

{ REST }



GraphQL

IS HET RENDABEL OM VAN EEN BESTAANDE REST OMGEVING OVER TE SCHAKELN NAAR GRAPHQL EN WELKE VOORDELEN BRENGT DIT MET ZICH MEE?

INTERNE PROMOTOR: STIJN WALCARIUS
EXTERNE PROMOTOR: SVEN ROEGIERS

ONDERZOEKSVRAAG UITGEVOERD DOOR

TIJL DE LANDTSHEER

VOOR HET BEHALEN VAN DE GRAAD VAN BACHELOR IN DE

MULTIMEDIA EN COMMUNICATIETECHNOLOGIE

HOWEST | 2019-2020

Woord vooraf

Ter afsluiting van mijn studies Media & Communication Technologies met uitstroomprofiel 'Web App Developer', schreef ik deze bachelorproef. De onderzoeksvraag kwam voor het eerst aan bod in de lessen back-end development.

Deze bachelorproef bespreekt eerst de research en technische demo uit de module 'Project 4', waarbij een vergelijkende studie twee gelijkaardige webapplicaties behandelt. Een ervan maakt gebruik van GraphQL en de andere van REST.

Na de analyse reflecteer ik het verkregen resultaat met het werkveld. Hiervoor nam ik contact op met Bart Wullems en Ben Elsen, die GraphQL gebruiken in een bedrijfscontext.

Graag bedank ik Stijn Walcarius, mijn promotor die mijn vooruitgang evalueerde bij zowel mijn technische uitwerking als de bachelorproef en telkens bijstuurde waar nodig.

Daarnaast wil ik Johan Vannieuwenhuyse bedanken, die het onderwerp van deze bachelorproef voor het eerst aan bod bracht en mij motiveerde hier onderzoek rond te verrichten. Ook bedank ik Johan Vannieuwenhuyse voor de lessen "full stack development", waardoor ik over een goede basis beschikte om zowel met mijn project als mijn stage snel vooruitgang te boeken.

Speciale dank gaat ook uit naar District01, om mij de kans te geven ervaring op te doen in het werkveld gedurende een periode van vijftien weken. Gezien de omstandigheden was het onmogelijk om gedurende heel deze periode op kantoor te werken, maar ik werd uitstekend begeleid bij het telewerken. Tot slot bedank ik mijn stagementor Sven Roegiers voor de begeleiding van zowel mijn stage als mijn bachelorproef.

19 april 2020,

Tijl De Landtsheer

Abstract

Deze bachelorproef onderzoekt de vraag “Is het rendabel om van een bestaande REST omgeving over te schakelen naar GraphQL en welke voordelen brengt dit met zich mee?”. Er wordt ook onderzocht welke mogelijkheden er zijn om te migreren en hoe we de service kunnen beschermen tegen de kwetsbaarheden die het gebruik van GraphQL als querytaal met zich meebrengt.

Voor het technisch onderzoek wordt een vergelijkende studie gemaakt, die twee gelijke webapplicaties behandelt. De ene maakt gebruik van REST en de andere van GraphQL. De toepassing is een webapplicatie die zowel de huidige weersomstandigheden als die van komende twee weken weergeeft. Er wordt aandacht besteed aan de ontwikkeling van de server, de implementatie van de API en de responstijden van beide varianten.

Het onderzoek vooraf en de technische demo bewijzen dat door de grootte van de response te reduceren, de responstijd in de meeste gevallen sneller is dan bij de REST API. Dit onderzoek toont ook het belang van de serverkeuze aan. Deze heeft niet alleen impact op de ontwikkeling van de server en de implementatie van de API, maar ook op de responstijden van queries en mutations.

De vrijheid van de client door het gebruik van een querytaal, zorgt voor kwetsbaarheden waartegen de server beschermd moet worden om een denial of service te voorkomen. Ook caching wordt moeilijker door het gebruik van de single access point.

De migratie van REST naar GraphQL, al dan niet door het gebruik van GraphQL als wrapper, zal in veel gevallen rendabel zijn door de gereduceerde responsgrootte, die voortkomt uit de eliminatie van overfetching. De client is minder afhankelijk van de server doordat die zelf bepaalt welke data de response zal bevatten.

Inhoudsopgave

Woord vooraf	1
Abstract.....	3
Inhoudsopgave	4
Figurenlijst.....	5
Lijst met afkortingen	6
Verklarende woordenlijst	7
1 Inleiding.....	8
2 Research.....	9
2.1 Wat zijn de voor- en nadelen van het gebruik van GraphQL als querytaal?	9
2.2 Hoe staan GraphQL en REST tegenover elkaar?	10
2.3 Hoe staan GraphQL en gRPC tegenover elkaar?	12
2.4 In welke mate is een single access point beschermd tegen een groot aantal requests komende van de client?	13
2.5 Maakt het gebruik van een querytaal de toepassing kwetsbaarder?	13
2.6 Autorisatie in GraphQL.....	15
2.7 Welke technologie is sneller op vlak van ontwikkeling en opvraging?.....	16
2.8 In hoeverre kan een combinatie van API-technieken een webapplicatie performanter maken?.....	19
2.9 Hoe is het gebruik van GraphQL verlopen sinds de initial release in 2015?	20
2.10 Hoe zal het gebruik van GraphQL evolueren naar de toekomst toe?	21
3 Technisch onderzoek	22
3.1 Werkwijze	22
3.2 Opstelling	23
3.3 Ontwikkeling API.....	24
3.4 Implementatie API	25
3.5 Responstijden	27
4 Reflectie	29
4.1 Resultaten technisch onderzoek	29
4.2 Bruikbaarheid projectresultaat	30
4.3 Overweging implementatie in bedrijfscontext	30
4.4 Alternatieven community	31
4.5 Economisch en maatschappelijke meerwaarde van GraphQL.....	32
4.6 Suggestie vervolgonderzoek	32
5 Advies	33
6 Conclusie	35
7 Literatuurlijst.....	36
8 Bijlages	39
8.1 Verslag Cybercrime Unit	39
8.2 Verslag In The Pocket	41
8.3 Project 4 – Installatiehandleiding	43
8.4 Project 4 – Gebruikershandleiding	46

Figurenlijst

Figuur 1 - REST/GraphQL endpoints [9].....	11
Figuur 2 - gRPC Client-server communicatie [12]	12
Figuur 3 - Query complexity GraphQL	14
Figuur 4 - Authenticatie GraphQL server [22]	15
Figuur 5 - Autorisatie in GraphQL layer	15
Figuur 6 - Aantal API calls per functie (REST - GraphQL) [23]	16
Figuur 7 - Aantal velden teruggegeven door API calls per functie (REST - GraphQL) [23]	16
Figuur 8 - GraphQL als wrapper [29]	19
Figuur 9 - Downloads van javascript implementatie GraphQL volgens npmtrends [33]	20
Figuur 10 - Technische demo: schermafbeelding 1	22
Figuur 11 - Technische demo: schemafbeelding 2.....	22
Figuur 12 - Technische demo: schermafbeelding 3	22
Figuur 13 - Schema opstelling technische onderzoek.....	23
Figuur 14 - Meetresultaten responstijden GraphQL - REST	27
Figuur 15 - Overzicht snelheden javascript servers GraphQL – REST [25].....	28

Lijst met afkortingen

API	Application Programming Interface
AWS	Amazon Web Services
CRUD	Create Read Update Delete
DoS	Denial of Service
DTO	Data Transfer Object
GraphQL	Graph Query Language
GUI	Graphical User Interface
HATEOAS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	Javascript Object Notation
NPM	Node Package Manager
OAS	Open Api Specification
ORM	Object-relational mapping
REST	Representational state transfer
RPC	Remote Procedure Call
SDL	Schema Definition Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

Verklarende woordenlijst

Authenticatie: Authenticatie verwijst naar het proces van het confirmeren van een gebruiker zijn identiteit.[1]

Autorisatie: Autorisatie verwijst naar het proces van het verifiëren waar de gebruiker toegang toe heeft. [1]

Denial Of Service (DoS): Een denial of service attack zorgt ervoor dat informatiesystemen, toestellen of andere netwerkbronnen ontoegankelijk worden voor buitenstaanders door een te zware belasting. [2]

Distributed Denial of Service (DDoS): Een distributed denial of service komt voor wanneer meerdere machines samenwerken om een denial of service op eenzelfde target te veroorzaken. [2]

Middleware: Middleware komt voor in de http-pipeline. Één input wordt omgezet naar één output (vb: error messaging, authentication approvals, encrypting, logging). Middleware componenten zijn functies die reageren en antwoorden op een http-request en het resultaat uiteindelijk aan de http- response bezorgen.

Object Relational Mapper (ORM): ORM zorgt voor een eenduidige vertaling van de dataobjecten aan OOP kant op de applicatieserver naar de tabellen of documenten op de databaseserver.

Open API Specification (OAS): De open API specification zorgt voor een interface voor RESTful APIs die het mogelijk maakt voor zowel mensen als computers om de mogelijkheden van de service te raadplegen zonder door de source-code of documentatie te moeten lopen. [3]

Throughput: De hoeveelheid data die getransporteerd wordt van de ene locatie naar de andere in een bepaalde tijdsperiode.

1 Inleiding

REST is een architectuur waarmee de meeste developers ongetwijfeld al gewerkt hebben. De sterke opvolger van SOAP is al sedert 2000 een standaard wanneer sprake is van een webAPI. In 2015 maakte Facebook GraphQL open source en is er aan een sneltempo een community opgebouwd, waardoor meer en meer libraries en tooling beschikbaar werden.

Toen GraphQL voor het eerst aan bod kwam in de les, werd al snel duidelijk waarom meer en meer geopteerd wordt voor deze querytaal. De client bepaalt in de request welke data teruggestuurd wordt in de response. Hierdoor kan de hoeveelheid data die de response bevat sterk gereduceerd worden. Toch blijft REST een standaard bij het bouwen van webAPIs. Deze bachelorproef onderzoekt de vraag 'Is het rendabel om van een bestaande REST omgeving over te schakelen naar GraphQL en welke voordelen brengt dit met zich mee?'.

Binnen mijn stagebedrijf wordt standaard REST gebruikt om webAPIs te schrijven. Het is een standaard die een in bedrijfscontext nog steeds voldoet aan de behoeften en veel voordelen biedt. GraphQL werd nog niet eerder gebruikt, maar dit wordt wel actueel opgevolgd en de onderzoeksvraag is hier zeker relevant.

Dit onderzoek geeft niet enkel een antwoord op de hoofdvraag, maar zal ook volgende deelvragen beantwoorden:

- Wat zijn de voordelen en nadelen van het gebruik van GraphQL als querytaal?
- Hoe staan GraphQL en REST tegenover elkaar?
- Hoe staan GraphQL en gRPC tegenover elkaar?
- Welke technologie is sneller op vlak van ontwikkeling en opvraging?
- In welke mate is een single access point beschermd tegen een groot aantal requests komende van de client?
- Maakt het gebruik van een querytaal de toepassing kwetsbaarder?
- Hoe zit het met autorisatie bij GraphQL?
- Welke factoren bepalen de keuze voor GraphQL?
- In hoeverre kan een combinatie van API-technieken een webapplicatie performanter maken?
- Hoe is het gebruik van GraphQL verlopen sinds de initiële release in 2015?
- Hoe zal het gebruik van GraphQL evolueren naar de toekomst toe?

De onderzoeksgegevens in deze bachelorproef zijn afkomstig van verschillende onderzoeken die het gebruik van GraphQL bespreken. Er werden in de technische demo twee webapplicaties gebouwd, waarbij de ene gebruik maakte van REST en de andere van GraphQL. Zowel de snelheid als de ontwikkeling werd besproken. De resultaten van het technisch onderzoek komen overeen met de ondervindingen van andere geraadpleegde bronnen en de ondervindingen van ontwikkelaars die GraphQL gebruiken in een bedrijfscontext.

2 Research

2.1 Wat zijn de voor- en nadelen van het gebruik van GraphQL als querytaal?

GraphQL is een opensource querytaal voor APIs, ontwikkeld door Facebook. GraphQL is ontwikkeld om te zorgen voor meer flexibiliteit en efficiëntie bij client-server communicatie. De front-end developer krijgt zelf de mogelijkheid om queries of mutations op te stellen. GraphQL voert deze queries uit op de server en geeft enkel de data gedefinieerd door een type system terug. Een type system beschrijft welke types objecten de GraphQL server kan teruggeven. Deze beschrijving van de mogelijkheden van de server wordt ook een schema genoemd. Het schema dient als contract tussen de client en de server. [4]

Queries worden geschreven in een geneste object structuur, waarna de GraphQL een object type teruggeeft in JSON-formaat. Doordat op de server schema's en types worden opgesteld, ziet de client welke GraphQL services beschikbaar zijn vanop één endpoint. De client kan, zoals hieronder weergegeven, kiezen welke velden hij opvraagt, waardoor geen overbodige data meegestuurd wordt in de response.

```
current(cityname: "Brussels"){
  time
  city{
    name
    country
  }
  main {
    temp
    pressure
    humidity
  }
  wind{
    speed
  }
  clouds{
    all
  }
}
```

```
"current": {
  "time": 1489490917,
  "city": {
    "name": "Brussels",
    "country": "BE"
  },
  "main": {
    "temp": 285.63,
    "pressure": 1032,
    "humidity": 71
  },
  "wind": {
    "speed": 4.1
  },
  "clouds": {
    "all": 0
  }
}
```

Om server-side data aan te passen wordt in GraphQL gebruik gemaakt van mutations. Net zoals bij queries kan een mutation een Object Type teruggeven. De client kiest zelf welke velden de response bevat na een succesvolle mutation. Alle queries en mutations worden teruggestuurd naar één endpoint. Dit zorgt voor een eenvoudige toegankelijkheid zonder dat voor elke resource een andere endpoint moet aangesproken worden.

GraphQL heeft vier grote voordelen, hierbij verwijs ik naar de documentatie van GraphQL. [4]

- 1) De front-end developer kiest zelf welke data hij nodig heeft en krijgt ook enkel deze data terug. Dit resulteert in het versturen van een kleinere hoeveelheid data bij het oproepen van een query of mutation.
- 2) Door het gebruik van het type system en de beschrijving ervan in schema's, kunnen alle requests worden verstuurd naar één endpoint. Dit zorgt voor een eenvoudige toegankelijkheid.
- 3) Doordat alle data verstuurd wordt naar dezelfde endpoint kunnen in een call ook verschillende resources worden aangesproken, met als gevolg dat er minder round-trips moeten gebeuren.
- 4) GraphQL genereert automatisch documentatie voor de APIs gebaseerd op het schema. Dit zorgt voor een consistente documentatie.

