wordt gestart. Hierdoor is het mogelijk om bij te houden wie de download heeft gestart en zo te achterhalen wie de API Key heeft bemachtigd.

De patiënt moet de mogelijkheid hebben om in te loggen met gebruikersnaam en wachtwoord waardoor dit protocol niet geschikt is voor het gebruik binnen dit project.

2.3. Custom protocols:

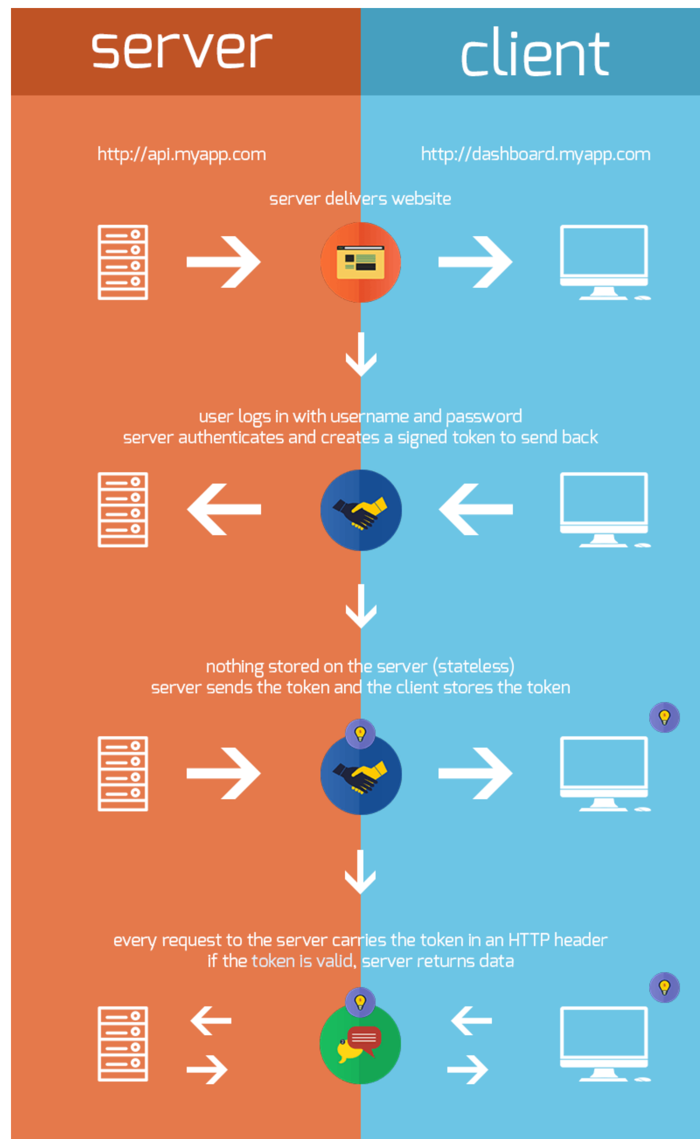
Ontwerpen van een custom protocol is een goede manier voor een beveiliging te bekomen die perfect is voor het gewenste systeem. Het is echter zeer complex om zo een protocol uit te werken. Tenzij er een volledige kennis is en alle fijne kneepjes van 'cryptographic digital signatures' gekend zijn kan dit protocol best vermeden worden. Doordat het protocol eigenhandig wordt gemaakt door een ontwikkelaar zal enkel de ontwikkelaar zelf de complexiteit achter de geïmplementeerde beveiliging verstaan. Hierdoor zal niemand anders in staat zijn om het protocol op een eenvoudige manier te gebruiken. Wanneer toch de keuze wordt gemaakt om een custom protocol te ontwerpen is het aangeraden om een library te ondersteunen voor het implementeren te vergemakkelijken (Stormpath, 2013).

2.4. API Token Authentication:

bij deze methode voegt de gebruiker een api-token toe aan het einde van de URL, of aan de header, om zo een request te authentiseren. Token based authentication is overal op het web aanwezig. Tokens zijn de beste manier om authenticatie af te handelen voor meerdere gebruikers. Organisaties zoals Facebook, Twitter, Google+, GitHub, en zo veel meer gebruiken tokens.

Gegevens van de gebruiker worden niet opgeslagen op de server of in een sessie. Dit concept zorgt dat veel problemen verdwijnen die komen kijken bij het opslaan van informatie op de server. Bij elke request is de token (die de gebruiker krijgt bij het inloggen) nodig, deze token wordt in de http-header meegestuurd waardoor een stateless http request wordt behouden. Een stateless protocol is een communicatieprotocol dat ervoor zorgt dat elke request als een onafhankelijke request wordt verwerkt ongeacht vorige requests (Stateless protocol, sd).

Afbeelding 2 toont het principe tussen server en client. De patiënt authentiseert zichzelf gebruik makend van een username en een wachtwoord. Indien de authenticatie correct verloopt, krijgt de gebruiker als response een api_token. Deze api_token wordt bij elk request meegestuurd waardoor de patiënt wordt herkend door de server. Als de api_token correct is bij de patiënt die de request verstuurt wordt er een response gegeven met de data waar de patiënt om vroeg (Sevilleja, 2015).



Figuur 2: Api token authentication (Sevilleja, 2015)

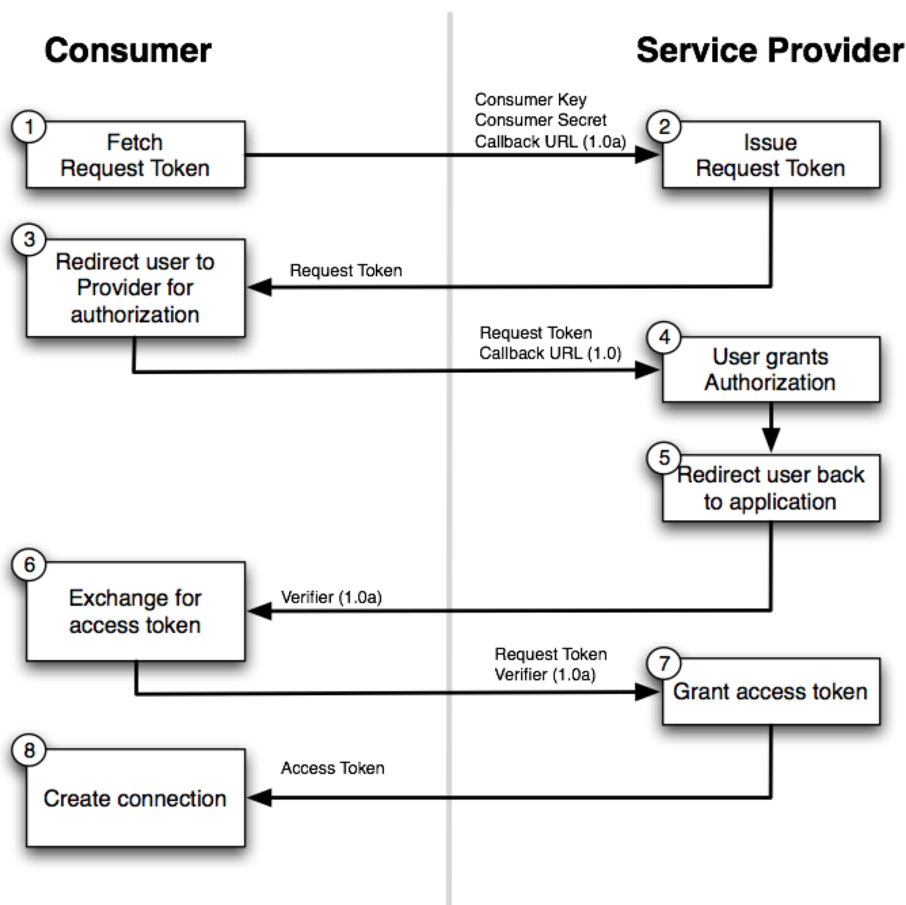
De combinatie tussen het eenmalig inloggen met de gebruikersgegevens waarna een api_token wordt verkregen, is een goede start voor dit project. Het nadeel is echter dat de beveiliging niet verder kan worden uitgebreid.

2.5. OAuth:

OAuth is een Token-based protocol. Door middel van een access_token, welke wordt bekomen na het inloggen, wordt er toegang gegeven tot data van de server. Er zijn twee versies van OAuth, elk met zijn sterke en zwakke punten. Deze zullen worden besproken in 2.5.3.

2.5.1. OAuth1.0:

OAuth1.0 is een veel gebruikt, getest, veilig en signature-based protocol. Over het algemeen wordt aangenomen dat OAuth1 veiliger maar minder flexibel en complexer is dan OAuth2.0. Het is gebaseerd op het hebben van gedeelde geheimen tussen de gebruiker en de server die gebruikt worden om een signature te ontwerpen. De gebruiker laat vervolgens de server toe om de authenticiteit van API-verzoeken te controleren. Het protocol maakt dus gebruik van een cryptografische signature (meestal HMAC-SHA1) dat het token en andere gegevens die nodig zijn voor een request combineert. Dit verreist een extra stap namelijk dat de server om een request token vraagt. Het grote voordeel van OAuth1.0 is dat de token nooit rechtstreeks wordt doorgestuurd, waardoor de mogelijkheid dat iemand het doorgestuurde passwoord of token ziet volledig wordt geëlimineerd. Bij dit protocol is het gebruik van TLS, in tegenstelling tot OAth2.0 en basic authentication, niet nodig. Als het zeer gevoelige data is wordt het gebruik van TLS toch aangeraden. Dit niveau van beveiliging komt met een prijs namelijk dat het genereren en valideren van de signature een complex proces kan zijn. Onderstaande afbeelding toont de werking van OAuth1.0, waar een request token wordt gegenereerd, hoe deze wordt verwerkt, en zo een access_token wordt vrijgegeven (Stormpath, 2013) (Mayko, 2015).