Het gebruik van GraphQL heeft naast deze voordelen ook verschillende nadelen. De drie grootste problemen waar developers tegenaan lopen, resulteren in extra programmeerwerk.[5]

- 1) Het gebruik van de single access point wil zeggen dat er geen gebruik gemaakt wordt van de native http-caching om het meermaals ophalen van dezelfde data tegen te gaan. Er moet dus caching worden voorzien op de client.
- 2) De gebruiker bepaalt zelf welke data hij terugkrijgt en kan bijgevolg de server zwaar belasten door een grote hoeveelheid aan data op te vragen in een request. Dit moet afgehandeld worden, zodat kwaadaardige queries de server niet vertragen of stilleggen.
- 3) Bij GraphQL resulteert elke query of mutation in status code 200. Wanneer de query niet succesvol is, komt bovenaan de response een object, met daarin een array die de verschillende error messages bevat.

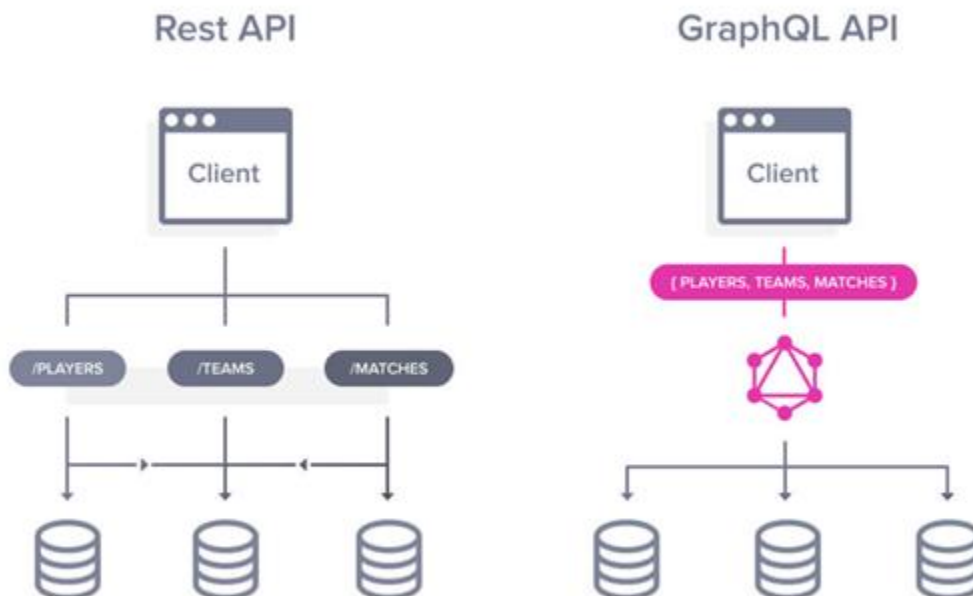
2.2 Hoe staan GraphQL en REST tegenover elkaar?

REST is een architectuur voor stateless client/server toepassingen. De data wordt getransporteerd over HTTP1 en geserialiseerd naar JSON, HTML of XML, waarbij JSON het meest voorkomende formaat is. Er wordt gebruik gemaakt van een URI om toegang te krijgen tot de data. Deze resources kunnen geïdentificeerd worden met een unieke URL. HTTP verbs (GET, POST, UPDATE & DELETE) worden gebruikt voor interactie met de resources. REST is een specificatie van hoe de server zijn data moet blootstellen voor de client. [3]

Zoals elke andere architectuur is REST gebaseerd op voorwaarden waaraan moet worden voldaan als een interface als REST wil gerefereerd worden. Tegenwoordig wordt een API al snel RESTful genoemd, terwijl die vaak niet voldoet aan alle voorwaarden waaraan een RESTful API moet voldoen. Om de voordelen van REST te begrijpen, bekijken we eerst deze voorwaarden. Zo kunnen we de verschillen in werking vergelijken. [3] [4] [8]

1. **Client-server** – Door afscheiding van userinterface concerns en data storage concerns (separation of concerns) wordt de portability van de userinterface over verschillende platformen verbeterd en wordt schaalbaarheid van de server verbeterd.
2. **Stateless** – Elke request van de client naar een server moet alle nodige informatie bevatten om de request te begrijpen, dit maakt de request onafhankelijk. Bij de server wordt geen gebruik gemaakt van opgeslagen context. Session state wordt volledig door de client behandeld.
3. **Cacheable** – De data in de response moet impliciet of expliciet gelabeld worden als cacheable of non-cacheable. Als een response cacheable is dan krijgt de client het recht om dezelfde data te herbruiken voor latere equivalente requests.
4. **Uniform interface** – Om aan een uniforme interface te voldoen zijn er verschillende voorwaarden: identificatie van de resources, manipulatie van de resources door representations, zelf beschrijvende berichten en hypermedia as the engine of application state (RESTful).

5. **Layered system** – In een layered system weet de client niet of die rechtstreeks verbonden is met de server of door een intermediair. Een intermediair kan securitylagen toevoegen voor de server.
6. **Code-on-demand** (optioneel) – REST biedt de client de mogelijkheid om functionaliteiten uit te breiden door het downloaden en uitvoeren van code in de vorm van een script of applet.



Figuur 1 - REST/GraphQL endpoints [9]

GraphQL voldoet aan twee van deze voorwaarden, met name client-server en stateless. Caching moet worden voorzien op de client zelf, waarbij geen gebruik gemaakt kan worden van http-caching. Een uniforme interface, waarbij elke URI een beschrijving of identificatie is van de resource, wordt bij GraphQL vervangen door één endpoint. Een layered system is iets wat GraphQL niet heeft door zijn open schema eigenschap. Ook het optionele code-on-demand ontbreekt bij GraphQL, de mogelijkheid om additionele server side logica toe te voegen is er niet.

Bij GraphQL hoeft slechts één endpoint aangesproken te worden, bij een REST API worden verschillende endpoints aangesproken. De URL van de endpoint is bij REST de globally unique identifier die gebruikt wordt op de client om aan caching te doen. GraphQL heeft als voordeel dat men voor alle requests naar dezelfde endpoint kan verwijzen, wat zorgt voor een eenvoudige toegankelijkheid naar de server. Met als nadeel dat geen ingebouwde caching ondersteuning aanwezig is. Zo moet de front-end developer zelf zorgen voor caching. De keuze van de server kan bepalen hoe deze geïmplementeerd wordt, aan de hand van globally unique IDs. [10]

Bij REST zijn over- en under-fetching vaak terugkomende termen. Er wordt in een call te veel of te weinig data meegestuurd. Wanneer er te veel data wordt meegestuurd, wordt de grootte van de response onnodig groot. Wanneer er te weinig data wordt meegestuurd, moet een nieuwe call gebeuren en moet er een nieuwe connectie gemaakt worden. Het gebruik van DTOs of response models bij REST volstaat niet om dit tegen te gaan. De server bepaalt nog steeds welke data de response bevat. Over-fetching wordt bij GraphQL vermeden, doordat de gebruiker zelf de vrijheid krijgt om te kiezen welke data hij opvraagt. Under-fetching wordt bij GraphQL vermeden doordat verschillende resources in een request kunnen opgevraagd worden op dezelfde endpoint.

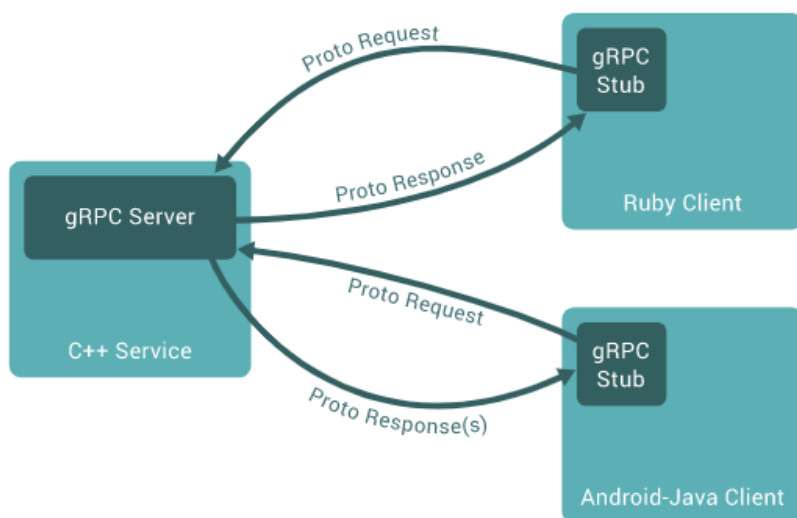
Doordat bij GraphQL verschillende resources kunnen aangesproken worden in een round-trip, kan een request ook resulteren in meerdere verschillende statuscodes. Wanneer een deel van de query faalt, bevat de response als eerste veld een array met de verschillende status codes en error messages. [10]

GraphQL lost dus verschillende tekortkomingen en inefficiënties op waarmee ontwikkelaars geconfronteerd werden bij het interacteren met een REST API. Met als nadeel dat vrijheid aan de kant van de client bij GraphQL voor problemen zorgt die bij REST niet voorkomen.

2.3 Hoe staan GraphQL en gRPC tegenover elkaar?

gRPC is een open source RPC framework, ontwikkeld door Google. Het principe van RPC (Remote Procedure Calls) laat toe om vanuit de server een functie op de client te activeren en omgekeerd. Dit is te vergelijken met een sockets werking. Communicatie tussen client en server gebeurt via het protobuf protocol. Het gebruik van protocol buffers maakt het mogelijk om code te genereren voor elke programmeertaal. [8]

gRPC maakt in tegenstelling tot GraphQL gebruik van HTTP/2 transport. Hierdoor kunnen meerdere binaire streams gemultiplexed worden over een TCP-connectie. De latency daalt en downloaden gaat sneller. Aangezien bij HTTP een TCP-handshake nodig is bij elke request en HTTP/2 gebruik maakt van streaming, is er een grote snelheidswinst. Hoewel de response van een HTTP/1 API meestal een JSON-format bevat als payload, gebruikt gRPC Protobuf messages. Dit is een strongly typed bericht dat automatisch kan worden omgezet naar elke programmeertaal. [11]



Figuur 2 - gRPC Client-server communicatie [12]

Het gebruik van gRPC is tot op heden meer gebruikt voor interne services en IOT. Dit komt omdat gRPC in de browser nog niet ondersteund is. Als we een gRPC service willen gebruiken in een webapplicatie, kunnen we een gRPC aan de hand van een REST-gateway beschikbaar stellen, waarbij verschillende voordelen van het framework wegvallen door het gebruik van HTTP/1. Er worden ook terug verschillende endpoints aangesproken en er moet weer aan JSON-parsing gedaan worden.

Om via de browser gRPC te kunnen gebruiken, kan ook gebruik gemaakt worden van gRPC-web, met een proxy om te converteren van HTTP naar HTTP/2. Deze biedt een gelimiteerde browser-ondersteuning, waarbij client en bi-directional streaming niet ondersteund is. Ook server streaming is

beperkt ondersteund. gRPC-messages zijn ook gecodeerd, waardoor de requests in binair format niet leesbaar zijn voor de gebruikers. [13]

gRPC is snel en het biedt in bepaalde scenario's veel voordelen (vooral op vlak van snelheid), maar meer tegenover HTTP APIs (REST, GraphQL, ...) in een interne microservice IOT context. In de zin van een webAPI is de browser nog niet ver genoeg ontwikkeld om gebruik te maken van gRPC. Daarnaast zou GraphQL nog steeds het grote voordeel hebben dat de client zelf bepaalt welke data de response bevat.

2.4 In welke mate is een single access point beschermd tegen een groot aantal requests komende van de client?

In een REST API wordt een rate-limit ingesteld op het aantal request, komende van dezelfde client of van hetzelfde IP-adres, om DoS-attacks tegen te gaan. Wanneer de rate limit overschreden wordt resulteert elke endpoint gewoonlijk in een status code 429 (too many requests) of kan de client connectie gethrottled worden. Er kunnen verschillende limieten ingesteld worden voor de endpoints. [14]

Bij GraphQL is het limiteren van het aantal requests niet voldoende om de server te beschermen, aangezien in een query resources meermaals kunnen opgevraagd worden door nesting. Hier moet een rate limit ingesteld worden op het niveau van de queries en mutations of op het niveau van de resolvers. Er bestaan ook verschillende libraries die helpen bij het opzetten van rate-limiting. Op deze manier is de server ook beschermd tegen een groot aantal requests komende van de client. [10] [15]

Throttling kan bij GraphQL gebaseerd worden op server time, de tijd die een query nodig heeft om uitgevoerd te worden. Hoe we de single access point nog kunnen beschermen tegen een groot aantal requests of zware en complexe queries wordt in hoofdstuk 2.5 beschreven.

2.5 Maakt het gebruik van een querytaal de toepassing kwetsbaarder?

GraphQL maakt het mogelijk om als client zelf te bepalen welke data je terugkrijgt voor een request. Het gebruik van een querytaal maakt de toepassing wel kwetsbaarder, doordat de client de vrijheid krijgt om zelf te bepalen welke data de reponse bevat. Zo kan die ook grote en complexe queries gaan uitvoeren op de server. Op deze manier kan de server te zwaar belast worden, waardoor deze vertraagt of faalt. Dit kan zijn door een DoS-attack of slecht geoptimaliseerde queries die zorgen voor een te zware belasting op de server. Er zijn extra maatregelen nodig om de server hiertegen te beschermen. [19] [20]

Een eerste stap zou zijn dat we de grootte van de query limiteren. Aangezien de query als string wordt verstuurd kunnen we de lengte van de string controleren. Daarnaast kan ook best een time-out ingesteld worden op de duur van een query. Dit is een eenvoudige manier om de server te beschermen tegen kwaadaardige queries, die hetzelfde object duizenden keren in één request zou kunnen opvragen.

Verder kunnen we depth-limiting toepassen. Dit kan geïmplementeerd worden door een validation rule toe te voegen. Zo kan de client maar een aantal niveaus diepgaan binnen eenzelfde query. Hoe diep een client kan gaan, hangt af van de grootte van de toepassing en moet goed overwogen worden bij het beveiligen van de server.

Er zijn ook velden binnen een query die complexer zijn om af te handelen dan de rest. Dit is gebaseerd op argumenten. Query complexity moet gelimiteerd worden, omdat ook dit voor overhead zorgt bij de GraphQL Server. [20] [21]

```

query {
  author(id: "abc") {           # complexity: 1
    posts(first: 5) {          # complexity: 5
      title                     # complexity: 1
    }
  }
}

```

Figuur 3 - Query complexity GraphQL

De server kan ook zorgen voor persistente queries van zodra de toepassing op productie wordt geplaatst. Dit zorgt voor het whitelisten van bepaalde queries. In het geval van Apollo Server kan de library `persistgraphql` alle queries exporteren die de client gebruikt in een JSON-bestand. Met als nadeel dat dit zorgt voor inefficiëntie, omdat de service dan minder dynamisch is naar toekomstige aanpassingen toe. Er wordt een vaste lijst van bruikbare queries opgesteld.[19]

Beveiliging tegen deze kwetsbaarheden is bij de opbouw van elke GraphQL service een noodzakelijk element. GraphQL kan onmogelijk zelf bepalen in welke mate de queries beveiligd moeten worden omdat de beveiliging afhangt van de complexiteit van de applicatie. De ontwikkelaar van de GraphQL service bepaalt zelf in hoeverre er maatregelen getroffen moeten worden om ervoor te zorgen dat de server niet faalt bij kwaadaardige queries.