Figuur 3: werken Oauth1.0 (Mayko, 2015)

2.5.2. OAuth2.0:

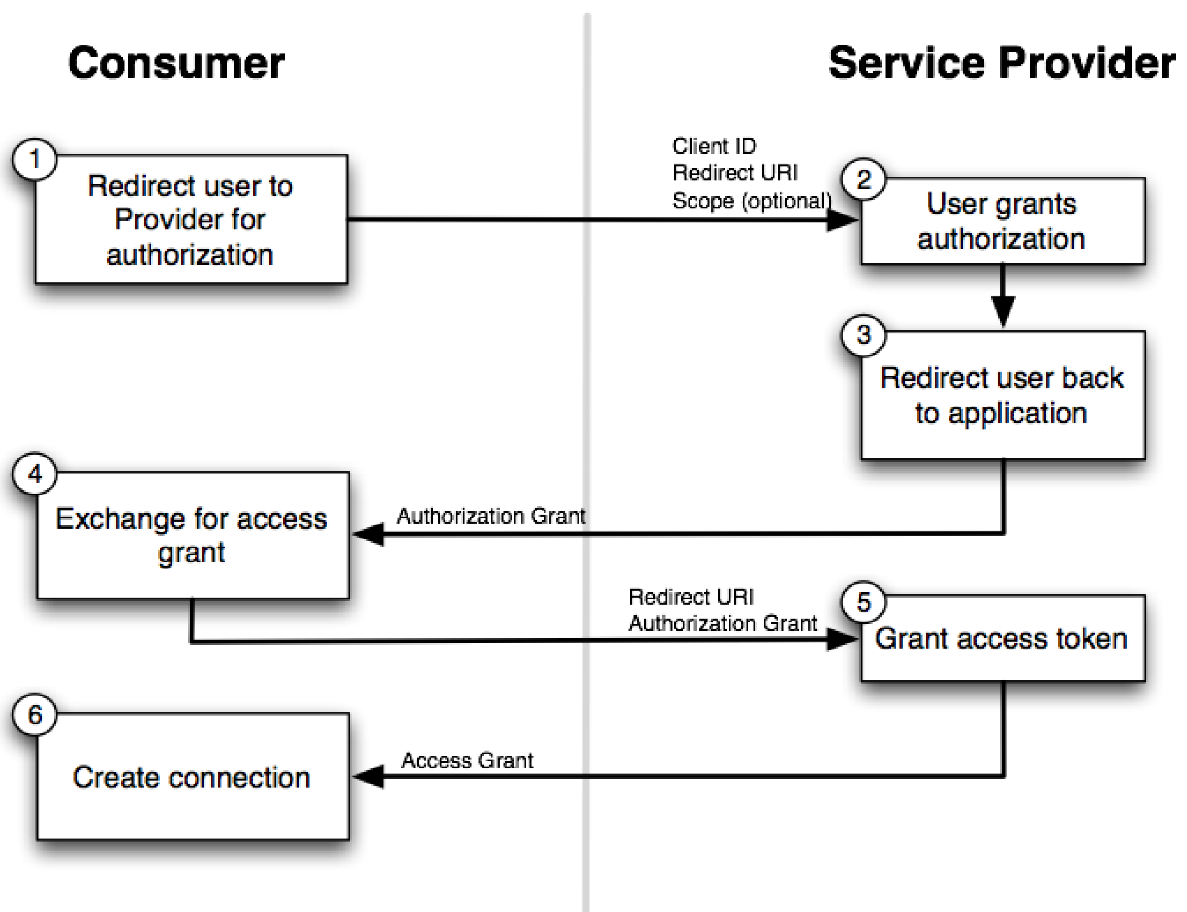
OAuth2 klinkt als een evolutie van OAuth1, maar in werkelijkheid is het een heel andere kijk op verificatie waarbij een poging wordt gedaan om de complexiteit te verminderen.

Alle encryptie wordt nu afgehandeld door TLS, betekenis TLS 2.1, waardoor het gebruik van signatures wordt verwijderd. Hierdoor zijn de cryptografische algoritmes voor het maken, het genereren en het valideren van een signature niet langer nodig (Stormpath, 2013).

OAuth2.0 brengt 5 nieuwe flows welke de gebruiker flexibiliteit biedt. Afhankelijk welk type systeem het is, kan een bepaalde flow worden gebruikt (Mangal, 2013):

- User-Agent Flow: wordt gebruikt bij native applicaties voor zowel mobiele als desktop systemen. Er wordt gebruikt gemaakt van het Implicit Grant-type verder uitgelegd bij **Fout! Verwijzingsbron niet gevonden.**;
- Web Server Flow: typisch voor systemen die worden gebruikt welke toegankelijk zijn via een webbrowser. Werkt volgens het Authorization Code grant **Fout! Verwijzingsbron niet gevonden.**;
- Username and Password Flow: er moet een vertrouwelijke band zijn tussen de user en de client. Dit is nodig omdat de credentials van de user worden doorgestuurd;
- Assertion Flow: door een assertion voor te leggen kan de client een access_token verkrijgen. Een voorbeeld hiervan is Security Assertion Markup Language (SAML).
- Client Credentials Flow: de client credentials worden doorgestuurd voor een access_token te verkrijgen.

Figuur 4 toont aan dat de werking van OAuth2.0 minder stappen verreist dan deze van OAuth1.0.



Figuur 4: werking OAuth2.0 (Mayko, 2015)

2.5.3. OAuth1.0 vs. OAuth2.0:

Ondanks dat de naam doet vermoeden dat OAuth2.0 verder gebouwd is op OAuth1.0 zijn er grote verschillen tussen beide. Er is reeds veel gezegd en geschreven dat OAuth1.0 veiliger is dan OAuth2.0. Toch gebruiken veel grote bedrijven OAuth2.0 (vooral sociale netwerken zoals facebook). OAuth2.0 biedt de functie dat gebruikers kunnen beslissen tot welke acties de aanroepende applicatie toegang heeft, wat dient als een limiet aan wat derden kunnen doen met informatie. Nog een belangrijk punt om te vermelden is dat de tweede versie van OAuth2.0 flexibeler is.

Redenen waarom het OAuth2.0 protocol is ontworpen (nadelen OAuth1.0) (Mangal, 2013):

- Gebruik van een signature is bij OAuth 2.0 verwijderd wat het protocol veel minder complex maakt. Bij OAuth2.0 is een verbinding met HTTPS vereist.
- OAuth bestaat uit 2 belangrijke onderdelen, een access_token verkrijgen en deze gebruiken bij een request. De extra flows die OAuth 2.0 biedt zoals hierboven vermeld zijn geschikt voor mobiele applicaties voor een betere voorziening van deze 2 onderdelen.
- OAuth 2.0 biedt de broodnodige scheiding tussen authenticatie en het verwerken van API requests.

Nadelen OAuth2.0 (Mangal, 2013):

- Tokens vervallen: dit nadeel is tegelijk ook een voordeel. Het nadeel is dat wanneer de access_token is vervallen de gebruiker opnieuw moet inloggen. Dit komt de gebruiksvriendelijkheid niet ten goede. Het voordeel hiervan is dat vorige tokens ongeldig worden waardoor hackers minder tijd hebben voor een geldig token te achterhalen;
- Extra beveiliging nodig: bij elke grote implementatie is het een vereiste om beveiligde eindpunten en een beveiligde URL te hebben, anders zal het veel te gemakkelijk zijn voor een hacker om de tokens te ontcijferen.

Voordelen Oauth2.0 (Mangal, 2013):

- Bij het gebruik van OAuth2 is het verplicht om de gegevens te versturen door middel van een beveiligde TLS connectie;
- Door gebruik te maken van een refresh_token is het probleem met het vervallen van de tokens opgelost;
- Het implementeren van OAuth2.0 is veel minder complex dan OAuth1.0. Als grootste reden is hiervoor dat er geen signature nodig is;
- API keys worden niet in een onveilige omgeving opgeslagen maar er worden short lived tokens opgeslagen op een 'onbetrouwbare' omgeving (smartphone van de gebruiker) welke na verloop van tijd vervallen;
- Het password grant flow kan worden gebruiken voor het inloggen door middel van een acces_token. Dit token wordt lokaal opgeslagen op de smartphone;
- Er kan gebruik gemaakt worden van een JWT, JSON Web Token. Het acces token wordt hierin opgeslagen. Buiten dit token kunnen er ook meerdere gegevens worden opgeslagen zoals de auteur en een vervaltijd voor het token.

De keuze is gemaakt om OAuth2.0 te implementeren voor dit systeem. Dit wordt verder besproken bij 3.6.

3. Ontwikkelingsfase

3.1. PHP framework: Laravel

Voor een PHP-programmeur zijn heel wat verschillende frameworks beschikbaar. Een framework maakt het voor developers makkelijker om applicaties te ontwikkelen. Allemaal hebben ze hun specifieke eigenschappen met bijbehorende voor- en nadelen. In dit project is er gekozen om te werken met Laravel.

Laravel is een web applicatie framework en probeert de ontwikkeling van projecten vlotter te laten verlopen door veelvoorkomende functies te versoepelen. Dit zonder in te boeten op de functionaliteit van het project. Door deze aanpak is het mogelijk om geavanceerde projecten te ontwikkelen met minder code dan voorheen (Introduction, sd).

3.1.1. Composer

Composer is een tool voor dependency management in PHP. Door middel van composer is het mogelijk om bepaalde bibliotheken voor het project te installeren en up-to-date te houden. Composer beheert de bibliotheken per project door ze te installeren binnen een bepaalde map in het project (Nils Adermann, sd).

3.1.2. Artisan

Artisan is de command-line interface horende bij Laravel. Er worden nuttige commando's voorzien die kunnen gebruikt worden in de ontwikkelingsfase. Zo zijn er commando's voorzien voor bijvoorbeeld het aanmaken van migrations 3.2.1, of een controller. Het is ook mogelijk om zelf commando's te schrijven (Artisan Console, sd).

3.1.3. MVC: Model View Controller

De standaard structuur binnen een laravel project heeft een directory genaamd app/ met 3 subdirectory's namelijk: models/, views/ en controllers. Dit verwijst naar de MVC structuur.

- Model: modellen zijn gebaseerd op real-world items zoals een persoon, een vraag of een antwoord. Modellen zijn permanent en worden buiten het programma opgeslagen, zoals in een database. In een model worden al de gegevens van het object opgeslagen, een voorbeeld hiervan is een Question. Een Question bevat 2 paramaters, namelijk een question_type_id en text.

```php
namespace Dharma\Models;

use Illuminate\Database\Eloquent\Model;

class Question extends Model {

        protected $table = 'question';
        protected $fillable = ['question_type_id', 'text'];
        public $timestamps = false;
```

Figuur 5: Laravel model deel 1

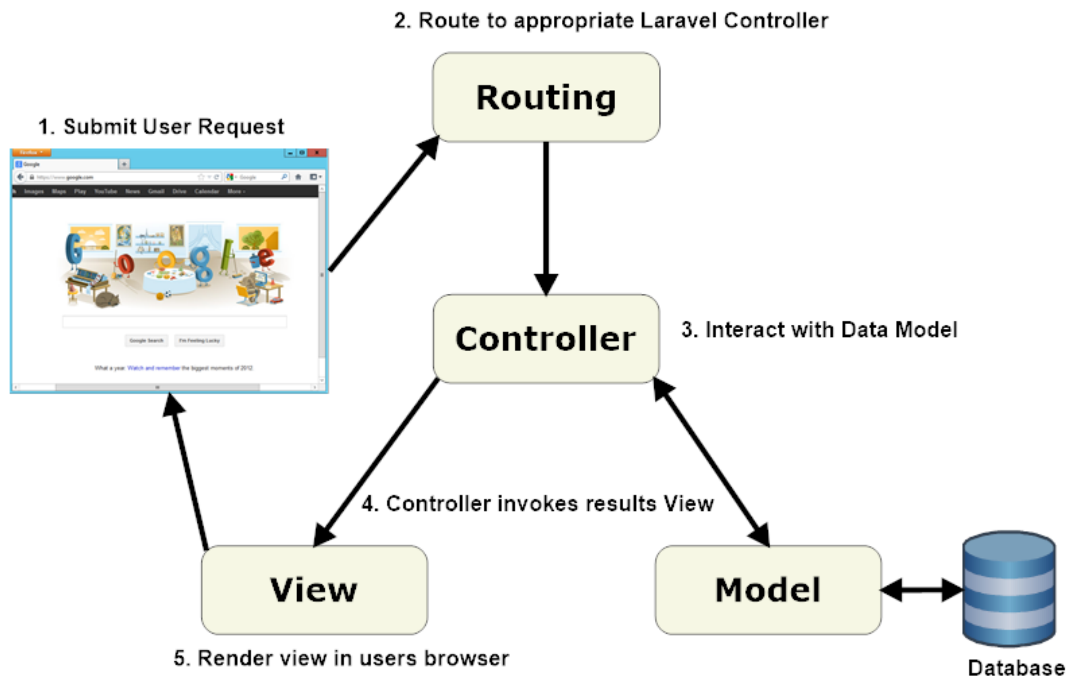
```php
public function answers() {
        return $this->belongsToMany('Dharma\Models\Answer',
                'question_answer', 'question_id', 'answer_id');
}

public function questionLists() {
        return $this->belongsToMany('Dharma\Models\QuestionList',
                'question_list_question', 'question_id',
                'question_list_id');
}

public function questionType(){
        return $this->belongsTo('Dharma\Models\QuestionType');
}

public function questionListQuestions(){
        return $this->hasMany('Dharma\Models\QuestionListQuestion');
}
public function questionGroups() {
        return $this->belongsToMany('Dharma\Models\QuestionGroup',
                'question_group_question', 'question_id',
                'question_group_id');
}
```

Figuur 6: Laravel model deel 2

De twee voorgaande afbeeldingen tonen het model van Question. Het eerste deel toont de variabelen welke in het object zitten. Door middel van Eloquent ORM binnen een model worden relaties tussen tabellen in de database gedefinieerd binnen de code. Hierdoor zijn gegevens uit gerelateerde tabellen makkelijk toegankelijk. Figuur 6 toont het gebruik van Eloquent ORM. Zo is bijvoorbeeld de tabel Question verbonden met de tabel Answer. ORM, wat staat voor object-relational mapper, is een programmeertechniek om data om te zetten in objectgeoriënteerde programmeertalen. Dit creëert een virtuele database die binnen de programmeertaal kan worden gebruikt (Architecture of Laravel Applications, sd);

- View: Een view geeft de visuele representatie van een model. Het is verantwoordelijk voor het genereren van een user interface welke dan is gebaseerd op data uit het model (Architecture of Laravel Applications, sd);
- Controller: Een controller zorgt voor de koppeling tussen een model met het daarbij horende view. De controller is verantwoordelijk voor het verwerken van de input en neemt de beslissing welke actie wordt uitgevoerd (Architecture of Laravel Applications, sd).

Figuur 7 toont de structuur van een MVC opgebouwd project binnen Laravel.



Figuur 7: MVC met routing (Architecture of Laravel Applications, sd)

Wanneer er een request wordt verstuurd via een web browser of een mobiele applicatie wordt deze verwerkt binnen routing. Gebaseerd op URL zal bepaald worden welke controller wordt aangesproken om zo de verwerking uit te voeren.

3.1.4. Routing

Alle Laravel routes zijn gedefinieerd in app/Http/routes.php, dat automatisch wordt geladen door het framework. In deze file wordt de verbinding tussen een request en een controller vastgelegd. Routes kunnen worden opgedeeld in verschillende groepen, waarbij iedere groep eigen voorwaarden kan hebben. Door gebruik te maken van een prefix op een routegroep is het mogelijk om reeds te beginnen met een standaard URL. Zo is het mogelijk om een goed overzicht te behouden welke routes bij welk deel van het project horen (HTTP Routing, sd).

3.2. Database

De eerste stap in deze masterthesis was het ontwerpen van een goed databasemodel. Bij het ontwerpen van een databasemodel is het belangrijk om van in het begin op een correcte en overzichtelijke manier te werk te gaan. Dit databasemodel wordt als uitbreiding toegevoegd aan een reeds bestaand ontwerp. Hierdoor is het belangrijk dat de juiste naamgeving wordt gebruikt zodat het toevoegen en uitbreiden van dit schema niet voor verwarring zorgt.

3.2.1. Migrations

Voor het uitbreiden van de database wordt gebruik gemaakt van migrations. Door middel van deze migrations is het mogelijk om tabellen toe te voegen aan een reeds bestaande database. Via het Atisan commando "make:migration" wordt de migration onmiddellijk toegevoegd in de directory database/migrations. Samen met het commando wordt de file name van de migration meegegeven zoals te zien is in onderstaande afbeelding (Database: Migrations, sd).

```
php artisan make:migration create_user_phone_type_table
```

Figuur 8: migratie database toevoegen

Elke file name wordt voorzien van een timestamp. Aan de hand van deze file name wordt nagegaan welke migration wanneer is toegevoegd en welke migration reeds is uitgevoerd. Volgende afbeelding toont een reeks migrations die zijn aangemaakt. Er is te zien dat de file name wordt voorafgegaan met de timestamp en ook wordt geordend volgens deze timestamp.

2016_06_02_111427_create_user_phone_table.php	20/07/2016 10:19	PHP-bestand	1 kB
2016_06_02_125025_create_foreign_key_user_phone.php	20/07/2016 10:19	PHP-bestand	1 kB
2016_06_03_103723_delete_phone_type_from_user.php	20/07/2016 10:19	PHP-bestand	1 kB
2016_06_03_112725_create_user_phone_type_table.php	20/07/2016 10:19	PHP-bestand	1 kB
2016_06_03_113607_add_user_phone_type_id_to_user_phone.php	20/07/2016 10:19	PHP-bestand	1 kB
2016_06_03_113831_create_foreign_key_user_phone_type_id.php	20/07/2016 10:19	PHP-bestand	1 kB

Figuur 9: Lijst van migraties

Op de afbeelding is ook te zien dat de migration een PHP-bestand is. Elke migration is een extensie van de klasse Migration en wordt voorzien van 2 methodes: up en down. De up methode wordt gebruikt om een nieuwe kolom met bijbehorende tabellen toe te voegen aan de database. Via de down methode wordt de bewerking die wordt gedaan in de up methode ongedaan gemaakt.

```php
/**
 * Run the migrations.
 *
 * @return void
 */
public function up()
{
    Schema::create('user_phone_type', function(Blueprint $table) {
        $table->increments('id');
        $table->string('phone_type', 255);
    });
}
```

Figuur 10: Tabel aanmaken via migratie

Figuur 10 toont een voorbeeld van de up methode. Hier wordt een tabel, user_phone_type aangemaakt samen met de kolommen id en phone_type.

```
/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::drop('user_phone_type');
}
```

Figuur 11: Migratie down functie

Door middel van de drop functie wordt de down methode voorzien om de up methode van het aanmaken van de tabel user_phone_type ongedaan te maken.

Voor de aangemaakte migrations uit te voeren wordt het Artisan commando “php artisan migrate” uitgevoerd. Om de down methode te gebruiken is het Artisan commando “php artisan migrate:rollback” voorzien.