Het is dan ook vanzelfsprekend dat er verschillende libraries zijn die de server hiertegen helpen beschermen, zoals: `graphql-rate-limit`, `graphql-cost-analysis`, `graphql-validation-complexity` en `graphql-query-complexity`. Op deze manier kan de beveiliging van de server snel geconfigureerd worden.

SQL-injection kan net zoals bij REST ook bij GraphQL voorkomen, en ook hier is het de verantwoordelijkheid van de back-end developer om de server hiertegen te beschermen. Deze zou manueel karakters kunnen escaperen, maar wanneer er gebruik wordt gemaakt van een betrouwbare ORM zoals `sequelize`, is de toepassing hier direct tegen beschermd. Hetzelfde geldt voor NoSql-injection. [20]

Het gebruik van een querytaal maakt de toepassing dus wel kwetsbaarder. Er moet aandacht besteed worden aan het beveiligen en limiteren van queries. Dit is een logisch gevolg van de vrijheid die de client krijgt bij het opvragen van data.

2.6 Autorisatie in GraphQL

Autorisatie is een belangrijke factor bij de beslissing om te veranderen naar een nieuwe technologie. Het is ook belangrijk om hier aandacht aan te besteden. Hoewel dit bij GraphQL minder een kwestie is van autorisatie, maar veeleer van de manier waarop autorisatie geïmplementeerd wordt.

In GraphQL worden authenticatie en autorisatie afgehandeld door gebruik te maken van query context. Deze wordt ingevuld met de token uit de autorisatie header van de request. Elke resolver heeft toegang tot de context. Om autorisatie snel toe te passen, kan de code in de resolver worden geplaatst. Dit is perfect mogelijk aangezien de context van daaruit beschikbaar is. In een kleinschalige service is deze implementatie niet verkeerd, maar het probleem dat zich hier voordoet, is dat de autorisatie code telkens gedupliceerd moet worden, met als gevolg dat het kan voorkomen dat autorisatie logica niet overal synchroon is. [21]

```
const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: ({ req }) => {
    const token = req.headers.authorization || '';
    const user = getUser(token);
    return { user };
  },
})
```

Figuur 4 - Authenticatie GraphQL server [22]

```
day(_, args, context){
  if (!context.user) throw new Error('Not authenticated');
  return Day
    .findOne({"city.name": args.cityname})
    .orFail( () => new Error('your document not found') )
},
```

Figuur 5 - Autorisatie in GraphQL layer

Autorisatie hoort eigenlijk thuis in de business logic layer. Hierbij verwijs ik naar de documentatie van GraphQL. Resolvers zorgen voor het afhandelen van de request. Het ophalen en bewerken van data gebeurt in de business logic layer. Zo is er meer herbruikbaarheid en zorgen we ervoor dat de autorisatie logica overal synchroon is, wanneer er een aanpassing gebeurt die invloed heeft op de toegang van de clients. [21]

Naast het gebruik van context, verschilt GraphQL niet veel van vertrouwde omgevingen zoals REST APIs. In de technische demo wordt gebruik gemaakt van JWT-tokens, een methode die al jaren gebruikt wordt bij andere webAPIs. Er kan geverifieerd worden of de client toegang heeft om een query of mutation uit te voeren door de payload van de token, die in de header werd meegestuurd, te bekijken. Hoe autorisatie geïmplementeerd wordt, hangt af van de ontwikkelaar van de service en de complexiteit van de toepassing. Achterliggend kan autorisatie op dezelfde manier geïmplementeerd worden als bij een REST API.

2.7 Welke technologie is sneller op vlak van ontwikkeling en opvraging?

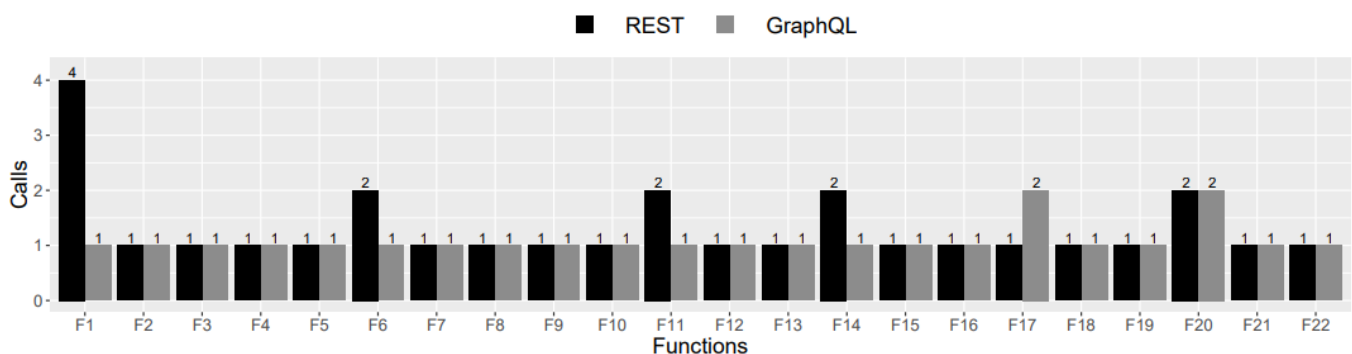
Responstijden

Er werden nu drie verschillende API-technologieën besproken, met elk zijn voor- en nadelen. Eerst wordt de snelheid ervan besproken, vervolgens de snelheid op het vlak van ontwikkeling en implementatie. Dit onderzoek bespreekt wanneer het voordelig is om van REST naar GraphQL over te schakelen en zal minder diep in gaan op gRPC.

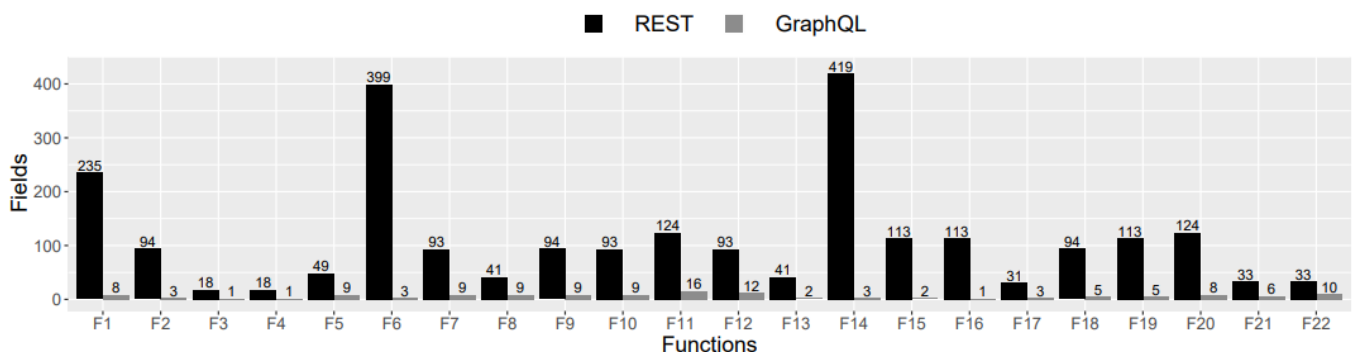
gRPC maakt, in tegenstelling tot REST en GraphQL, gebruik van HTTP/2 en kan dus aan streaming doen. Dit is wat gRPC doet uitblinken. HTTP/2 heeft verschillende voordelen waaronder multiplex streaming en minder latency, waarbij slechts 1 TCP-connectie nodig is. Het gebruik van HTTP/2 in combinatie met de protocol buffer, waarbij de payload binair kleiner is dan bij JSON/ XML of andere formats, maakt gRPC significant sneller dan REST of GraphQL die gebruik maken van het oudere HTTP/1. [13]

REST en GraphQL maken beide gebruik van HTTP/1 en meestal van een JSON payload. Op het vlak van snelheid verschillen ze weinig van elkaar, maar doordat GraphQL de client de mogelijkheid biedt om zelf te bepalen welke data de response bevat, wordt de response kleiner en dus ook sneller. Dit komt omdat de hoeveelheid data die in een response wordt meegestuurd beperkt wordt tot het minimum, waardoor over-fetching wordt vermeden.

In een studie van ASERG Group, Department of computer science werd bestudeerd hoeveel invloed GraphQL heeft tegenover REST op het vlak van round-trips en payload. Hieruit bleek dat documenten teruggegeven door de REST APIs op het vlak van het aantal velden 94% en op het vlak van het aantal bytes 99% gereduceerd kunnen worden na de migratie naar GraphQL. [23]



Figuur 6 - Aantal API calls per functie (REST - GraphQL) [23]



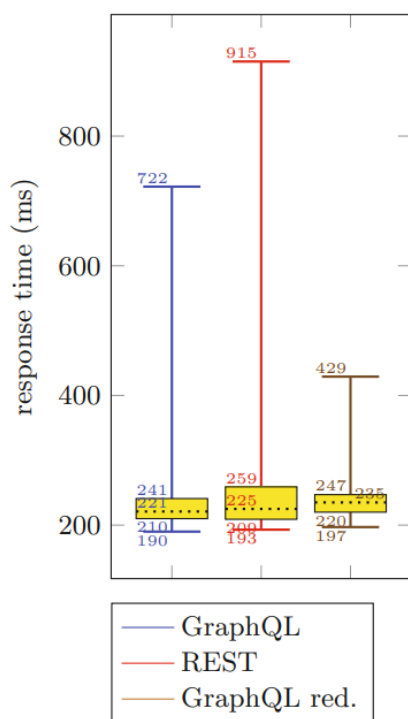
Figuur 7 - Aantal velden teruggegeven door API calls per functie (REST - GraphQL) [23]

In een onderzoek van Nadesh Eeda werden de laadtijden van een real-time dashboard vergeleken. Ook hieruit bleek dat de laadtijd van de pagina door het gebruik van GraphQL 245% sneller was. Hier

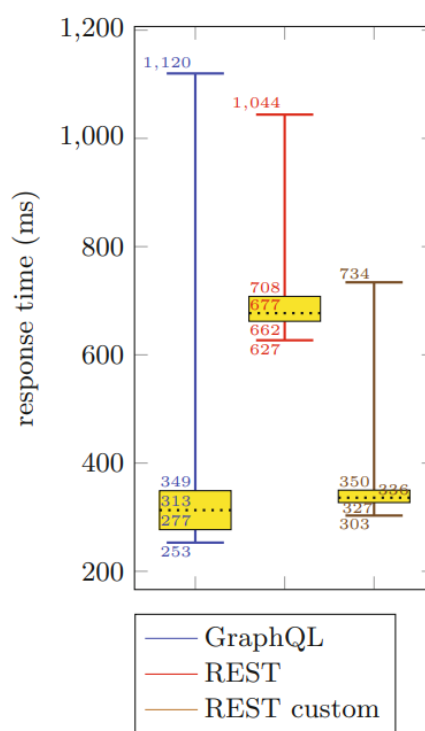
fungeerde GraphQL als gateway voor een REST endpoint, een SOAP endpoint en externe services. De snelheidswinst is voor het grootste deel te danken aan de eliminatie van over-fetching, waardoor er minder data over het netwerk wordt verstuurd. [6]

In een onderzoek van Alexandra Elbakyan, waar gebruik gemaakt wordt van Java, werden zowel voor REST als GraphQL drie verschillende experimenten uitgevoerd. In experiment 1 werd dezelfde data opgehaald in beide requests, komende van één resource. Dit neemt het aspect van de kleinere payload weg, zodat de snelheid van de GraphQL server vergeleken kan worden. In dit geval werd een java library gebruikt om de GraphQL server op te zetten. De resultaten tonen aan dat GraphQL hier op 1000 requests een kleinere responstijd vertoont. Er werd ook een gereduceerde GraphQL query opgemaakt waarbij het over-fetching fenomeen werd weggenomen. [24]

In experiment 2 werden meerdere gelinkte resources ingeladen. In GraphQL worden meerdere resources aangesproken in een request. In de REST-variant werden sequentieel drie verschillende resources opgehaald om dezelfde data terug te krijgen. In een derde variant werd een geoptimaliseerde REST-endpoint aangemaakt om specifiek deze resources in een response terug te geven. Ook in dit experiment was GraphQL met een klein verschil gemiddelde sneller dan de REST varianten.



Figuur 7 - Experiment 1: atomic resources aanvragen [24]



Figuur 8 - Experiment 2: aanvragen van meerdere gelinkte resources [24]

Dit is niet de enige factor die bepaalt of het gebruik van GraphQL sneller is dan REST. Hierbij speelt caching ook een rol. De front-end developer moet zelf zorgen voor caching. Maar op het vlak van opvraging van de technologie, zonder rekening te houden met welke server er wordt gebruikt, zorgt GraphQL in de meeste gevallen voor een snelheidswinst tegenover REST. Dit komt omdat over-fetching vermeden wordt en de response kleiner is. Ook wanneer de response dezelfde data terugstuurt, is GraphQL in het onderzoek van Alexandra Elbakyan sneller. Dit onderzoek alleen is niet voldoende om aan te tonen dat GraphQL sneller is, aangezien de snelheid ook afhangt van welke server gebruikt wordt.

Ontwikkelingstijden server

Op vlak van ontwikkeling van de server zijn de drie technologieën moeilijk te vergelijken. Dit hangt grotendeels af van welke server er wordt gebruikt. Om een voorbeeld te geven: Apollo Server is een van de meest gebruikte servers om met GraphQL te werken, maar is trager dan GraphQL-Jit. De reden dat Apollo-Server toch gebruikt wordt, is omdat deze extra functionaliteiten biedt bij het ontwikkelen en gebruiken van een GraphQL API.