3.2.2. Foreign Keys

Bij het ontwerpen van het databasemodel voor een dynamische vragenlijst is het gebruik van foreign keys heel belangrijk. Door middel van een foreign key wordt een kolom in een tabel gekoppeld aan een primary key kolom in een andere tabel. De tabel waarin de foreign key wordt aangemaakt wordt de child tabel genoemd. De tabel naar waar de foreign key verwijst wordt de parent tabel genoemd. Een foreign key kan via een migration worden toegevoegd aan bestaande kolommen binnen een reeds aangemaakt tabel (Constraints, sd).

```
public function up()
{
    Schema::table('question_answer', function(Blueprint $table) {
        $table->foreign('question_id')->references('id')->on('question')
            ->onDelete('restrict')
            ->onUpdate('restrict');
    });
    Schema::table('question_answer', function(Blueprint $table) {
        $table->foreign('answer_id')->references('id')->on('answer')
            ->onDelete('restrict')
            ->onUpdate('restrict');
    });
```

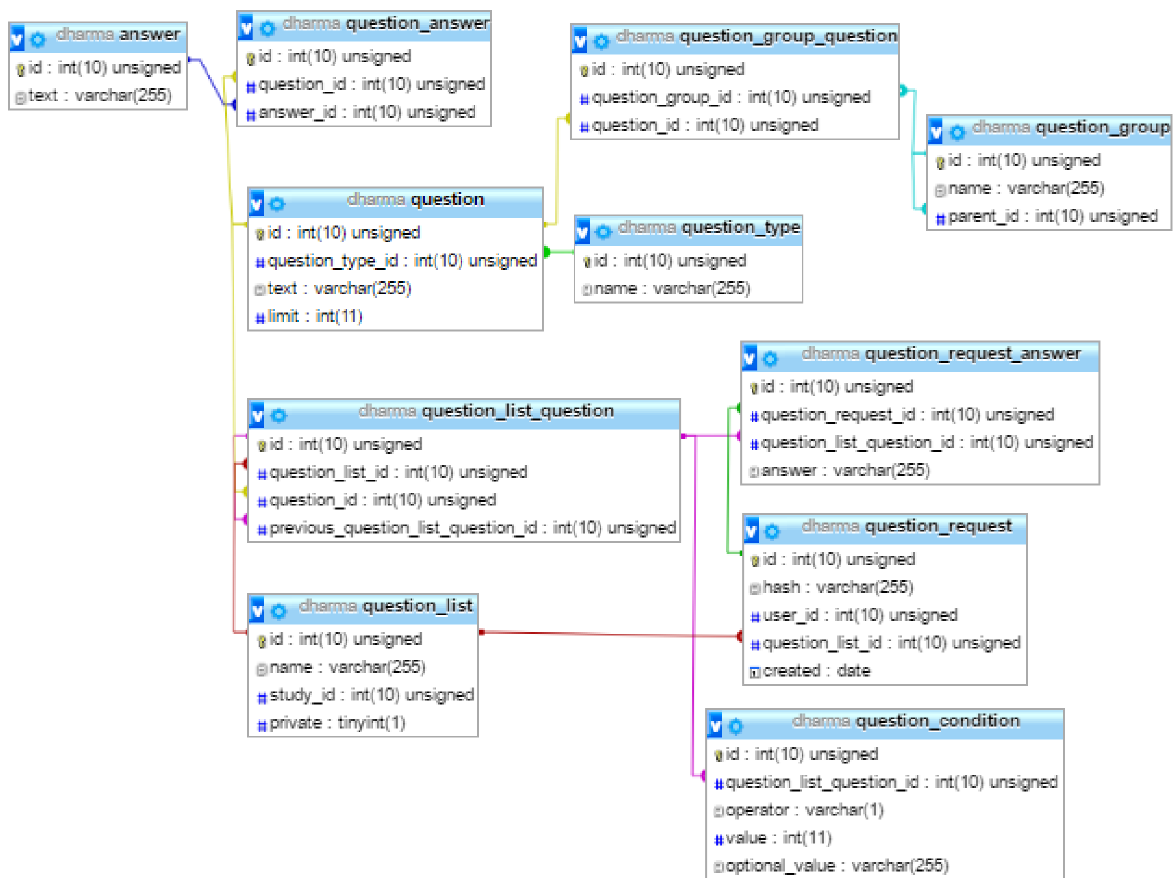
Figuur 12: Foreign keys toevoegen

```php
public function down()
{
    Schema::table('question_answer', function(Blueprint $table) {
        $table->dropForeign('question_answer_question_id_foreign');
    });
    Schema::table('question_answer', function(Blueprint $table) {
        $table->dropForeign('question_answer_answer_id_foreign');
    });
```

Figuur 13: Down functie voor foreign keys

De migration wordt zoals hierboven beschreven gegenereerd door het artisan commando "make:migration" waarin een up en down methode is voorzien. In de up methode wordt de foreign key aangemaakt en in de down methode wordt het aanmaken van deze foreign key verwijderd. Figuren 12 en 13 tonen het aanmaken en verwijderen van foreign keys die gebruikt worden bij de tabel question_answer. In dit voorbeeld zijn er 3 tabellen namelijk: question_answer, question en answer. Aan de hand van de naamgeving is reeds te zien dat de tabel question_answer een koppeling is tussen de tabel question en de tabel answer. Zowel de tabel question als de tabel answer wordt voorzien van primary key tabel id. Beide id's worden in de tabel question answer gebruikt waarvoor 2 foreign key's worden aangemaakt. De kolom question_id binnen de tabel question_answer wordt via een foreign key gekoppeld aan de kolom id binnen de tabel question. Hetzelfde wordt gedaan voor de kolom answer_id binnen de tabel question_answer en de kolom id binnen de tabel answer. Aan de hand van dit voorbeeld is ook te zien dat de naamgeving van elke tabel en elke kolom heel belangrijk is zodat alle kolommen en tabellen op een duidelijke manier kunnen worden gekoppeld.

3.2.3. Databasemodel dynamische vragenlijst



Figuur 14: Database model vragenlijst

Bovenstaand schema toont de tabellen die worden gebruikt voor het aanmaken van een dynamische vragenlijst. De tabel question_list is een dynamische vragenlijst. Iedere vragenlijst wordt opgesteld uit verschillende vragen. In de tabel question_list_question worden door foreign key's de vragen gekoppeld aan een vragenlijst. Wanneer een vraag gebruik maakt van een operator (=, <, >) wordt deze gekoppeld aan een condition. Elke vraag is van een bepaald type: multiple choice, yes or no, open of scale. Wanneer de vraag van het type multiple choice is worden de mogelijke antwoorden vanuit de tabel answer gekoppeld via de tabel question_answer aan de vraag. Wanneer een vragenlijst wordt samengesteld kan deze in de tabel question_request worden gekoppeld aan een patiënt. Door middel van de hash die bij de request hoort kan de patiënt de vragenlijst ophalen via een smartphone. Wanneer de patiënt de vragenlijst heeft ingevuld en de antwoorden doorstuurt worden deze opgeslagen in de database. In de tabel question_request_answer worden de antwoorden gekoppeld met de vraag aan de hand van de tabel question_list_question.

3.3. API voor DHARMA

3.3.1. RESTful resource controllers

Een RESTful API is een API die http-requests gebruikt voor handelingen uit te voeren. REST benadrukt dat de interactie tussen gebruikers en servers wordt beperkt tot een aantal acties. Deze acties worden verbs genoemd. Elke verb heeft een specifieke betekenis waarbij een specifieke handeling hoort. De mogelijkheden worden getoond in tabel 1 aan de hand van een voorbeeld namelijk user (RESTful Resource Controllers, sd).

Verb	Path	Action	Route Name
GET	/user	index	user.index
GET	/user/create	create	user.create
POST	/user	store	user.store
GET	/user/{user}	show	user.show
GET	/user/{user}/edit	edit	user.edit
PUT/PATCH	/user/{user}	update	user.update
DELETE	/user/{user}	destroy	user.destroy

Tabel 1: RESTful Resource Controllers

- GET: het ophalen van een bron, hierbij zijn meerdere acties mogelijk
  - Index: toont een lijst met alle users;
  - Create: maakt een nieuwe user aan;
  - Show: toont users aan de hand van een variabele, dit kan bijvoorbeeld het id zijn van de user;
  - Edit: mogelijkheid tot het bewerken van een user.
- POST: slaat 1 of meerdere users op in de database.
- PUT/PATCH: update een specifieke user.
- DELETE: verwijdert een specifieke user.

3.3.2. Routes

Voor de connectie tussen de server en de smartphone worden er verschillende routes aangemaakt. Zoals te zien in onderstaande afbeelding zijn er 6 routes die worden gebruikt met elk een eigen functie. Elke route bestaat uit verschillende delen waaruit wordt afgeleid welke functie de route heeft. De eerste kolom toont aan of de route een POST of een GET is. Bij get worden gegevens opgehaald van de server, bij POST worden gegevens gestuurd naar de server. De middelste kolom toont aan welke URL wordt aangesproken voor de bepaalde route te gebruiken. De derde kolom toont aan welke methode wordt gebruikt. Door middel van show worden gegevens getoond aan de hand van een bepaalde variabelen. Wanneer er gebruik wordt gemaakt van store worden er gegevens doorgestuurd naar de database waar deze gegevens worden opgeslagen. Bij index worden er gegevens getoond zonder een variabele mee te geven.

```
POST       | api/v2/measurement                                    | api.v2.measurement.store
POST       | api/v2/oauth/access_token                             |
POST       | api/v2/question                                       | api.v2.question.store
GET|HEAD   | api/v2/question/{question}                            | api.v2.question.show
GET|HEAD   | api/v2/request                                        | api.v2.request.index
GET|HEAD   | api/v2/request/{request}                              | api.v2.request.show
```

Figuur 15: API routes

Volgende afbeelding toont hoe de routes worden aangemaakt. De routes zijn opgebouwd in 2 groepen. De eerste groep zet een prefix voor al de gebruikte routes binnen de groep, hierdoor begint elke URL met dezelfde gegevens namelijk /api/v2. Door gebruik te maken van een middleware worden vereisten voor de route aan te spreken vastgelegd. De middleware 'json_only' zorgt ervoor dat een request enkel JSON data mag bevatten. In de eerste groep wordt de route voor het bekomen van een access_token aangemaakt. Wanneer een access_token is verkregen kan de volgende groep worden aangesproken. Dit doordat er de middleware 'oauth' wordt gebruikt. Hier is een vereiste voor de communicatie dat er een geldige access_token wordt meegestuurd met de request.

```
Route::group(['prefix' => '/api/v2', 'middleware' => ['json_only']], function()
{
    Route::post('oauth/access_token', 'Api\AuthenticateController@oauth');
    Route::group(['middleware' => ['oauth']], function()
        {
            Route::resource('measurement', 'Api\MeasurementController' ,
                    ['only' =>['store']]);
            Route::resource('question', 'Api\QuestionController' ,
                    ['only' =>['store', 'show']]);
            Route::resource('request', 'Api\RequestController' ,
                    ['only' =>['show', 'index']]);
        });
});
```

Figuur 16: API routes group

Verder wordt er in de route file een koppeling gemaakt tussen de route en de bijbehorende controller. Wanneer een bepaalde route correct wordt aangesproken wordt er doorverwezen naar de controller waarna de bewerking wordt uitgevoerd. In dit project wordt voor de API gebruik gemaakt van 4 controllers welke worden getoond in tabel 2.