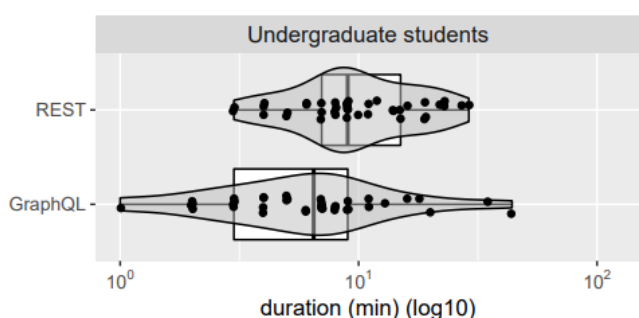
Dit blijkt ook uit het onderzoek van Ben Awad, die benchmarks heeft uitgevoerd op verschillende GraphQL servers, waarbij ook de snelste REST servers in een node omgeving vergeleken werden. Deze resultaten worden nog verder besproken in het technisch onderzoek. [25]

Het gebruik van een querytaal zorgt er wel voor dat de server beveiligd moet worden tegen kwaadaardige of slecht geoptimaliseerde queries. De mate waarin de server beschermd wordt, hangt af van de toepassing en zal in de meeste gevallen meer tijd kosten dan de beveiliging van een REST API, als gevolg van de vrijheid van de client bij het versturen van een request.

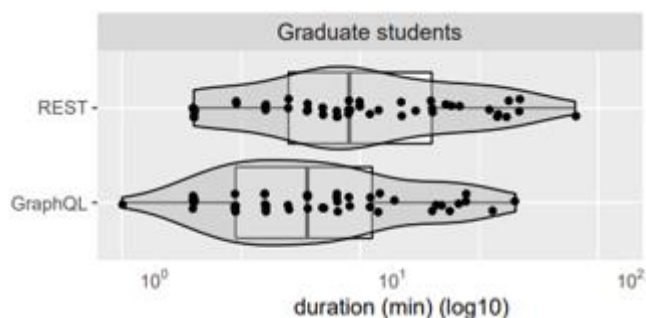
Ontwikkelingstijden implementatie op client

Wanneer we de client bekijken, kunnen we de snelheid van de implementatie van de API onderzoeken. De vrijheid van de client bij GraphQL heeft als gevolg dat er meer werk nodig is bij het opvragen van data, namelijk een query of mutation opstellen. Zo kan bepaald worden welke velden de response bevat. Bij REST hoeft slechts één endpoint aangesproken te worden zonder een selectie uit te voeren van de nodige velden. De GraphQL response zal, in tegenstelling tot de REST variant, altijd enkel de nodige data bevatten. Wanneer de toepassing verandert, kan een aanpassing in de query volstaan om over de nodige data te beschikken. De client is dus onafhankelijker bij gebruik van GraphQL.

Uit een studie van ASERG Group, Department of Computer Science bleek dat de implementatie van GraphQL in de meeste gevallen sneller ging dan die van REST. In deze studie werd een gecontroleerd experiment uitgevoerd bij 22 studenten (waarvan twaalf afgestudeerd), waarbij hen gevraagd werd om acht queries uit te voeren om een webservice te bevragen. De tijdswinst bij GraphQL neemt toe wanneer REST queries complexere endpoints gebruiken met verschillende parameters. Tegen de verwachtingen in overtreft GraphQL REST nog meer bij de afgestudeerde studenten en de deelnemers die ervaring hebben met REST.[26]



Figuur 9 - Duurtijd bij uitvoeren van alle opdrachten bij niet-afgestudeerden [26]



Figuur 10 - Duurtijd bij uitvoeren van alle opdrachten bij afgestudeerden [26]

Dit experiment werd uitgevoerd in IDLE, een IDE voor python, omdat IDLE geschikt is voor beginners, zeker in educatieve omgevingen. Het is ook belangrijk om te vermelden dat de deelnemers de mogelijkheid kregen om gebruik te maken van GraphiQL, ontwikkeld door GitHub. GraphiQL is een GUI om queries of mutations op te stellen, te bewerken en te testen. Binnen dit onderzoek is het gebruik

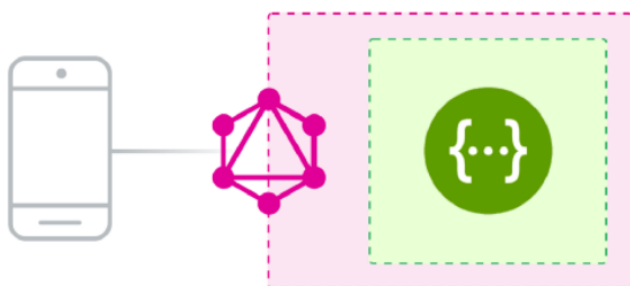
van GraphQL geen oneerlijk voordeel tegenover REST, aangezien dit een GUI is die bij GraphQL developers op dagelijkse basis gebruikt wordt om GraphQL queries op te bouwen.

De implementatie van REST of GraphQL verschilt naargelang van welke client er gebruikt wordt. In het technisch onderzoek van deze bachelorproef werd gebruik gemaakt van een Apollo Client “Vue Apollo”, een library die het gebruik van Apollo server vereenvoudigt in een Vue omgeving. [27]

2.8 In hoeverre kan een combinatie van API-technieken een webapplicatie performanter maken?

Een REST omgeving geeft een front-end developer niet de mogelijkheid om zelf te bepalen welke data meegestuurd wordt in de response. We kunnen de API wel uitbreiden, zodat de client GraphQL queries kan schrijven om enkel de nodige data terug te krijgen in de response. Zo kan GraphQL als gateway of wrapper functioneren om een bestaande REST omgeving toch de voordelen te bieden van GraphQL. [28]

De client bepaalt welke data die terugkrijgt, met als gevolg dat minder data verstuurd wordt in de response, er gebruik gemaakt wordt van één endpoint en er verschillende resources kunnen aangesproken worden in een request. Als er geopteerd wordt voor deze combinatie, valt de verantwoordelijkheid van client-side caching dan ook terug op de front-end developer.



Figuur 8 - GraphQL als wrapper [29]

Om GraphQL als wrapper te kunnen gebruiken, moet een schema gemaakt worden van de REST API. Dit kan handmatig gebeuren, aan de hand van SDL, wat het schrijven van schema's vereenvoudigt, met als nadeel dat het schema niet up-to-date blijft. Telkens wanneer de REST API verandert, moeten er manuele aanpassingen gebeuren.

Er kan ook gebruik gemaakt worden van een bestaand REST API schema zoals OpenAPI, om een GraphQL schema te genereren. Er zijn verschillende libraries die zorgen voor een automatisch gegenereerd GraphQL schema aan de hand van een Open Api Specification. Swagger-to-graphql converteert een bestaand swagger schema naar een executable GraphQL schema waar resolvers de REST endpoints gaan aanspreken. Op deze manier kan het gebruik van een GraphQL wrapper snel geïmplementeerd worden.

Wanneer GraphQL als wrapper dient, komt er een nieuwe kwetsbaarheid tevoorschijn. In een query kunnen er veel verschillende requests voorkomen naar de REST API. Er kan gemakkelijk een DoS-attack uitgevoerd worden. De request volume naar de REST API moet hierdoor beperkt worden. Dit is mogelijk via query complexity, depth limiting, persistent queries of andere maatregelen beschreven in hoofdstuk 2.5. Het GraphQL team is zich hiervan bewust en heeft een data loader library ontwikkeld die meehelpt om deze kwaadaardige queries te controleren. Deze library zorgt voor server-side batching & caching.

Wanneer de data loader dezelfde key (endpoint) meermaals in zijn lifetime ziet, geeft deze een gecachte, gememoriseerde versie van de response terug. [30] [31]

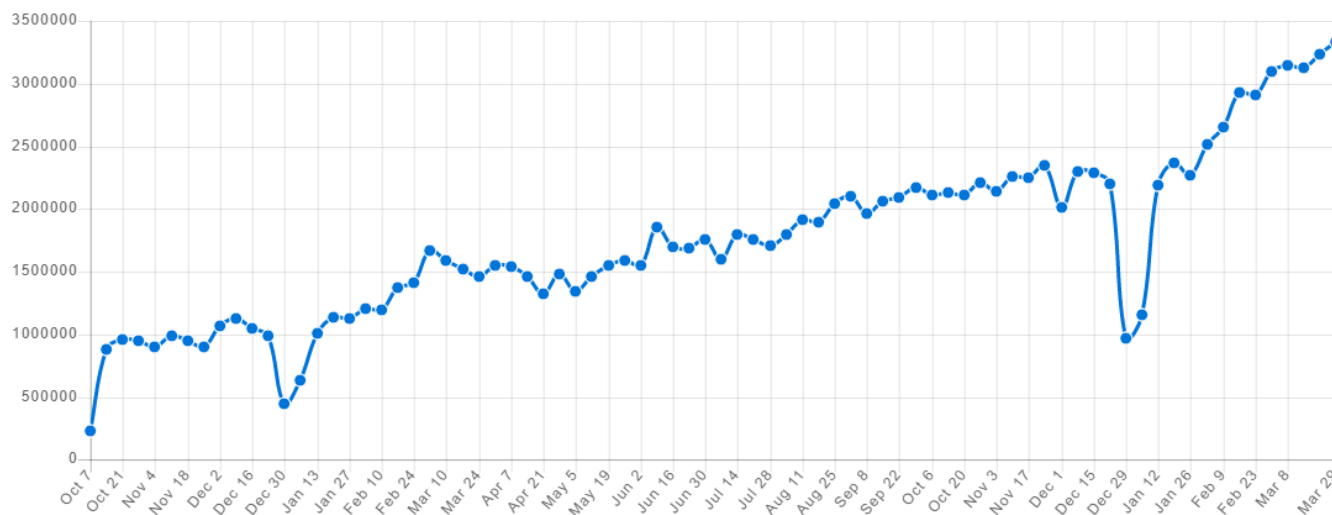
Nadesh Eeda, een student aan de universiteit van Western Ontario, heeft onderzocht of het gebruik van GraphQL als gateway de laadtijden van zijn real-time dashboard versnelt. Hieruit bleek dat GraphQL zorgde voor een snelheidswinst van 245% bij het inladen van de data. In dit onderzoek fungeerde GraphQL als gateway naar een REST API, een SOAP API en externe services. [6]

In een onderzoek van Watson Research Center werd een bestaande RESTful API omgevormd naar GraphQL aan de hand van de OASgraph library. De onderzoekers stelden vast dat APIs met een publiek beschikbare OAS gemakkelijk gewrapped kunnen worden door GraphQL, hoewel er vaak veel nodige informatie mist, waardoor er geen volledige automatische wrapping kan uitgevoerd worden. Deze tekortkomingen kunnen opgelost worden door de OAS aan te vullen of te corrigeren. [32]

2.9 Hoe is het gebruik van GraphQL verlopen sinds de initial release in 2015?

Facebook maakt gebruik van GraphQL voor zijn mobile apps en werd open source gesteld in 2015. Sindsdien maken naast facebook veel grote bedrijven gebruik van GraphQL, zoals: Airbnb, GitHub, PayPal, Twitter en nog veel meer. Over de jaren heen heeft GraphQL een grote community opgebouwd.

In 2018 verhuisde GraphQL van Facebook naar een nieuw gevestigde GraphQL Foundation, gehost door Linux Foundation. Het kan ondertussen gebruikt worden in veel verschillende omgevingen. In onderstaande grafiek zien we de wekelijkse downloads van de implementatie voor Javascript van 2019-2020, om een beeld te geven van de trend in de voorbije jaren. Sinds maart 2020 zijn er wekelijks meer dan 3 miljoen downloads van de javascript implementatie. [33]



Figuur 9 - Downloads van javascript implementatie GraphQL volgens npmtrends [33]

Ondertussen bestaat er voor bijna elke drawback van GraphQL een library, die met een kleine configuratie het probleem oplost. Zowel bijkomende beveiliging door het gebruik van een querytaal, als caching op de client en server, kunnen op deze manier snel worden opgelost. De community blijft groeien en het is mede hierdoor dat GraphQL langzamerhand, meer en meer gebruikt wordt.

2.10 Hoe zal het gebruik van GraphQL evolueren naar de toekomst toe?

Uit de grafieken van npmtrends is duidelijk dat GraphQL zal blijven groeien. Van januari tot maart zijn de wekelijkse downloads van de javascript implementatie met 30% gestegen. REST heeft zijn voordelen, maar de voordelen van GraphQL zijn niet te ontkennen, en meer en meer zal geopteerd worden voor GraphQL. Zeker omdat de grootste nadelen van GraphQL (geen build-in caching support en vulnerabilities door gebruik van queries) tegenwoordig snel kunnen opgelost worden door gebruik te maken van libraries die ontwikkeld worden door zowel GraphQL Foundation als de GraphQL community.

REST zal nog lang gebruikt worden, de architectuur zorgt voor uniformiteit bij het ontwikkelen van web services in verschillende projecten. De voordelen die de stateless client-server architectuur met zich meebracht, blijven belangrijk en relevant.

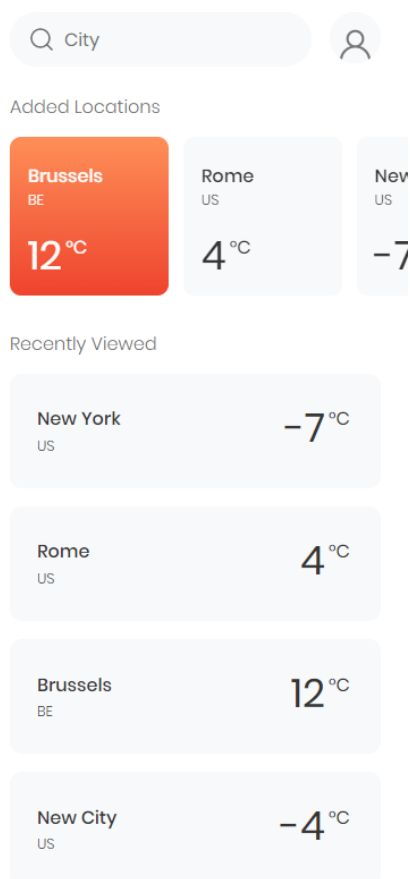
Ook het gebruik van GraphQL als gateway brengt al heel wat voordelen met zich mee en kan soms eenvoudig te implementeren zijn, waardoor een van de grootste drawbacks van REST, met name 'over-fetching' opgelost wordt. De beveiliging van de gateway om een denial of service te voorkomen en aan server side caching en caching te doen, kan snel opgezet worden door facebook's dataloader library, in combinatie met andere libraries die de kwetsbaarheden bij het gebruik van GraphQL als querytaal oplossen, zoals beschreven in hoofdstuk 2.5.