Question	Measurement	Request	Authenticate
store, show	store	show	oauth

Tabel 2: Resource Controllers

Een controller voor elke bewerking met een vragenlijst af te werken, QuestionController. Deze maakt gebruik van store en show. Vervolgens is er een controller voor de data van een meting door te sturen namelijk de MeasurementController waar een store wordt gebruikt. De RequestController gaat na of een bepaalde vragenlijst reeds is beantwoord doormiddel van een show. Ten slotte, de AuthenticatieController, controleert of een patiënt met correcte gegevens inlogt en voorziet deze van een access_token wanneer dit succesvol is. Hiervoor is een aparte functie namelijk oauth.

3.3.3. QuestionController: show

De QuestionController is opgedeeld in 2 delen, het ophalen van vragen en het opslaan van antwoorden. Hiervoor worden 2 methodes gebruikt: store en show.

Er wordt een request verstuurd wanneer een patiënt een vragenlijst moet invullen met als parameter een *hash*. Deze *hash* is in de database gekoppeld aan een specifieke vragenlijst. Wanneer de request is verstuurd, wordt de juiste controller aangeroepen. Dit is de QuestionController. Als variabele voor de methode 'show' wordt een *hash* meegestuurd. Aan de hand van de *hash* wordt via het model van de QuestionRequest het bijbehorende request uit de database gehaald. Dit door middel van de *hash* van de request te vergelijken met de *hash* van de requests opgeslagen in de database. Wanneer de juiste request is gevonden wordt deze opgeslagen in de variabele questionRequest. De variabele questionRequest bevat nu al de informatie die nodig is voor het ophalen van de vragen met bijbehorende gegevens. Allereerst wordt de user_id gecontroleerd of deze recht heeft om de des betreffende vragenlijst op te halen. Wanneer de patiënt geen recht heeft tot de vragenlijst wordt een error teruggestuurd.

```
public function show($hash)
{
  $questionRequest = QuestionRequest::where('hash', $hash)->get()->first();
  if($questionRequest->user_id == Authorizer::getResourceOwnerId())
  {
      $fromAlert = $questionRequest->from_alert;
      $questionsListQuestionId = QuestionListQuestion::where
              ('question_list_id', $questionRequest->question_list_id)->
              get()->pluck('id');
      $list = QuestionList::where('id', $questionRequest->question_list_id)->get();
      $conditions = QuestionCondition::whereIn('question_list_question_id',
              $questionsListQuestionId)->get();
      $questionListQuestion = QuestionListQuestion::where('question_list_id',
      $questionRequest->question_list_id)->with('question.answers')->get();

      $response = [
          'from_alert'=>$fromAlert,
          'list'=>$list,
          'conditions'=>$conditions,
          'questions'=>$questionListQuestion
      ];
      return $response;
  }
  else{
      return response()->json(["error"=>"User is not permitted for this action."], 401);
  }
}
```

Figuur 17: RequestController show

Bij een correcte authenticatie wordt de bewerking voor het ophalen van de vragenlijst verder gezet. Het antwoord op de request bestaat uit 4 delen. Ten eerste wordt gecontroleerd of de vragenlijst afkomstig is van een alert. Dit wil zeggen dat de metingen van de patiënt niet goed waren en er een vragenlijst is opgesteld om na te gaan wat hiervan de oorzaak kan zijn. Vervolgens wordt gecontroleerd van welke lijst de vragen afkomstig zijn. Hierna wordt er nagegaan of er vragen in de lijst zitten welke een bepaalde conditie hebben. Ten slotte worden de vragen opgehaald die bij deze lijst horen. Als er multiple choice vragen in de lijst zitten worden de bijbehorende antwoorden ook opgehaald.

Wanneer al deze gegevens zijn opgehaald, wordt de response op de request opgesteld. Hierin worden al de 4 delen opgenomen. Onderstaande afbeelding toont een voorbeeld van zo een response. Deze gegevens worden verwerkt door de applicatie op de smartphone waarna de gebruiker de vragen te zien krijgt en deze kan beantwoorden.


```
{
  "list": [
    {
      "id": 1,
      "name": "Prentale raadpleging (tussentijds)'
    }
  ],
  "conditions": [
    {
      "id": 1,
      "question_list_question_id": 3,
      "operator": "=",
      "value": 1,
      "optional_value": 0
    }
  ],
  "questions": [
    {
      "id": 1,
      "question_list_id": 1,
      "previous_question_list_question_id": null,
      "question": {
        "id": 1,
        "question_type_id": 2,
        "text": "Voelt u zich goed?",
        "limit": 0,
        "answers": [
          {
            "id": 1,
            "text": "Ja"
          },
          {
            "id": 2,
            "text": "Nee"
          }
        ]
      }
    },
```


Figuur 18: voorbeeld lijst van vragen

3.3.4. QuestionController: store

Wanneer de patiënt klaar is met het invullen van de vragen en deze terug verstuurd naar de server worden de ontvangen gegevens opgebouwd zoals in figuur 19.

```
{
    "hash": "eEGCRCwmKZ4fgMoFsc8g1iu2h1TEwvTpri5Rz70VtAdQR44KXQ3riG7w6h6i",
    "answers":
    [
        {
            "Id": 1,
            "QuestionListQuestionId": 2,
            "Answer": "2"
        },
        {
            "Id": 2,
            "QuestionListQuestionId": 6,
            "Answer": "3"
        }
    ]
}
```

Figuur 19: voorbeeld van antwoorden verstuurd naar server

Dit bevat de hash van de bijbehorende request die eerder was verstuurd naar de patiënt en het antwoord dat bij een bepaalde vraag behoord. Aan de hand van deze gegevens wordt in de QuestionController een bewerking uitgevoerd waarmee de antwoorden worden opgeslagen in de database. Net zoals bij het ophalen van de vragen wordt aan de hand van de hash gecontroleerd over welke QeustionReqeust het gaat. Wanneer de juiste request is opgehaald, wordt er eerst gecontroleerd aan de hand van de user_id of de patiënt recht heeft om de antwoorden te versturen naar de database. Als deze controle in orde is, wordt allereerst een controle uitgevoerd of er reeds antwoorden van deze request in de database zijn opgeslagen. Hierna wordt elk antwoord verwerkt. Voor elk antwoord wordt er aan de hand van het QuestionRequestAnswer model een questionRequestAnswer aangemaakt. Hierin wordt het answer, de question_request_id en de question_list_question_id toegevoegd. Vervolgens wordt het questionRequestAnswer opgeslagen in de database.

```php
class QuestionController extends Controller {

  public function store(Request $request)
  {
    $hash = $request['hash'];
    $questionRequestUserId = QuestionRequest::where('hash', $hash)->first()->user_id;

    if($questionRequestUserId == Authorizer::getResourceOwnerId())
    {
        QuestionRequestAnswer::where('question_request_id', $questionRequestUserId)->delete();
        foreach($request->input('answers') as $answer){
            var_dump ($answer);
            $questionRequestId = QuestionRequest::where('hash', $hash)->get()->pluck('id')->first();
            $questionRequestAnswer = new QuestionRequestAnswer();
            $questionRequestAnswer->answer = $answer['Answer'];
            $questionRequestAnswer->question_request_id = $questionRequestId;
            $questionRequestAnswer->question_list_question_id = $answer['QuestionListQuestionId'];
            $questionRequestAnswer->save();
        }
    }
    else{
        return response()->json(["error"=>"User is not permitted for this action."], 401);
    }
  }
```

Figuur 20: QuestionController store

3.3.5. MeasurementController

Het moet voor de patiënt mogelijk zijn om zelf meetdata in te geven op de smartphone en deze door te sturen naar de server. Hiervoor is de MeasurementController voorzien. Een meting bestaat uit 2 delen: timestamp en data. Het eerste deel, de timestamp, is de dag en het uur waarop de meting is doorgestuurd. Het tweede deel, de data, bevat het type meting dat is uitgevoerd en de bekomen waarde die de patiënt bij deze meting verkreeg. Figuur 21 toont een voorbeeld van dergelijke meting. Deze data wordt met een post verzonden naar de URL voor de route van een measurement aan te roepen. Hierna wordt de data verwerkt in de controller.

```json
{
  "measurements": [
    {
      "timestamp": "2025-04-19T20:20:49+2:00",
      "data": [
        {
          "type": 10,
          "value": 95
        }
      ]
    },
    {
      "timestamp": "2025-04-20T20:22:57+04:00",
      "data": [
        {
          "type": 10,
          "value": 105
        }
      ]
    }
  ]
}
```

Figuur 21: voorbeeld van metingen verstuurd naar de server

In de controller wordt nagegaan welke patiënt de meting doorstuurde. Vervolgens wordt de timestamp van de meting omgezet waarna de meting wordt opgeslagen in een array. Wanneer elke

meting is verwerkt wordt er een reeds bestaande MeasurementHandler aangemaakt waar de *user* wordt meegegeven als variabele. Een handler wordt opgeroepen bij een bepaalde actie, hier een store naar de database met meetdata. De array van metingen meegegeven aan deze handler wordt verwerkt en opgeslagen in de database.

```
class MeasurementController extends Controller
{
    public function store(Request $request)
    {
        $user = User::where('id', Authorizer::getResourceOwnerId())->get()->first();
        $data = $request->input('measurements');
        $measurements = array();
        foreach($data as $measurement){
            $timestamp = strtotime($measurement['timestamp']);
            $measurement['timestamp'] = $timestamp;
            $measurements[] = $measurement;
        }
        $newMeasurement = new MeasurementHandler($user);
        $newMeasurement->handle($measurements);
    }
}
```

Figuur 22: MeasurementController store

3.3.6. RequestController

Door middel van de RequestController wordt er aan de hand van een *hash* gecontroleerd of een vragenlijst reeds is ingevuld of niet.

```
public function show($hash)
{
    $question_request_is_answered = QuestionRequest::where('hash', $hash)->get()->first()->is_answered;
    $response                     = [
        'is_answered' => $question_request_is_answered
    ];

    return $response;
}
```

Figuur 23: RequestController show

3.3.7. AuthenticatieController

Alvorens de patiënt bovenstaande bewerkingen kan uitvoeren moet er hij of zij een authenticatie doorvoeren. Door middel van een post verstuurt de patiënt gegevens opgebouwd zoals te zien is op figuur 24 naar de server.


```
{
  "grant_type": "password",
  "client_id": "f3d259ddd3ed8ff3843839b",
  "client_secret": "4c7f6f8fa93d59c45502c0ae8c4a95b",
  "password": "patient",
  "username": "anouk15@gmail.com",
  "phone_token": "eh2KMeYNp5k:APA91bEg-ivQj5PzV2WUNI0wQxFHVSUqpisuBXkXbNbzuHpe2b25c99QE-8Zkez6WSU4QFD5_2nJkvId4zxktZnS7IkjmdZnugPmHb1HBS
      -v5Iy-dgyoooy1HlHEhdYJjjL34BMEHNVq",
  "phone_type": 1
}
```


Figuur 24: gegevens voor aanvraag access_token

Aan de hand van grant-type password, zie **Fout! Verwijzingsbron niet gevonden.**, weet de controller dat er controle wordt doorgevoerd aan de hand van het passwoord dat de patiënt meestuurt. De werking van deze beveiliging samen met client_id en client_secret wordt bij 3.6 besproken.

Wanneer de patiënt inlogt, worden phone_token en phone_type meegestuurd naar de controller. Als de controle van password en username correct is wordt userPhone aangemaakt naar het model UserPhone. Hierin wordt de user_id, phone_token en user_phone_type_id geplaatst. Indien er nog geen smartphone gekoppeld is aan de patiënt worden deze gegevens opgeslagen in de database. Als de patiënt reeds had ingelogd vanaf een smartphone worden de gegevens in de database geupdate. Deze gegevens worden gebruikt bij het versturen van een vragenlijst naar de smartphone van de patiënt via een pushnotificatie 3.4. Volgende code toont de werking van de AuthenticatieController.

```
class AuthenticateController extends Controller
{
    public function oauth(Request $request)
    {
        $pgf = new PasswordGrantVerifier();
        $pgf->verify($request['username'], $request['password']);
        if(Auth::user())
        {
            $userId = Auth::user()->id;
            $userPhone = UserPhone::where('user_id', $userId)->first();
            if($userPhone == null)
            {
                $userPhone = new UserPhone();
                $userPhone->user_id = $userId;
                $userPhone->phone_token = $request['phone_token'];
                $userPhone->user_phone_type_id = $request['phone_type'];
                $userPhone->save();
            }
            else
            {
                $userPhone->user_id = $userId;
                $userPhone->phone_token = $request['phone_token'];
                $userPhone->user_phone_type_id = $request['phone_type'];
                $userPhone->update();
            }
        }
        return Response::json(Authorizer::issueAccessToken());
    }
}
```

Figuur 25: AuthenticateController oauth

3.4. Pushnotifications

Wanneer een patiënt een vragenlijst moet invullen moet het voor een onderzoeker mogelijk zijn om een pushnotificatie te versturen naar het mobiele apparaat van de patiënt. Deze notificatie bevat de hash welke is gekoppeld aan een vragenlijst. Het is mogelijk om deze notificatie te versturen naar zowel een Android als een iOS apparaat. Hiervoor is gekozen om een open source library te gebruiken van David Bennun (Ryan, 2016). Deze library wordt via composer toegevoegd aan het Laravel project.

Als de onderzoeker een pushnotificatie wilt versturen naar de smartphone van een patiënt is er een vereiste dat de patiënt reeds met de smartphone is ingelogd. Wanneer een patiënt met een smartphone inlogt wordt in de database aan de patiënt een smartphone gekoppeld. Voor het versturen van de pushnotificatie wordt er een request aangemaakt waaruit de phone_token en het phone_type wordt gehaald. Hierna wordt het id van deze request doorgestuurd naar de NotificationHandler. In deze handler gebeurt de verwerking voor het versturen van de pushnotificatie naar de smartphone aan de hand van het meegestuurde id. Ten eerste wordt de QuestionRequest opgehaald door te zoeken op dit id. Deze request is gekoppeld aan een patiënt welke een user_id heeft. Vervolgens wordt aan de hand van het phone_type, Android of iOS, een pushnotificatie verstuurd naar de smartphone van de patiënt met als message de hash van de verstuurde request.

```
public function sendNotification($request_id)
{
    $request = QuestionRequest::find($request_id);
    $user_phone = UserPhone::where('user_id', $request->user_id)->first();
    $phone_token = $user_phone->phone_token;
    $user_phone_type = UserPhoneType::find($user_phone->user_phone_type_id);
    $phone_type = $user_phone_type->phone_type;
    $collection = PushNotification::app($phone_type)
            ->to($phone_token)
            ->send($request->hash);
}
```

Figuur 26: verzenden push notificatie

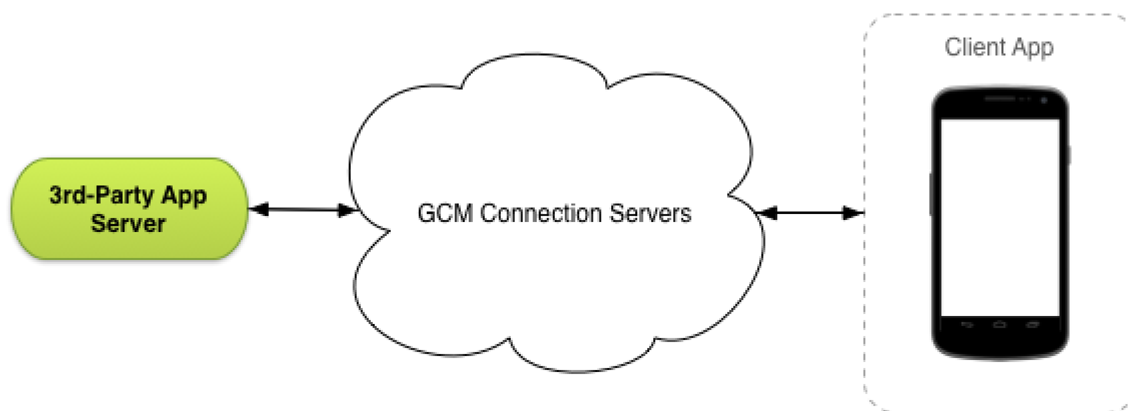
Dit deel is voor zowel Android als iOS gemeenschappelijk. Er wordt echter een onderscheidt gemaakt. Zoals te zien in volgende afbeelding, welke de config file is voor het versturen van de pushnotificaties, wordt er bij IOS gebruik gemaakt van apns 3.4.2, en bij Android van gcm 3.4.1.

```
return array(

    'IOS'     => array(
        'environment' =>'development',
        'certificate' =>storage_path('app/NotificationCertificate.pem'),
        'passPhrase'  =>'Medion1006',
        'service'     =>'apns'
    ),
    'Android' => array(
        'environment' =>'production',
        'apiKey'      =>'AIzaSyAZFhXA7Nq8UaWeMRTE_w9NScXx8LPEs6Q',
        'service'     =>'gcm'
    )

);
```

Figuur 27: config file push notificaties

3.4.1. Android: GCM

Door middel van Google Cloud Messaging is het mogelijk om een push notificatie naar een Android cliënt te versturen. Figuur 27 is te zien dat er een apiKey nodig is wanneer gebruikt wordt gemaakt van GCM. Developers kunnen hun applicatie registreren bij Google Cloud Messaging waarna deze de apiKey voor de applicatie ontvangen (Ryan, 2016).

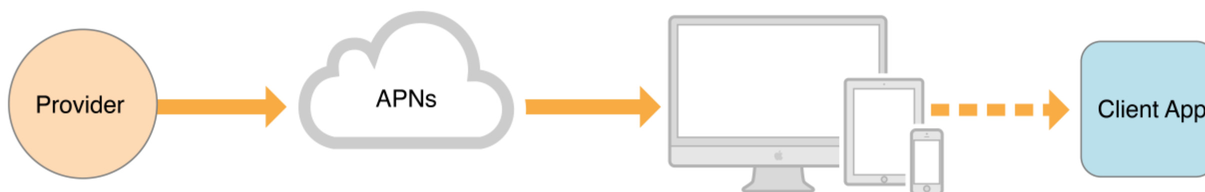


Figuur 28: werking GCM (Kiran V S, sd)

Figuur 28 toont de werking van gcm. Een 3rd-party (onderzoeker) stuurt een message naar de GCM Connection Servers. Deze message bevat de apiKey die bij de applicatie hoort, de phone_token en het bericht dat wordt getoond bij de pushnotificatie. Met deze gegevens weet de server naar welke smartphone de pushnotificatie moet worden verstuurd waarna de patiënt deze zal ontvangen als de applicatie pushnotificaties van gcm toestaat (Kiran V S, sd).

3.4.2. iOS: APNs

Versturen van een pushnotificatie naar een IOS toestel maakt gebruikt van de APNs server. Onderstaand diagram toont de werking van APNs. De onderzoeker stuurt een bericht samen met de phone_token naar de APNs servers. De routing van de APNs server zorgt ervoor dat de pushnotificatie naar de juiste smartphone wordt verstuurd, waarna de smartphone ervoor zorgt dat de pushnotificatie door de juiste applicatie wordt weergegeven (Apple Push Notification Service, sd).



Figuur 29: werking APNs (Apple Push Notification Service, sd)

3.5. Schedules

Verschillende patiënten moeten voor een nauwe opvolging na een bepaalde tijd een vragenlijst kunnen invullen. Ook is het mogelijk dat een bepaalde vragenlijst herhaaldelijk voorkomt in een studie en zo wordt verstuurd naar patiënten. Beide toepassingen kunnen dagelijks, wekelijks en maandelijks zijn. Hiervoor zijn extra tabellen toegevoegd aan de database voor een user te koppelen aan een question_schedule of door een study te koppelen aan de question_schedule.