3 Technisch onderzoek

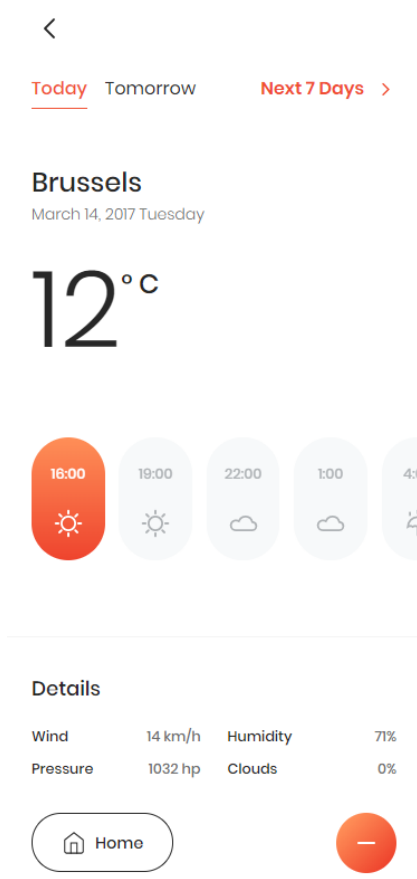
3.1 Werkwijze

Er werd een vergelijkende studie gemaakt, van twee gelijke webapplicaties. De ene maakt gebruik van GraphQL en de andere van REST. Beide webapplicaties geven af de huidige weersomstandigheden en weersvoorspellingen van 22 000 steden weer. Zowel de ontwikkelingstijden als de query- en responstijden werden onderzocht. Er werd voor de client en server bekeken welke stappen ondernomen moesten worden voor beide varianten.

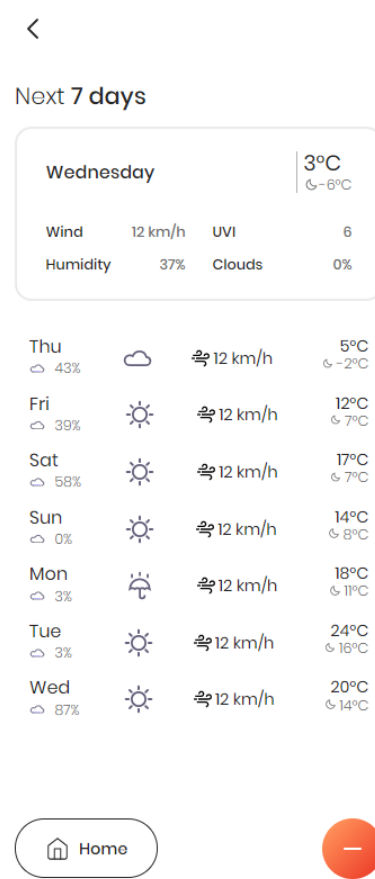
Aangezien het project ook de query- en responstijden onderzocht, was het belangrijk om over een grote dataset te beschikken, zodat zwaardere queries konden worden uitgevoerd om het verschil in responstijd te kunnen bewijzen. OpenWeather biedt de mogelijkheid om de toestand op 14 maart 2017 te downloaden in bulk als JSON-bestanden. Er zijn dus drie verschillende JSON-bestanden, waarbij respectievelijk de huidige weersomstandigheden, de komende 36 uren en de komende veertien dagen weergegeven worden. [34]



Figuur 10 - Technische demo: schermafbeelding 1



Figuur 11 - Technische demo: schermafbelding 2



Figuur 12 - Technische demo: schermafbeelding 3

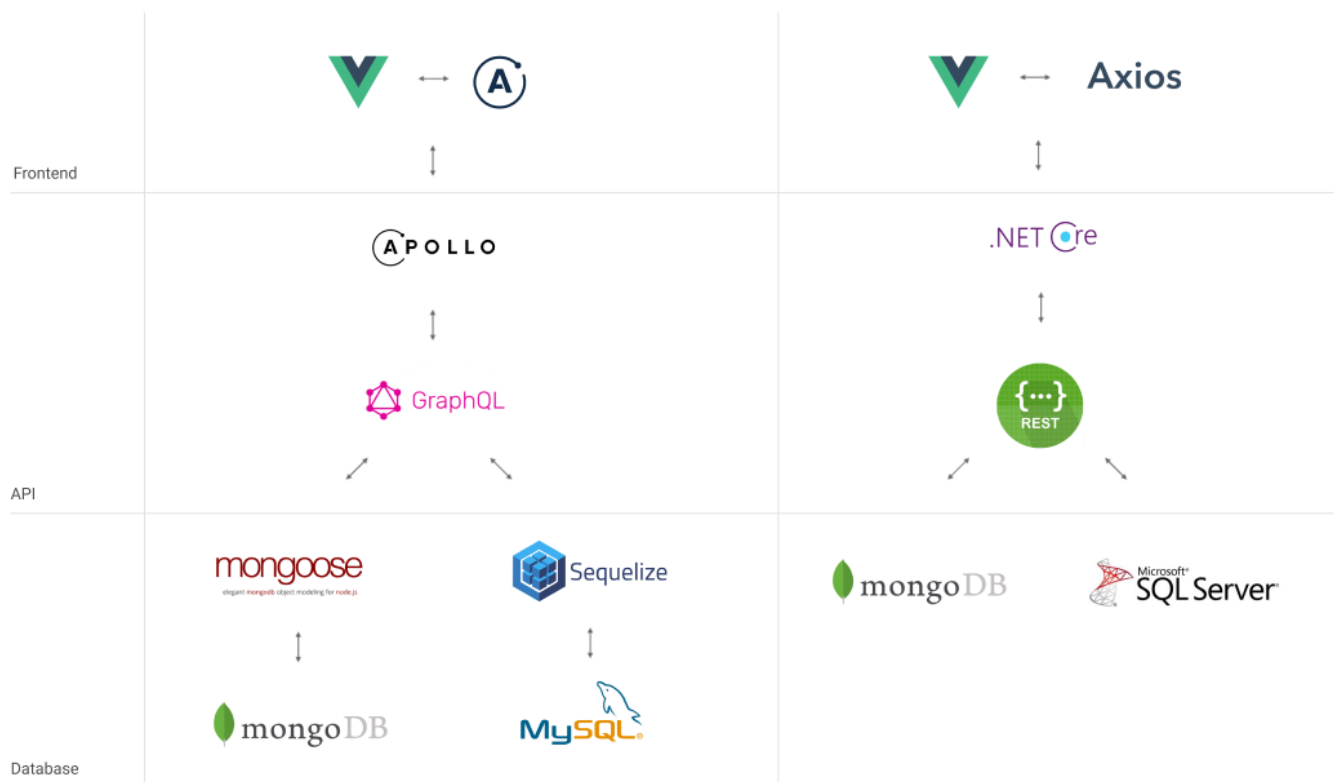
Daarnaast werd autorisatie ook bekeken. Om de gebruikers op te slaan, werd gebruik gemaakt van een SQL-database. Zo kan de misconceptie dat GraphQL eerder zijn nut bewijst in een NoSQL-database omwille van zijn geneste object structuur weerlegd worden.

Doordat er nog geen ervaring was met het gebruik van GraphQL als querytaal, kunnen de ontwikkelingstijden in deze technische demo niet aantonen of er een snelheidswinst is tegenover een REST API. De opbouw van de server en de implementatie van de API worden wel besproken en op basis

hiervan wordt een prognose gemaakt of het voordelig is om in een bedrijfscontext te migreren naar GraphQL wat snelheid van de opbouw en de implementatie betreft.

3.2 Opstelling

De webapplicaties maakten beide gebruik van Vue.js als front-end om de APIs te bevragen. De snelheid van zowel de ontwikkeling als de query en de responstijden ervan werden onderzocht. Er werd gebruik gemaakt van een NoSQL-database en een SQL-database met verschillende CRUD-acties. De weersomstandigheden werden in de NoSQL-database opgeslagen en de gebruikers in de SQL-database.



Figuur 13 - Schema opstelling technische onderzoek

Opstelling GraphQL

De keuze voor ApolloServer werd gebaseerd op zijn populariteit en de mogelijkheden ervan. Het Apollo platform is een industry standard voor GraphQL implementaties. De GraphQL queries moesten ook gemonitord kunnen worden om te kijken hoe lang deze duren en wat de oorzaak hiervan was. Apollo biedt cloud services aan die dit mogelijk maken. Het platform is open-source en heeft sindsdien een grote community opgebouwd, die ook nu nog blijft groeien. Het dankt zijn populariteit deels aan zijn interoperabiliteit voor verschillende frameworks zoals Vue of React. In het technisch onderzoek werd client-side gebruik gemaakt van Apollo-vue, om de server aan te spreken.

Om de NoSQL-database aan te spreken, werd gebruik gemaakt van Mongoose als ORM. Dit is een van de ORM's die de meeste functionaliteiten biedt en gemakkelijk geïmplementeerd kan worden in een javascript omgeving. Mongoose voorziet een straight-forward, schema gebaseerde oplossing om data naar een model over te brengen. En het beschikt over verschillende built-in functionaliteiten zoals type casting, validation en query building. [35]

Om de SQL-database aan te spreken werd gebruik gemaakt van Sequelize als ORM. Deze biedt ondersteuning voor zowel MsSQL, als MySQL, MariaDB, Postgres, ... Sequelize staat al jaren aan de top in Node.js omgevingen en biedt de mogelijkheid om migrations uit te voeren, waarvan gebruik gemaakt wordt om de entiteiten op te stellen.

Als database-server werd gekozen voor MySQL voor de SQL-database en MongoDB als NoSql-database. Deze worden het meest gebruikt volgens de statistieken van DB-Engine ranking op 26 juli 2019, waarbij MySQL aan de top staat als Relational Database-Management System. [36] [37]

Opstelling REST

Er werd gebruik gemaakt van EntityFramework Core. Dit is een framework gebaseerd op een ORM, die zorgt voor het mappen van de databasetabel naar het model in de webapplicatie. Er werd gebruik gemaakt van Microsofts Identity Service om authenticatie af te handelen, waarbij de usertabellen automatisch worden aangemaakt en dus ook van Microsoft SQL-server, om de IdentityService van het Entityframework te kunnen benutten. Ondanks het verschil in SQL-database, werd gebruik gemaakt van dezelfde en meest voorkomende NoSql-database server MongoDB.

Om de data afkomstig van de REST api weer te geven en te consumeren, werd gebruik gemaakt van axios, een promise based http-client voor de browser en node.js.

3.3 Ontwikkeling API

Een deel van het project onderzoekt hoe de ontwikkeling van GraphQL verloopt en hoelang deze duurt in verhouding met die van een REST Api. Aangezien de technische demo hielp bij het leerproces van GraphQL, was het vanzelfsprekend dat de ontwikkeling van GraphQL langer zou duren, omdat hier nog geen voorkennis van was. Hiervoor werd in de planning meer tijd voorzien.

REST server

Een REST service werd aangemaakt door endpoints in de controllers te definiëren. Elke action kreeg zijn endpoint. Hierin werden in een MVC-structuur repositories opgeroepen, die data-access afhandelen. Dit kan in andere patterns onder een andere naam voorkomen.

Er werden modellen of aggregates aangemaakt. Dit is het volledige object waarnaar de database entiteit wordt gemapt. Wanneer de response velden bevatte die niet meegestuurd moesten worden in de response, werd gebruik gemaakt van een DTO of ResponseModel.

Om de user tabellen aan te maken, werd de IdentityService van het Entity Framework gebruikt. Dit zorgde voor een correcte implementatie, waarbij veel werk gespaard werd. Er hoefden slechts enkele properties toegevoegd te worden aan het user model en de migratie zorgde voor de generatie van de databasetabellen.

GraphQL server

Een GraphQL Service werd aangemaakt door types en fields te definiëren en hiervoor functies te schrijven. De query die afgevuurd werd door de client bevat altijd fields die teruggestuurd worden in de response en bevat optioneel arguments, aliases en fragments. Hiervoor verwijs ik naar de GraphQL documentatie.

Er werden, net zoals bij REST, modellen voorzien om aan mapping te kunnen doen. Hierbij wordt gebruik gemaakt van de ORM's Sequelize en Mongoose.

Om de gebruiker te laten zien welke velden hij kon opvragen werden schema's voorzien. Hierin staan de verschillende types. Aangezien GraphQL services in elke programmeertaal geschreven kunnen

worden, wordt gebruik gemaakt van “GraphQL Schema Language”. Hierbinnen is een “object type” een object dat aangevraagd kan worden op de service.

In de resolver gebeurt de data-access en wordt het model teruggegeven met enkel de opgevraagde velden in dezelfde structuur als de query. Het is ook op dit niveau, de business logic layer, dat authenticatie en autorisatie plaatsvindt. Via de context kan de client zichzelf identificeren en kan op de server bepaald worden of de gebruiker rechten heeft om de resource op te halen of te bewerken.

Hoelang het duurt om de API op te stellen, hangt af van de server en van de functionaliteiten die de applicatie bevat. In mijn geval duurde het opstellen van beide webapplicaties even lang, maar er werd nog geen rekening gehouden met de vulnerabilities die GraphQL met zich meebrengt. Doordat GraphQL de client de mogelijkheid geeft om zelf te bepalen welke data de response bevat, moeten we ervoor zorgen dat de client geen kwaadaardige queries kan versturen naar de server. Hiervoor moeten er op de server enkele security layers toegevoegd worden om een DoS te voorkomen.

3.4 Implementatie API

Implementatie REST

Data ophalen van een REST API is geen enkele developer onbekend. Dit kan op verschillende manieren gebeuren, maar binnen mijn Vue front-end heb ik gekozen om gebruik te maken van Axios.

Hiervoor moet een base-url worden opgesteld, waarna verschillende endpoints kunnen aangesproken worden door een `axios.get` uit te voeren. Bij het aanmaken van een axios instantie, kan door toevoeging van een autorisatie header, waarin de bearer token opgehaald wordt uit local storage, aan authenticatie gedaan worden. Zo maakt de client zich bekend en kan er op de server bepaald worden of deze geautoriseerd is om acties uit te voeren.

```
const createApi = () => {
  api = axios.create({
    baseURL: process.env.VUE_APP_API_URL,
    timeout: 360000,
    headers: {
      Authorization: `Bearer ${localStorage.getItem('token')}`,
    },
  });
};
```

Om een request te doen naar de server is een get action, met enkel de endpoint voldoende. Door de token in local storage op te slaan, wordt deze bij elke request meegestuurd in de autorisatie header. Op dezelfde wijze kan een post, update of delete toegepast worden. Hierbij worden de parameters meegegeven als tweede argument.

```
api.get(`/current/${this.city}`)
  .then((result) => {
    this.weather = result.data;
  });
```

In dit geval wordt het weather object gelijkgesteld aan de data afkomstig van de API, waar onnodig veel informatie inzit, aangezien niet alle velden daaruit gebruikt worden binnen de webapplicatie. De implementatie bevat niet veel code.

Wanneer we op een pagina verschillende resources nodig hebben, zoals op de detailpagina van een stad, waarbij we zowel de huidige weersomstandigheden nodig hebben als die van de komende uren, worden twee endpoints aangesproken. Er moeten twee connecties worden gelegd.