Door middel van laravel is er een commando geschreven, CheckQuestionSchedule, waarmee wordt gecontroleerd of er een vragenlijst moet worden verzonden. In het commando wordt er voor elk schedule nagegaan of er die dag een dagelijkse, wekelijkse of maandelijkse vragenlijst moeten worden verstuurd.

Wanneer er een vragenlijst moet worden verstuurd, wordt er een QuestionRequest aangemaakt waarbij de vragenlijst wordt gekoppeld met een hash en deze wordt verzonden naar de patiënt.

3.6. Beveiliging van API: OAuth2.0

3.6.1. Keuze voor OAuth2.0

- ✓ Er worden verschillende grant-types ter beschikking gesteld die kunnen worden geïmplementeerd binnen het systeem. Een grant-type is een manier om de authenticatie van de patiënt te valideren. Naargelang welk type wordt gebruikt zijn er verschillende variabelen nodig om in te loggen;
- ✓ De patiënt kan inloggen met username en wachtwoord;
- ✓ Bij het inloggen wordt een extra controle uitgevoerd op client_id en client_secret wat voor extra beveiliging zorgt;
- ✓ Bij succesvolle authenticatie krijgt de patiënt een access_token waarmee het aanroepen van de API mogelijk is. De patiënt wordt aan de hand van dit token geauthentiseerd.
- ✓ Dit token vervalt na een vooringestelde tijd wat het systeem veiliger maakt. Er is minder tijd om een token de achterhalen en wanneer dit toch lukt, vervalt de token en moet er een nieuwe worden achterhaald;
- ✓ Door gebruik te maken van een refresh_token blijft het systeem gebruiksvriendelijk. De patiënt moet na het vervallen van een access_token niet opnieuw inloggen.
- ✓ De beveiliging kan worden uitgebreid door extra controle uit te voeren bij het aanroepen van de API. Bijvoorbeeld een controle op het client_id.

3.6.2. Library

Voor de implementatie van OAuth2.0 wordt er gebruik gemaakt van een library. De library gemaakt door Luca Degasperi (Degasperi, sd) biedt ondersteuning voor de implementaties die nodig zijn binnen dit systeem.

3.6.3. Grant-types

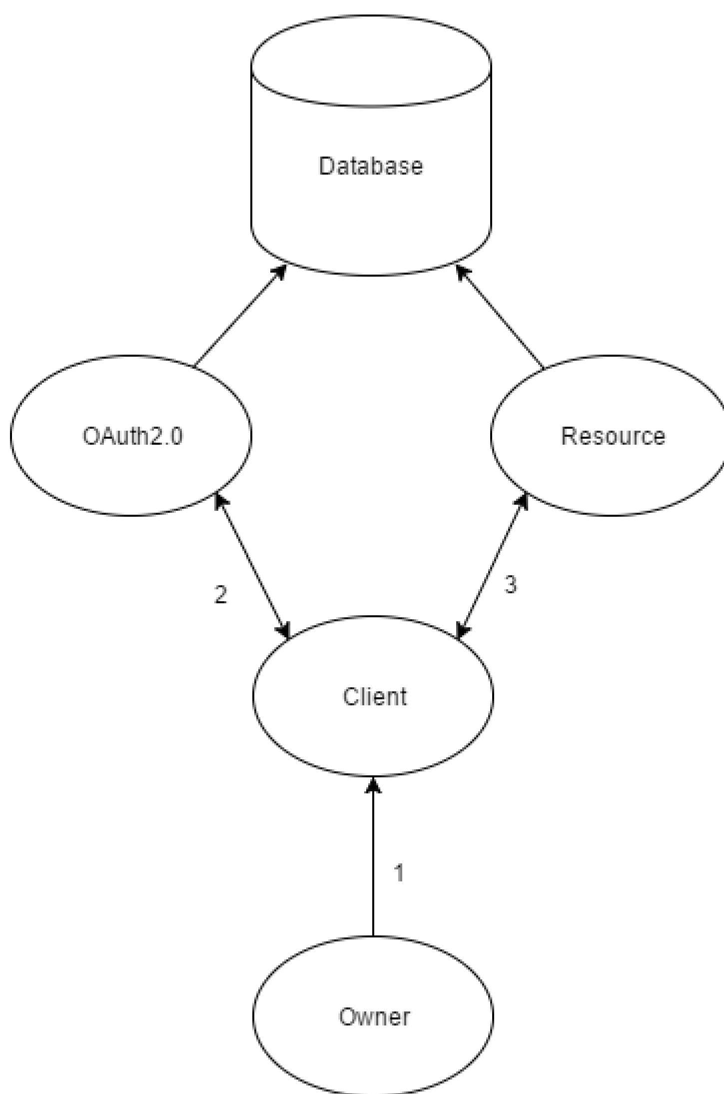
- Authorization code grant: wanneer er wordt ingelogd door bijvoorbeeld een facebook account wordt dit grant-type gebruikt. De gebruiker geeft na het inloggen toestemming om onder andere naam en e-mailadres te gebruiken (Degasperi, sd);
- Implicit grant: dit grant type lijkt op het authorization code grant. Het wordt gebruikt wanneer de client de eigen credentials niet geheim kan houden (Degasperi, sd);
- Resource owner credentials grant: ook wel password grant genoemd. De credentials van de patiënt worden samen met de credentials van de client doorgestuurd naar de server voor authenticatie (Degasperi, sd);
- Client credentials grant: bij dit type wordt enkel gebruik gemaakt van de cliënt credentials voor authenticatie (Degasperi, sd);
- Refresh token grant: samen met een access_token wordt er ook een refresh_token meegestuurd na authenticatie (Degasperi, sd).

In dit systeem wordt het resource owner credentials grant gecombineerd met het refresh token grant.

3.6.4. Set-up

Bij het opzetten van het OAuth2.0 protocol worden er in de database verschillende tabellen toegevoegd. Zo is er de tabel oauth_clients. In deze tabel worden alle projecten toegevoegd die beveiligd worden door deze implementatie. Hierbij wordt er een nieuwe client aangemaakt waarbij een client_id en client_secret wordt vastgelegd.

De werking wordt uitgelegd aan de hand van onderstaande figuur. Bij stap 1 zal de owner, in dit systeem een patiënt, de inloggegevens ingeven op de client, hier de mobiele applicatie. Deze gegevens worden in stap 2 verzonden naar het OAuth2.0 protocol. Deze gegevens worden vervolgens gevalideerd waarna er al dan niet een access_token wordt teruggestuurd. Wanneer de client over een geldig access_token bezit kan deze de resource, hier de server van DHARMA, aanroepen in stap 3. Als het access_token overeen komt met het access_token dat voorzien is bij de patiënt worden requests vanaf de client beantwoord met data vanuit de database.



Figuur 30: Set-up OAuth2.0

De configuratie van dit systeem bestaat uit verschillende stappen. Ten eerste wordt er een middleware toegevoegd aan de routes welke aangeeft dat ze worden beveiligd volgens OAuth. Hierdoor is het nodig om een access_token mee te sturen bij elke request.

3.6.5. Authenticatie

De eerste stap voor het toevoegen van de beveiliging is authenticatie van de patiënt. De patiënt moet na het ingeven van een correct username of emailadres in combinatie met het juiste wachtwoord ingelogd worden op de applicatie. OAuth2.0 voorziet verschillende grant-types voor authenticatie. Voor de combinatie username/emailadres met wachtwoord is het resource owner credential grant-type, ook wel password type, voorzien. Voor de gebruiker gaat het inloggen als volgt:

1. Wanneer een gebruiker de app op het mobiele apparaat opent, wordt er gevraagd om hun gebruikersgegevens (gebruikersnaam/e-mailadres en wachtwoord) in te vullen.
2. Wanneer de gebruiker inlogt, wordt er een POST verstuurd van de mobiele app met deze gebruikersgegevens via TLS.
3. De gebruikersgegevens worden gevalideerd. Wanneer dit succesvol is verlopen wordt er een access token gecreëerd voor de gebruiker die vervalt na een bepaalde tijd.
4. Dit acces token wordt op het mobiele apparaat opgeslagen en gebruikt voor toegang te krijgen tot de API service.
5. Zodra dit acces token verloopt, wordt opnieuw gevraagd voor de gebruikersgegevens.

Voor dit principe toe te passen wordt in de config file van OAuth2.0 het grant-type toegevoegd zoals in volgende afbeelding.

```
'password' => [
    'class' => '\League\OAuth2\Server\Grant\PasswordGrant',
    'callback' => 'Dharma\PasswordGrantVerifier@verify',
    'access_token_ttl' => 3600
],
```

Figuur 31: resource owner credentials grant-type (Degasperi, sd)

Na het succesvol inloggen wordt een access_token voorzien die 3600 seconden geldig is. Wanneer een patiënt wilt inloggen wordt de AuthenticatieController aangeroepen met volgende gegevens.


```
{
  "grant_type": "password",
  "client_id": "f3d259ddd3ed8ff3843839b",
  "client_secret": "4c7f6f8fa93d59c45502c0ae8c4a95b",
  "password": "patient",
  "username": "anouk15@gmail.com",
  "phone_token": "eh2KMeYNp5k:APA91bEg-ivQj5PzV2WUNI0wQxFHVSUqpisuBXkXbNbzuHpe2b25c99QE-8Zkez6WSU4QFD5_2nJkvId4zxktZnS7IkjmdZnugPmHb1HBS
      -v5Iy-dgyoooy1H1HEhdYJjjL34BMEHNVq",
  "phone_type": 1
}
```


Figuur 32: gegevens voor aanvraag access_token met resource owner credentials grant-type

Enkel het wachtwoord en de username worden door de patiënt zelf ingegeven. Het grant_type, client_id en client_secret zorgen hierdoor voor een extra beveiliging. Deze worden op de mobiele applicatie bewaard. Wanneer het client_id en het client_secret meegestuurd in de request niet overeen komen met deze in de database zal de patiënt niet kunnen inloggen.

Voor authenticatie van username en wachtwoord wordt een PasswordGrantVerifier aangemaakt in de AuthenticatieController. Via deze PasswordGrantVerifier wordt een verify uitgevoerd op het username en wachtwoord. Wanneer al de gegevens correct zijn wordt er een response ontvangen die er als volgt uitziet.

```
{
  "access_token": "dEI0bk6gWV6vcYLMNzZebVtbK1DQTkMLrcGJtEpa",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "5qQfjU8FOPPBhpsjfE2hBqtkeYXw4BvqtUMPbloe"
```

Figuur 33: verkregen access_token en refresh_token na authenticatie

3.6.6. Aanroepen API

Met dit access_token kan de patiënt gedurende 3600 seconden requests versturen naar de server.

Voor de patiënt is het niet gebruiksvriendelijk als opnieuw inloggen nodig is wanneer het geldig access_token vervalt. Hiervoor wordt het protocol uitgebreid met het grant_type voor refresh_tokens.

```
'refresh_token' => [
'class' => '\League\OAuth2\Server\Grant\RefreshTokenGrant',
'access_token_ttl' => 3600,
'refresh_token_ttl' => 31557600
]
```

Figuur 34: refresh token grant-type

Wanneer het access_token vervalt zal er een request worden verstuurd naar de OAuth2.0 server dat als volgt wordt opgebouwd.

```
{
  "grant_type": "refresh_token",
  "client_id": "f3d259ddd3ed8ff3843839b",
  "client_secret": "4c7f6f8fa93d59c45502c0ae8c4a95b",
  "refresh_token": "5qQfjU8FOPPBhpsjfE2hBqtkeYXw4BvqtUMPbloe"
}
```

Figuur 35: gegevens voor aanvraag nieuw access_token met een refresh_token

De response op dit request zal opnieuw een refresh_token en een access_token bevatten welke verder worden gebruikt.

4. Resultaten

4.1. API voor DHARMA

Een API is ontwikkeld waarmee:

- ✓ De patiënt vragen kan ophalen;
- ✓ De patiënt antwoorden op de vragen kan terugsturen naar de server;
- ✓ De patiënt meetdata kan doorsturen naar de server;
- ✓ De patiënt kan inloggen aan de hand van de juiste gegevens;
- ✓ Kan worden nagegaan of een vragenlijst reeds is ingevuld.

4.2. Verzenden van pushnotificaties

Het is mogelijk om een pushnotifactie te verzenden naar het mobiel apparaat van de patiënt wanneer er een vragenlijst moet worden ingevuld. Dit naar zowel Android als iOS toestellen.

4.3. Gebruik van schedules

Onderzoekers kunnen zowel patiënten als studies koppelen aan een vragenlijst. Zo kunnen patiënten om de dag, week of maand standaard een vragenlijst toegestuurd krijgen.

4.4. Beveiliging met OAuth2.0

OAuth2.0 is geïmplementeerd in het systeem:

- ✓ Patiënten moeten zich volgens het resource owner credentials grant authentiseren;
- ✓ Aan de hand van een access_token wordt nagegaan of de patiënt recht heeft om de API aan te roepen;
- ✓ Er wordt een controle uitgevoerd of de patiënt na het succesvol aanroepen van de API recht heeft om een bepaalde vragenlijst op te halen of antwoorden te versturen;
- ✓ Met het refresh token grant worden nieuwe access_tokens aangevraagd wanneer de vorige tokens zijn vervallen.

5. Besluit

5.1. Persoonlijke ervaring

Door ter plaatse, in het ZOL, aan het project te werken leer je snel bij hoe het er in het bedrijfsleven aan toe gaat. Door het dagelijks houden van een scrum worden de projecten goed opgevolgd en besproken. Met deze ervaring achter de rug kan je voorbereider naar het echte bedrijfsleven stappen.

Een belangrijk punt binnen de bedrijfswereld is dat communicatie zeer belangrijk is. Wanneer er in teamverband wordt gewerkt is het nodig om elkaar op de hoogte te houden van vorderingen binnen het project alsook nieuw ideeën met elkaar te bespreken.

5.2. Conclusie en mogelijke uitbreiding

De eerste doelstelling was een databasemodel ontwerpen. In overleg met mede-ontwikkelaars is een databasemodel uitgewerkt dat voorziet aan de nodige eisen van het systeem.

Een volgende doelstelling van deze thesis was, met het gekozen framework (Gregoire, juni 2016), een API ontwikkelen die voor een verbinding zorgt tussen de server en de applicatie. Deze API kan verder worden uitgebreid naar de wensen van de onderzoekers. Ook is zowel het gebruik van pushnotificaties als het gebruik van schedules voorzien.

Tijdens de masterproef is Firebase Cloud Messaging (FCM) uitgekomen ter vervanging van Google Cloud Messaging (GCM). Deze aanpassingen kan worden doorgevoerd voor het verzenden van pushnotificaties naar Android toestellen.

Een volgende doelstelling was het beveiligen van de API. Er is onderzoek gedaan naar de verschillende protocollen waarna de keuze is gemaakt om OAuth2.0 te gebruiken. Oauth2.0 is een flexibel protocol waarmee de beveiliging van de API op een overzichtelijke manier kan worden geïmplementeerd. Dit door gebruik te maken van de verschillende grant-types.

De beveiliging van OAuth2.0 kan worden uitgebreid door op meerdere gegevens te controleren. Zo kan er bij elke request een controle worden uitgevoerd op het unieke client_id. Ook kan OAuth2.0 worden uitgebreid met JSON Web Tokens. Veder kunnen Scopes worden toegevoegd aan de beveiliging waarmee per patiënt rechten worden gedefinieerd.

Bibliografie

Apple Push Notification Service. (sd). Opgeroepen op april 2016, van developer.apple: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

Architecture of Laravel Applications. (sd). Opgeroepen op februari 2016, van laravelbook: http://laravelbook.com/laravel-architecture/

Artisan Console. (sd). Opgeroepen op februari 2016, van laravel: https://laravel.com/docs/5.2/artisan

AviD. (2010). *Is BASIC-Auth secure if done over HTTPS?* Opgeroepen op juni 2016, van security.stackexchange: http://security.stackexchange.com/questions/988/is-basic-auth-secure-if-done-over-https

Badri. (2012). *Basic Authentication with ASP.NET Web API*. Opgeroepen op juni 2016, van lbadri.wordpress: https://lbadri.wordpress.com/2012/07/26/basic-authentication-with-aspnet-web-api/

Constraints, F. K. (sd). Opgeroepen op februari 2016, van laravel: https://laravel.com/docs/5.2/migrations#foreign-key-constraints

Database: Migrations. (sd). Opgeroepen op februari 2016, van laravel: https://laravel.com/docs/5.2/migrations

Degasperi, L. (sd). *oauth2-server-laravel*. Opgeroepen op juni 2016, van github: https://github.com/lucadegasperi/oauth2-server-laravel

Gregoire, M. (juni 2016). *Dynamische vragenlijsten in Laravel.* België: Hogeschool PXL, ZOL.

Hazlewood, L. (2012). *Top Six Reasons to Use API Keys (and How!)*. Opgeroepen op juni 2016, van stormpath: https://stormpath.com/blog/top-six-reasons-use-api-keys-and-how

HTTP Routing. (sd). Opgeroepen op februari 2016, van laravel: https://laravel.com/docs/5.2/routing

Introduction. (sd). Opgeroepen op februari 2016, van laravel.com: https://laravel.com/docs/4.2/introduction

Kiran V S, S. S. (sd). *State-of-the ART Location Recognization Development Over Smartphones by Integrating Cloud Based LBS's*. Opgeroepen op april 2016, van rroij: http://www.rroij.com/open-access/stateofthe-art-location-recognizationdevelopment-over-smartphones-byintegrating-cloud-based-lbss.php?aid=45885

Mangal, A. (2013). *OAuth 2.0 - The Good, The Bad & The Ugly*. Opgeroepen op juni 2016, van tutsplus: http://code.tutsplus.com/articles/oauth-20-the-good-the-bad-the-ugly--net-33216

Mayko, L. (2015). *Choosing an OAuth Type for Your API*. Opgeroepen op juni 2016, van api2cart: https://www.api2cart.com/blog/choosing-oauth-type-api/

McKinley, H. L. (2003). *SSL and TLS: A Beginners Guide*. Opgeroepen op juni 2016, van sans: https://www.sans.org/reading-room/whitepapers/protocols/ssl-tls-beginners-guide-1029

Nils Adermann, J. B. (sd). *Dependency Manager for PHP*. Opgeroepen op februari 2016, van getcomposer: https://getcomposer.org/

Pelssers, M. (juni 2015). *Achieving a mobile care solution enabling smart and responsive interaction.* België: Hogeschool PXL, ZOL.

RESTful Resource Controllers. (sd). Opgeroepen op februari 2016, van laravel: https://laravel.com/docs/5.2/controllers#restful-resource-controllers

Rouse, M. (sd). *Transport Layer Security (TLS)*. Opgeroepen op juni 2016, van searchsecurity.techtarget: http://searchsecurity.techtarget.com/definition/Transport-Layer-Security-TLS

Ryan. (2016). *Sending push notifications with Google Cloud Messaging and Laravel 5*. Opgeroepen op april 2016, van ryanstelmat: http://ryanstelmat.com/sending-push-notifications-with-google-cloud-messaging-and-laravel/

Sandoval, K. (2015). *API Security: The 4 Defenses of The API Stronghold*. Opgeroepen op juni 2016, van nordicapis: http://nordicapis.com/api-security-the-4-defenses-of-the-api-stronghold/

Sandovel, K. (2015). *API Keys ≠ Security: Why API Keys Are Not Enough*. Opgeroepen op juni 2016, van nordicapis: http://nordicapis.com/why-api-keys-are-not-enough/

Sevilleja, C. (2015). *The Ins and Outs of Token Based Authentication*. Opgeroepen op juni 2016, van scotch: https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication

Stateless protocol. (sd). Opgeroepen op augustus 2016, van wikipedia: https://en.wikipedia.org/wiki/Stateless_protocol

Stormpath, T. (2013). *Secure Your REST API… The Right Way*. Opgeroepen op februari, van stormpath: https://stormpath.com/blog/secure-your-rest-api-right-way

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Ontwikkeling van een API voor het stellen van dynamische vragen met behulp van de smartphone

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2016**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt
behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -,
vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten
verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Deprez, Michiel

Datum: **29/08/2016**