Implementatie GraphQL

De implementatie van GraphQL kan op verschillende manieren gebeuren. Dit hangt af van de gebruikte server. Wanneer Apollo Server gebruikt wordt, zoals in dit project, zal de Apollo Client het gebruik van de API vergemakkelijken. Een request kan in Apollo als volgt uitgevoerd worden, waarbij enkel de naam van de methode, de naam van de query en de variabelen als argumenten worden meegegeven.

```
apollo: {
  current: {
    prefetch: true,
    query: currentsbycity,
    variables() {
      return { cityname: this.city };
    },
  },
},
```

De queries en mutations kunnen per bestand worden opgeslagen met extensie '.gql', dit zorgt voor centralisering en helpt bij het bewaren van een overzicht. Per query bepaalt de gebruiker welke velden er nodig zijn om mee te sturen in de response. Het motto van GraphQL "What you need is what you get" bewijst hier zijn nut. Authenticatie zal automatisch gebeuren in vue-apollo.js. In de context wordt de token meegestuurd vanuit local storage. Standaard staat de key hiervan ingesteld op 'apollo-token', maar deze kan worden aangepast.

```
query CurrentByCity($cityname: String!){
  current(cityname: $cityname){
    city{
      name
      country
    }
    main {
      temp
      pressure
      humidity
    }
  }
}
```

Wanneer we bij GraphQL data nodig hebben uit verschillende resources kan dit, mits een aangepaste query door de client, wel in één request gebeuren.

De implementatie van de API hangt af van de complexiteit van de toepassing. Het grote voordeel bij GraphQL is dat de gebruiker de vrijheid krijgt om te bepalen welke data hij terugkrijgt. De implementatie in dit project verliep ongeveer even snel bij REST als bij GraphQL, maar dit is niet voldoende om een antwoord te geven op de vraag welke van de twee sneller is op vlak van implementatie bij de client. Dit hangt te veel af van de functionaliteiten binnen de applicatie en van de server die gebruikt wordt. Een kleinschalige test, op meerdere personen geeft hier een goed beeld van. Hiervoor verwijst ik naar het onderzoek van ASERG Group, Department of Computer Science. Uit dit experiment bleek de implementatie van GraphQL sneller te verlopen dan deze van REST, bij zowel de afgestudeerde als niet-afgestudeerde studenten. Het experiment wordt beschreven in hoofdstuk 2.7. [26]

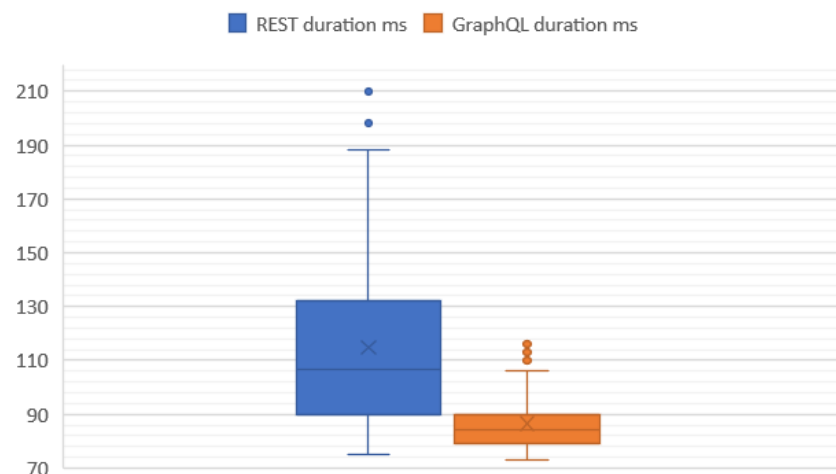
3.5 Responstijden

Wanneer de keuze maakt wordt om over te stappen naar een nieuwe technologie, is de snelheid ervan een grote factor. Het kan zijn dat GraphQL voordelen biedt voor de front-end developers, maar dit wil niet zeggen dat de webapplicatie performanter wordt.

De grootte van de response werd al besproken in het vorige hoofdstuk. Het gebruik van GraphQL resulteert in een kleinere hoeveelheid data die over het netwerk verstuurd wordt. In de technische demo wordt dit nogmaals duidelijk, wanneer we de totale hoeveelheid data bekijken om het weekoverzicht van dezelfde stad te bekijken:

<input type="checkbox"/> Rome	200	xhr	VM1003:1	5.0 KB	69 ms
<input type="checkbox"/> graphql	200	fetch	VM2056:1	4.1 KB	33 ms

Nu rest ons nog de vraag of GraphQL in dit project sneller is, wanneer dezelfde hoeveelheid data verstuurd wordt. Om dit te onderzoeken werd gebruik gemaakt van de postman API-client. De request van cities bevat dezelfde data, enkel de namen van de stad. De responsgrootte bedraagt 250KB bij zowel REST als GraphQL. In de volgende boxplot worden 100 metingen gevisualiseerd. Hier kunnen we zien dat de GraphQL variant beter scoort. Zowel minimum, maximum, gemiddelde, mediaan en uitschieters scoren hier beter. Dit is te danken aan Apollo Server die in dit geval sneller is dan .NET Core. Dit kan te wijten zijn aan een grotere hoeveelheid aan middleware. Er is dus onderzoek nodig naar welke factoren een invloed hebben op de snelheid van de server.



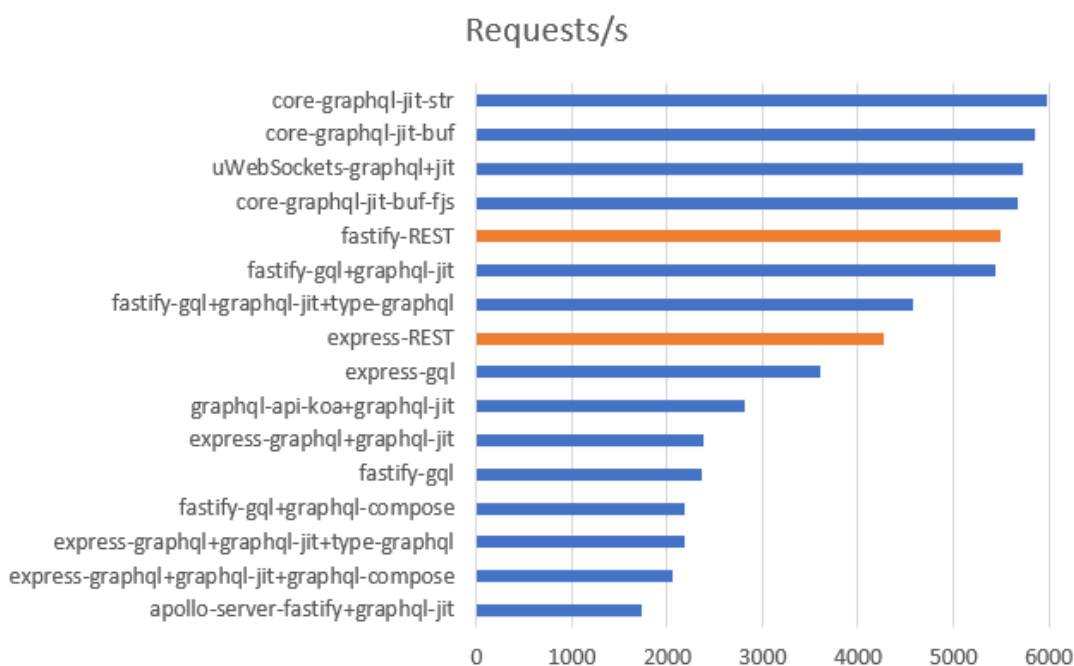
Figuur 14 - Meetresultaten responstijden GraphQL - REST

Deze resultaten komen ook overeen met die van GRIAL Research Group. In dit onderzoek werd gebruik gemaakt van Graphene, een GraphQL framework voor python. Hier was een snelheidswinst van 37% op vlak van network loading times. [38]

Dit was ook het geval in het onderzoek van Naresh Eada, die als GraphQL server gebruik maakte van KOA en bij REST gebruik maakte van Restify (beide node servers). In dit onderzoek fungeerde GraphQL als wrapper en werden de resultaten gebaseerd op de loading time van een Dashboard, waarbij GraphQL een snelheidswinst vertoonde van 245%. [6]

Hoe we de snelheden nog meer kunnen optimaliseren is door onderzoek te doen naar welke GraphQL server sneller is. Hiervoor verwijst ik naar benchmarks van Ben Awad, een GraphQL enthousiast, waarin verschillende GraphQL node servers met elkaar vergeleken worden.

Hierin worden ook de snelste REST servers onderverdeeld. Daaruit bleek Apollo een van de traagste servers te zijn op vlak van zowel requests per seconde, als latency en throughput. Dit kan te wijten zijn aan de grote hoeveelheid aan extra functionaliteiten, met als voordeel dat zowel de opbouw als implementatie van de API een relatief gemakkelijk proces wordt. Wanneer er geen gebruik gemaakt wordt van extra functionaliteiten, raadt hij aan om over te stappen naar core-graphql-jit. De snelste server op basis van zijn benchmarks. Hieronder staat een visualisering van de requests per seconde voor de verschillende servers. [25]



Figuur 15 - Overzicht snelheden javascript servers GraphQL – REST [25]

4 Reflectie

4.1 Resultaten technisch onderzoek

Het doel van project 4 was om te onderzoeken of het al dan niet rendabel is om van een bestaande REST omgeving over te schakelen naar het gebruik van GraphQL als querytaal. Uit onderzoek vooraf bleek GraphQL veel voordelen te bieden op verschillende vlakken, die ook aangetoond worden in de technische demo. De bevindingen van dit technisch onderzoek worden vergeleken met de ervaringen van verschillende externen die GraphQL gebruiken in de praktijk.

Een GraphQL service opbouwen, gebeurt aan de hand van een schema dat de object types omvat die de client kan opvragen of bewerken. De complexiteit van de webapplicatie, samen met de patterns die gebruikt worden, bepalen hoe GraphQL geïmplementeerd wordt. In project 4, een kleinschalige webapplicatie die de weersvoorspellingen weergeeft, was de ontwikkelingstijd van de service - ondanks de onbekendheid met GraphQL als querytaal - niet langer dan de ontwikkelingstijd van de REST API. Het is niet onbelangrijk om aan te halen dat het gebruik van GraphQL, doordat dit een querytaal is, verschillende kwetsbaarheden met zich meebrengt. Dit werd echter niet in dit project behandeld. Er is wel onderzoek gevoerd rond de implementatie van veiligheidsmaatregelen bij het gebruik van GraphQL als querytaal, en deze moeten ook in rekening worden gebracht wanneer we de ontwikkeling van een GraphQL service vergelijken met een REST variant.

Bart Wullems is een back-end developer en intussen ook businessunit manager bij Ordina. Hier wordt tegenwoordig gebruik gemaakt van GraphQL. Het is dus interessant om de bevindingen van dit onderzoek te vergelijken met zijn ervaringen in de praktijk. Bij Ordina beveiligd men de service door veiligheidsmaatregelen toe te passen zoals omschreven in hoofdstuk 2.5. Het beveiligen vergt volgens Bart zonder enige twijfel meer inspanning dan bij een REST API.

Doordat het gebruik van GraphQL als querytaal nog moest aangeleerd worden tijdens de opbouw van het project in dit onderzoek, kunnen de ontwikkelingstijden in de technische demo niet voldoende aantonen of er een snelheidswinst is tegenover een REST API. De opbouw van de server en de implementatie van de API worden wel besproken, en op basis hiervan wordt een prognose gemaakt of het voordelig is om in een bedrijfscontext om te migreren naar GraphQL op vlak van de snelheid van de opbouw en de implementatie. De snelheid van de implementatie van GraphQL en REST bij de client werd onderzocht door ASERG Group, department of computer science. Uit dit experiment met 22 deelnemers blijkt de snelheid van de implementatie van GraphQL hoger te zijn dan bij de REST variant. Dit experiment wordt toegelicht in hoofdstuk 2.7.

Bart heeft een tweeledig antwoord, wanneer we praten over de implementatie van GraphQL tegenover die van een REST API. De realiteit is dat de meeste mensen waarmee hij samenwerkt al iets van REST kennen, maar geen ervaring hebben in GraphQL. Om een complexere GraphQL service goed te ontwerpen, komt er dus wel wat denkwerk aan te pas. Eenmaal die eerste hinderpaal overwonnen is, is het effectief implementeren ervan even snel langs de back-end kant. Langs de front-end kant ziet hij typisch een versnelling en is het veel eenvoudiger om de API te gebruiken en te integreren in een applicatie. Zijn voorkeur als consumer gaat steeds uit naar een GraphQL API, doordat deze flexibeler zijn, beter om kunnen met verschillende vereisten en typisch sneller geïmplementeerd geraken.

Wanneer we de responstijden vergelijken tussen REST en GraphQL, wordt duidelijk aangetoond dat er een snelheidswinst is bij het gebruik van GraphQL. Dit is een logisch gevolg van een kleinere hoeveelheid data die verstuurd wordt, doordat de client bepaalt welke data de response zal bevatten. Bij de REST variant wordt door de server bepaalt welke data teruggestuurd wordt in de response. Dit verschijnsel, waarbij te veel data verstuurd wordt over het netwerk, noemt men over-fetching en wordt dus vermeden bij het implementeren van GraphQL.

Indien we louter de snelheid bekijken, wanneer de responsgrootte van beide responses even groot is, moeten we verschillende factoren in rekening brengen. De keuze van de server is een belangrijke factor. Uit project 4 blijkt de Apollo Server die gebruik maakt van GraphQL, snellere responstijden te vertonen dan de REST API in .Net Core. Er is onderzoek nodig naar welke factoren een invloed hebben op de snelheid van de server.

Bij Ordina werden uiteenlopende resultaten vastgesteld. Het al dan niet ondersteunen van HTTP2 maakt een groot verschil. Wanneer je een HTTP1 service samenneemt met een GraphQL API, is de tweede systematisch sneller dankzij onder andere de batching mogelijkheid en het vermijden van over-fetching. Bij HTTP2 zijn de verschillen kleiner en meestal verwaarloosbaar. Volgens Bart zit de echte snelheidswinst vooral in het gebruiksgemak als consumer.

Toen de webapplicatie in dit onderzoek over al zijn functionaliteiten beschikte, werd besloten om autorisatie te implementeren, aangezien dit een cruciaal element is bij het bouwen van WebAPIs. De implementatie in GraphQL kon worden toegepast door gebruik te maken van de query context. In dit onderzoek werd ook duidelijk dat het minder een kwestie is van autorisatie bij GraphQL, maar meer de manier waarop autorisatie geïmplementeerd wordt.

4.2 Bruikbaarheid projectresultaat

De uitvoering van project 4 bewijst verschillende voordelen, wanneer er van een REST omgeving overgeschakeld wordt naar het gebruik van GraphQL, hoewel er elementen ontbreken die zorgen voor implementatiehindernissen in een bedrijfscontext. Door de vrijheid die de client krijgt bij het bepalen welke data teruggestuurd wordt in de response, kunnen kwaadaardige queries leiden tot een denial of service. Er zijn verschillende maatregelen die de toepassing hiertegen kunnen beschermen. Deze staan opgelijst in hoofdstuk 2.5. Verschillende libraries kunnen hier hulp bij bieden.

Deze beveiliging neemt meer tijd in beslag dan die van een REST API, maar moet toegepast worden alvorens de service publiek toegankelijk wordt gesteld. Als deze beveiliging correct wordt geïmplementeerd, biedt GraphQL verschillende voordelen die aangetoond werden in dit project. Dit wordt tevens aangehaald door Bart Wullems, back-end developer bij Ordina.

Er moet ook goed overwogen worden welke server geschikt is voor de toepassing die gebouwd wordt. Apollo Server biedt bijvoorbeeld extra functionaliteiten die het bouwen van de toepassing kunnen vereenvoudigen, maar zorgt tegelijkertijd voor veel extra overhead, waardoor dit een van de tragere servers is in een javascript omgeving.

4.3 Overweging implementatie in bedrijfscontext

Implementatie GraphQL

Om te weten of de implementatie van GraphQL in een bedrijfscontext rendabel is, moet er rekening gehouden worden met verschillende aspecten. We bekijken de mogelijkheden, waarbij de meerwaarde en de hindernissen afgewogen worden tegen elkaar.

Het gebruik van GraphQL zorgt voor vrijheid en eenvoud bij de client. De response bevat enkel nodige informatie en alle requests kunnen verstuurd worden naar één endpoint. Wanneer een aanpassing gebeurt aan de front-end kant van een webapplicatie, moet er dus niets veranderd worden aan de kant van de server. Front-end en back-end teams kunnen onafhankelijk van elkaar werken, door het gebruik van een schema.

Ook bij Ordina ervaren ze een snelheidswinst door het gebruiksgemak als gebruiker van de service. Bart haalt aan dat GraphQL een flexibiliteit biedt die vooral zijn waarde toont in veranderende situaties, die typisch voorkomen op plaatsen waar de eindgebruiker van het systeem een rol speelt. Dit antwoord

komt overeen met hoe de implementatie van GraphQL in dit onderzoek ervaren werd, waarbij het front-end project minder afhankelijk was van de GraphQL service.

Er moet bijkomend wel gezorgd worden voor een beveiliging tegen kwaadaardige of slecht geoptimaliseerde queries. Bij Ordina vertelt Bart dat de beveiliging van de service zonder twijfel meer tijd in beslag neemt dan die van een REST API. Dit antwoord stemt ook overeen met wat Ben Awad vermeldde.

Ben is een GraphQL enthousiast die eerder in dit onderzoek al vermeld werd omwille van zijn onderzoek naar welke server performanter is in een node omgeving. Zijn ervaringen zijn dus zeker een meerwaarde, aangezien hij een expert is in het gebruik van GraphQL. Hij ervaart GraphQL als een stap vooruit, hoewel er ook nadelen zijn die overeenstemmen met de ondervindingen in dit onderzoek. Hij vermeldt dat GraphQL hem veel geholpen heeft, bij de implementatie in een front-end webapplicatie. Er moet echter aandacht besteed worden aan de extra stappen die ondernomen moeten worden bij het bouwen van de service. Het beveiligen van de service neemt meer tijd in beslag dan het beveiligen van een REST API.

GraphQL kan met de juiste implementatie resulteren in snelle responstijden dan die van een REST API. Nu rest ons nog de vraag wanneer het rendabel is om een bestaand project om te vormen tot GraphQL en welke mogelijkheden er zijn. In een bedrijf kan een bestaand project niet zomaar omgevormd worden naar GraphQL, omdat daar niet altijd budget voor is. Indien de snelheid een bepalende factor is om te kiezen voor een nieuwe technologie, pakt GraphQL uit door de eliminatie van over-fetching in combinatie met server-side batching. Ook Bart haalt aan dat dit in snellere responstijden resulteert.

Combinatie API-technieken

Door het gebruik van GraphQL als wrapper, kan de REST API performanter worden. Op deze manier wordt over- en under-fetching vermeden, omdat de client de vrijheid krijgt om te bepalen welke data de response zal bevatten. De nodige beveiliging moet geïmplementeerd worden om te zorgen dat het gebruik van GraphQL de onderliggende services niet te zwaar belast.

Omdat het interessant is om ook ervaringen in de praktijk van GraphQL als wrapper te betrekken in dit onderzoek, nam ik contact op met Ben Elsen, een back-end developer bij Sentia. Ben gebruikt GraphQL als wrapper voor verschillende REST services. Hij ervaart bij de beveiliging van GraphQL weinig verschil in het beschermen van de endpoint, en dit om twee redenen. Enerzijds zijn er de cloud providers die een belangrijk stuk van de bescherming op zich nemen. Hierbij verwijst hij naar authenticatie door middel van integraties met user management systems en throttling solutions. Anderzijds zijn er een aantal best practices die niet veel verschillen van REST, zoals parameters checken op injection. Doordat Sentia gebruik maakt van AWS als cloud provider, zal deze ook zorgen voor throttling en beveiliging. Wanneer een dusdanig grote query te veel resources vraagt zal de API niet onderuitgaan, maar wordt er door AWS zelf een error gestuurd naar de client.

4.4 Alternatieven community

Er zijn tot op heden nog heel wat voorstanders van REST die GraphQL kritisch bekijken. Dit is normaal wanneer een nieuwe technologie zijn intrede maakt naast vertrouwde technologieën. Wanneer we de voorwaarden van een REST API bekijken, worden de voordelen snel duidelijk en zien we een gestructureerde architectuur. GraphQL voldoet slechts aan twee van de REST constraints. Door het gebruik van één endpoint, worden een deel van deze voorwaarden achterwege gelaten. Dit is een logisch gevolg van de vrijheid die GraphQL met zich meebrengt. Het is ook normaal dat ontwikkelaars zich vragen stellen bij de kwetsbaarheden van GraphQL, die kunnen zorgen voor een Denial of Service. De querytaal maakt het mogelijk om kwaadaardige queries op te stellen die de server te zwaar belasten. Dit is een kwetsbaarheid die bij REST niet voorkomt. De server bepaalt immers zelf wat de endpoint

teruggeeft. Er is een bijkomende beveiliging nodig bij de migratie van REST naar GraphQL, ten gevolge van de vrijheid van de client.

4.5 Economisch en maatschappelijke meerwaarde van GraphQL

In een bedrijfscontext is het belangrijk, wanneer er geopteerd wordt voor een nieuwe technologie, dat deze niet alleen kwalitatief winstgevend is. Als de service door gebruik van deze technologie op het vlak van snelheid en bruikbaarheid voordelen biedt, wil dit niet zeggen dat deze toepasbaar is op bestaande of nieuwe projecten. Er moet gekeken worden of de overgang naar deze technologie vlot te implementeren valt. Een leerproces zal sowieso tijd vragen en er moet overwogen worden of deze tijd op termijn rendabel is.

Ook bij Ordina ervaart men deze leercurve, waarbij de meeste developers met wie Bart in contact staat, geen ervaring hadden in het bouwen van GraphQL services. Hij vertelt wel dat het implementeren langs de back-end kant uiteindelijk even snel verloopt als bij REST. Langs de front-end kant ziet hij typisch een versnelling, waarbij het eenvoudiger is om de API te implementeren in een applicatie. Deze vaststelling komt overeen met de resultaten uit het onderzoek van ASERG Group waarbij een experiment aantoonde dat de implementatie van GraphQL gemiddeld sneller verloopt dan die van een REST API. Bij het bouwen van de technische demo, verliep de implementatie van GraphQL even snel als die van de REST API, zonder voorkennis van GraphQL.

Wanneer de toepassing verandert, kan een aanpassing in de query volstaan om over de nodige data te beschikken. De client is dus onafhankelijker bij het gebruik van GraphQL.

Aangezien grootschalige projecten niet herschreven kunnen worden, kunnen deze worden uitgebreid met GraphQL door het gebruik van GraphQL als gateway. Dit wordt bij Ordina ook toegepast, aangezien ze niet alle bestaande services wensten te herschrijven. Alle REST services werden achter een federated GraphQL schema geplaatst. Nieuwe projecten kunnen door het gebruik van GraphQL, zorgen voor op lange termijn beter onderhoudbare services. Dit is te danken aan het gebruik van de schema's en de single endpoint.

4.6 Suggestie vervolgonderzoek

In dit project werd Apollo Server gebruikt om de GraphQL service op te bouwen. Dit is een toegankelijke server met veel functionaliteiten, maar ook een van de tragere in een node omgeving. In het onderzoek van Ben Awad worden verschillende GraphQL servers in een node omgeving met elkaar vergeleken, waaruit blijkt dat de keuze van de server een significant verschil maakt in de duur van de responstijden. Het is duidelijk dat er veel snellere servers zijn dan Apollo Server, waarbij minder overhead zorgt voor snellere responstijden.

Dit onderzoek brengt veel duidelijkheid, wanneer overwogen wordt om van een REST omgeving te migreren naar GraphQL in een Node omgeving. Zowel de GraphQL servers als de snelste REST servers worden met elkaar vergeleken. Het zou dus interessant zijn om ook voor andere omgevingen zoals Python, Java en .NET Core een gelijkaardig onderzoek uit te voeren, waarbij benchmarks worden uitgevoerd op de verschillende de servers. Hiervoor hoeft geen webapplicatie uitgebouwd te worden, het volstaat om alleen de server aan eenzelfde reeks proeven te onderwerpen. Het kan ook interessant zijn om hierbij te onderzoeken welke factoren de snelheid van de server bepalen.

5 Advies

Wanneer een webAPI ontwikkeld wordt, grijpt men vaak terug naar de REST architectuur. Een vertrouwde manier van werken die al jarenlang de standaard is bij het bouwen van een webAPI. Het is belangrijk dat de server zich kan aanpassen aan variërende behoeften bij de client. De implementatie van een GraphQL service bij de client zorgt voor efficiëntie en flexibiliteit, waardoor de back-end zich minder moet aanpassen. Ook de andere voordelen die GraphQL met zich meebrengt kunnen niet langer worden genegeerd.

De principes van REST die al jarenlang worden gehandhaafd, zorgen een voor gestructureerde, stateless client-server communicatie. GraphQL volgt niet alle constraints van REST omdat het gebruik van GraphQL zorgt voor vrijheid van de client, waarbij er gebruik gemaakt wordt van één en dezelfde endpoint. Het schema geeft de beschikbare types weer, waar de client toegang toe heeft.

Een REST endpoint kan geoptimaliseerd worden om verschillende resources in een response terug te sturen. Dit is een vaak voorkomend patroon, waarbij de endpoints gestructureerd worden volgens de views binnen de clients webapplicatie. Dit heeft als nadeel dat bij een verandering van de UI, de back-end weer aangepast moet worden. Wanneer GraphQL zijn intrede maakt, is de client onafhankelijk van de server, waarbij op termijn het back-end team minder aanpassingen zal moeten doorvoeren en het bedrijf dus baat zal hebben bij de migratie.

Naar nieuwe projecten toe is een overweging van het gebruik van GraphQL op zijn plaats. Door de vooraf gedefinieerde responses per endpoint in een REST API, krijgt de client te veel onnodige data terug. GraphQL elimineert dit fenomeen en zorgt ervoor dat de client bepaalt welke data de response bevat. Ook under-fetching wordt vermeden, waardoor er minder individuele requests gedaan moeten worden. Zoals in de technische demo ook wordt aangetoond, is in de meeste gevallen de grootte van de response kleiner dan bij de REST variant.

Bestaande projecten kunnen in een bedrijfscontext niet zomaar omgevormd worden naar GraphQL. Hier kan het gebruik van GraphQL als wrapper rendabel zijn op het vlak van responsetijden door de eliminatie van over-fetching. Op deze manier kunnen ook verschillende services centraal beheerd worden. Net zoals bij Ordina en Sentia kan deze implementatie, zonder bestaande services te herschrijven, toch verschillende voordelen van GraphQL bieden.

Wanneer geopteerd wordt voor GraphQL, is het belangrijk om tijd en aandacht te besteden aan de beveiliging van de service, ook bij het gebruik van GraphQL als wrapper. Het gebruik van de querytaal zorgt ervoor dat een slecht geoptimaliseerde of kwaadaardige query de onderliggende service zwaar kan belasten en tot een denial of service kan leiden.

Aanbevelingen migratie GraphQL

De vrijheid van de client zorgt voor nieuwe kwetsbaarheden, waarmee de ontwikkelaar van de service rekening moet houden om een denial of service tegen te gaan. Het is aangeraden om in de documentatie van GraphQL te kijken hoe de service tegen deze kwetsbaarheden beschermd kan worden. Dit is een taak voor de back-end developer, aangezien de mate waarin queries beperkt worden, afhangt van de complexiteit van de service.

Het gebruik van GraphQL als wrapper komt meer en meer voor en kan een goede overweging zijn. Op deze manier kan het front-end team ook onafhankelijk werken en gebruik maken van de voordelen van GraphQL. Hier is de optimalisatie van het aantal requests naar de API een moeilijkheid waaraan aandacht moet worden besteed. Het is dus nodig om queries te beveiligen en server-side batching- en caching te voorzien, wat mogelijk is door libraries te gebruiken die door Facebook werden ontwikkeld. Dit heeft ook Bart Wullems ondervonden bij de implementatie van GraphQL als federated schema voor verschillende REST-services binnen Ordina.

Het gebruik van de single access point zorgt ervoor dat er geen gebruik gemaakt wordt van de native http-caching. Dit is een belangrijke factor bij het bouwen van webapplicaties, waartoe REST wel ondersteuning biedt door de uniforme interface. Ook in het technisch onderzoek was duidelijk dat caching een invloed heeft op de webapplicatie. Hier moet bij de client zelf gezorgd worden voor caching. Aangezien dit in GraphQL een terugkomend probleem is, is het dan ook vanzelfsprekend dat hiervoor patterns ontstaan zoals globally unique object IDs.

Autorisatie is geen probleem bij GraphQL en valt eenvoudig te implementeren. Er kan gewerkt worden met de query context, die wordt meegestuurd bij elke request. De beveiliging van de resources hangt af van de manier waarop authenticatie en autorisatie wordt toegepast op de server, net zoals bij REST. Het is aangeraden om autorisatie toe te passen in de business logic layer en niet in de resolvers, wat zorgt voor extra veiligheid en herbruikbaarheid.

Vooraleer GraphQL geïmplementeerd wordt, moet er ook aandacht besteed worden aan de keuze van de server. De Apollo server uit het technisch onderzoek heeft veel functionaliteiten die het gebruik van GraphQL zowel bij de server als de client vergemakkelijken. Het nadeel is echter dat er veel overhead is, waardoor dit een van de tragere servers is in een node omgeving. Apollo heeft dus een lage instapdrempel, wat zorgt voor een snelle leercurve van GraphQL, maar voor een kleinschalige webapplicatie zoals in het technisch onderzoek, waarbij veel van de functionaliteiten van Apollo niet worden gebruikt, kan een andere server voordeliger zijn.

Mogelijkheden persistente queries

Ook wanneer de client niet de vrijheid zou mogen hebben die GraphQL met zich meebrengt, zoals bij een traditionele REST API, bewijst GraphQL nog steeds zijn nut. Het is mogelijk om een lijst te exporteren van de mogelijke queries en mutations. De vrijheid van de client, die zorgt voor efficiëntie, flexibiliteit en onafhankelijkheid wordt dan opgeofferd. Het voordeel tegenover REST is dat hier nog steeds gebruik gemaakt kan worden van dezelfde endpoint en de schema's, waarbij over-fetching en under-fetching beperkt worden, wanneer de API geen publieke doeleinden dient.

Bij een publieke API, waarbij vaste contracten moeten worden gebruikt, is het minder geadviseerd om de API om te vormen naar GraphQL. De voordelen van de querytaal vallen weg en er moet nog extra beveiliging en caching worden voorzien. REST is hier beter geschikt door het gebruik van http-caching, dankzij de verschillende endpoints. Er zijn ook heel wat kwetsbaarheden bij GraphQL, die bij REST niet relevant zijn.

Implementatie GraphQL in een bedrijfscontext

De stap om GraphQL te gebruiken al dan niet als wrapper, is niet gemakkelijk om te maken. Ordinair ervaren de stap ook als moeilijk, aangezien de meeste developers geen ervaring hebben in het bouwen en gebruiken van GraphQL services. Om een iets complexere GraphQL API goed te ontwerpen komt er wel wat denkwerk aan te pas. Deze oefening vergt volgens Bart Wullems typisch in het begin wat extra tijd, door het gebrek aan ervaring. Eens deze hinderpaal overwonnen is, is het effectief implementeren van GraphQL even snel als die van REST langs de backend kant.

Langs de frontend kant is er typisch een versnelling bij het implementeren van de GraphQL service, doordat het eenvoudiger en gemakkelijker is om de API te gebruiken en te integreren in een applicatie. Op termijn wordt het front-end team ook onafhankelijker, doordat die zelf kan bepalen welke data de response zal bevatten. Het is dus verstandig om GraphQL te implementeren, aangezien de tijd die gespendeerd wordt tijdens het leerproces ervan rendabel is.

Bart merkt vooral meerwaarde van GraphQL op aan de begrenzingen van het systeem. Vooral waar de gebruiker van de service een client applicatie is en de behoeften ervan variëren, ziet hij het gemak van GraphQL als querytaal.

6 Conclusie

Om te weten of het rendabel is om over te schakelen van REST naar GraphQL, is het belangrijk om de voor- en nadelen van de migratie tegen elkaar af te wegen. GraphQL zorgt voor vrijheid bij de client doordat die zelf kan bepalen welke data de response zal bevatten. Bij variërende behoeften op de client kan een aangepaste query vaak volstaan, zonder dat er aanpassingen moeten doorgevoerd worden naar de server.

GraphQL zorgt voor efficiëntie en flexibiliteit, met als gevolg dat extra beveiliging nodig is. Het gebruik van de querytaal zorgt ervoor dat de client complexe, geneste queries kan opstellen. Dit kan leiden tot een denial of service. Er is extra werk nodig tegenover de beveiliging van een REST API, als logisch gevolg van de vrijheid die GraphQL met zich meebrengt aan de kant van de client.

Bij de keuze voor GraphQL moet ook overwogen worden of de client deze vrijheid hoort te hebben. Wanneer dit niet zo is, kan in de meeste gevallen REST dienstiger zijn dan GraphQL, aangezien caching en beveiliging van de server eenvoudiger zijn om te implementeren. Het gebruik van GraphQL met zijn schema's wordt dan minder relevant.

Om snel te migreren kan gebruik gemaakt worden van GraphQL als wrapper, waarbij de REST-endpoints aangesproken worden bij het uitvoeren van een query of mutation. Zo moeten bestaande APIs niet herschreven worden om te kunnen genieten van de voordelen van GraphQL. Ook hier moet aandacht besteed worden aan beveiliging, zodat de REST-endpoints niet meer worden opgeroepen dan nodig bij geneste queries. Deze implementatie wordt ook gebruikt als centraal aanspreekpunt voor verschillende services.

De migratie van REST naar GraphQL, al dan niet door het gebruik van GraphQL als wrapper, zal in veel gevallen rendabel zijn door de gereduceerde responsgrootte, die voortkomt uit de eliminatie van overfetching. Daarbovenop kan het front-end team onafhankelijker werken door het gebruik van GraphQL met zijn schema's.

7 Literatuurlijst

Alle bronnen waarvan je gebruikmaakt, zet je in de literatuurlijst. Je gebruikt hiervoor de IEEE-stijl.

- [1] Auth0, "Authorization", *Auth0 Docs*. [Online]. Beschikbaar op: <https://auth0.com/docs/>. [Geraadpleegd: 16-apr-2020]
- [2] "Understanding Denial-of-Service Attacks | CISA". [Online]. Beschikbaar op: <https://www.us-cert.gov/ncas/tips/ST04-015>. [Geraadpleegd: 16-apr-2020]
- [3] "OpenAPI Specification - Version 3.0.3 | Swagger | Swagger". [Online]. Beschikbaar op: <https://swagger.io/specification/>. [Geraadpleegd: 16-apr-2020]
- [4] "GraphQL: A query language for APIs." [Online]. Beschikbaar op: <http://graphql.org/>. [Geraadpleegd: 14-apr-2020]
- [5] "Advantages and Disadvantages of GraphQL | Stable Kernel". [Online]. Beschikbaar op: <https://stablekernel.com/advantages-and-disadvantages-of-graphql/>. [Geraadpleegd: 15-apr-2020]
- [6] N. Eeda, "Rendering real-time dashboards using a GraphQL-based UI Architecture", p. 68.
- [7] M. Cederlund, "Performance of frameworks for declarative data fetching", p. 107.
- [8] "FULL STACK DEVELOPMENT - 3NMCT". Johan Vannieuwenhuysse, 2020-2019.
- [9] humans.txt, "A GraphQL & Node.js Server Built with Express in No Time", *Add a Shopping Cart to Any Website*. [Online]. Beschikbaar op: <https://snipcart.com/blog/graphql-based-cms-tutorial-integration-with-graphcms-nodejs-and-apollo>. [Geraadpleegd: 15-apr-2020]
- [10] "GraphQL vs. REST: What you didn't know", *LogRocket Blog*, 26-jun-2019. [Online]. Beschikbaar op: <https://blog.logrocket.com/graphql-vs-rest-what-you-didnt-know/>. [Geraadpleegd: 15-apr-2020]
- [11] "Protocol Buffers vs. JSON - Bizety". [Online]. Beschikbaar op: <https://www.bizety.com/2018/11/12/protocol-buffers-vs-json/>. [Geraadpleegd: 15-apr-2020]
- [12] "gRPC", *gRPC*. [Online]. Beschikbaar op: <https://grpc.io/>. [Geraadpleegd: 15-apr-2020]
- [13] JamesNK, "Compare gRPC services with HTTP APIs". [Online]. Beschikbaar op: <https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison>. [Geraadpleegd: 15-apr-2020]
- [14] "Everything You Need To Know About API Rate Limiting | Nordic APIs |", *Nordic APIs*, 18-apr-2019. [Online]. Beschikbaar op: <https://nordicapis.com/everything-you-need-to-know-about-api-rate-limiting/>. [Geraadpleegd: 17-apr-2020]
- [15] "Security Points to Consider Before Implementing GraphQL | Nordic APIs |", *Nordic APIs*, 25-apr-2017. [Online]. Beschikbaar op: <https://nordicapis.com/security-points-to-consider-before-implementing-graphql/>. [Geraadpleegd: 17-apr-2020]
- [16] solo, "Graph Data and GraphQL API Development—Leap Graph", *Graph Data and GraphQL API Development—Leap Graph*. [Online]. Beschikbaar op: <https://leapgraph.com/graphql-api-security>. [Geraadpleegd: 15-apr-2020]
- [17] "Security and GraphQL Tutorial". [Online]. Beschikbaar op: <https://www.howtographql.com/advanced/4-security/>. [Geraadpleegd: 17-apr-2020]
- [18] "Securing Your GraphQL API from Malicious Queries", *Apollo Blog*. [Online]. Beschikbaar op: <https://www.apollographql.com/blog/securing-your-graphql-api-from-malicious-queries-16130a324a6b>. [Geraadpleegd: 15-apr-2020]
- [19] "Improve GraphQL Performance with Automatic Persisted Queries", *Apollo Blog*. [Online]. Beschikbaar op: <https://www.apollographql.com/blog/improve-graphql-performance-with-automatic-persisted-queries-c31d27b8e6ea>. [Geraadpleegd: 15-apr-2020]

- [20] M. Choren, "Discovering GraphQL endpoints and SQLi vulnerabilities", *Medium*, 23-sep-2018. [Online]. Beschikbaar op: <https://medium.com/@localh0t/discovering-graphql-endpoints-and-sqli-vulnerabilities-5d39f26cea2e>. [Geraadpleegd: 15-apr-2020]
- [21] "Authorization". [Online]. Beschikbaar op: <https://graphql.org/learn/authorization/>. [Geraadpleegd: 17-apr-2020]
- [22] "Authorization in GraphQL", *Apollo Blog*. [Online]. Beschikbaar op: <https://www.apollographql.com/blog/authorization-in-graphql-452b1c402a9>. [Geraadpleegd: 17-apr-2020]
- [23] G. Brito, T. Mombach, en M. T. Valente, "Migrating to GraphQL: A Practical Assessment", *2019 IEEE 26th Int. Conf. Softw. Anal. Evol. Reengineering SANER*, pp. 140–150, feb. 2019, doi: 10.1109/SANER.2019.8667986.
- [24] M. Vogel, S. Weber, en C. Zirpins, "Experiences on Migrating RESTful Web Services to GraphQL", in *Service-Oriented Computing – ICSOC 2017 Workshops*, Cham, 2018, pp. 283–295, doi: 10.1007/978-3-319-91764-1_23.
- [25] B. Awad, *benawad/node-graphql-benchmarks*. 2020 [Online]. Beschikbaar op: <https://github.com/benawad/node-graphql-benchmarks>. [Geraadpleegd: 15-apr-2020]
- [26] G. Brito en M. T. Valente, "REST vs GraphQL: A Controlled Experiment", *ArXiv200304761 Cs*, mrt. 2020 [Online]. Beschikbaar op: <http://arxiv.org/abs/2003.04761>. [Geraadpleegd: 15-apr-2020]
- [27] "Vue Apollo". [Online]. Beschikbaar op: <https://apollo.vuejs.org/>. [Geraadpleegd: 15-apr-2020]
- [28] S. Luscher, "Wrapping a REST API in GraphQL", 05-mei-2016. [Online]. Beschikbaar op: <https://graphql.org/blog/rest-api-graphql-wrapper/>. [Geraadpleegd: 15-apr-2020]
- [29] "How to wrap a REST API with GraphQL - A 3-step tutorial", *Prisma*. [Online]. Beschikbaar op: <https://www.prisma.io/blog/how-to-wrap-a-rest-api-with-graphql-8bf3fb17547d>. [Geraadpleegd: 15-apr-2020]
- [30] *Zero to GraphQL in 30 Minutes – Steven Luscher*. [Online]. Beschikbaar op: https://www.youtube.com/watch?time_continue=901&v=UBGzsb2UkeY&feature=emb_logo. [Geraadpleegd: 15-apr-2020]
- [31] *Wrapping REST with GraphQL - Jon Wong*. [Online]. Beschikbaar op: <https://www.youtube.com/watch?v=iW4il6wUlvs>. [Geraadpleegd: 16-apr-2020]
- [32] E. Wittern, A. Cha, en J. A. Laredo, "Generating GraphQL-Wrappers for REST(-like) APIs", in *Web Engineering*, Cham, 2018, pp. 65–83, doi: 10.1007/978-3-319-91662-0_5.
- [33] "graphql | npm trends". [Online]. Beschikbaar op: <https://www.npmtrends.com/graphql>. [Geraadpleegd: 15-apr-2020]
- [34] "Bulk downloading - OpenWeatherMap". [Online]. Beschikbaar op: <https://openweathermap.org/bulk>. [Geraadpleegd: 17-apr-2020]
- [35] "Mongoose ODM v5.9.9". [Online]. Beschikbaar op: <https://mongoosejs.com/>. [Geraadpleegd: 17-apr-2020]
- [36] "Sequelize", *Sequelize ORM*. [Online]. Beschikbaar op: <https://sequelize.org/>. [Geraadpleegd: 17-apr-2020]
- [37] "DB-Engines Ranking", *DB-Engines*. [Online]. Beschikbaar op: <https://db-engines.com/en/ranking/relational+dbms>. [Geraadpleegd: 17-apr-2020]
- [38] A. Vázquez-Ingelmo, J. Cruz-Benito, en F. J. García-Peñalvo, "Improving the OEEU's data-driven technological ecosystem's interoperability with GraphQL", in *Proceedings of the 5th International Conference on Technological Ecosystems for Enhancing Multiculturality*, Cádiz, Spain, 2017, pp. 1–8,

doi: 10.1145/3144826.3145437 [Online]. Beschikbaar op: <https://doi.org/10.1145/3144826.3145437>.
[Geraadpleegd: 16-apr-2020]

8 Bijlages

8.1 Verslag Cybercrime Unit

Op 8 januari 2020 waren er op de campus 'GKG' in Kortrijk twee bedrijfsessies die alle studenten verplicht dienden bij te wonen. De eerste sessie ging over de cyber crime unit, gegeven door Francis Nolf. Die praatte vooral over zijn ervaringen en legde vooral klemtoon op waartoe zij bevoegd zijn en waar zij beperkt zijn door de wet.

Francis Nolf is hoofdinspecteur van de politiezone Grensleie. Hij is samen met zijn team bezig met de bestrijding van ICT-criminaliteit, met als doel ook burgers te beschermen tegen allerlei vormen van criminaliteit. Hij schoolt zichzelf ook constant bij om up-to-date te blijven. Zijn unit is verantwoordelijk voor de tapkamer voor af luistering van telefoongesprekken, BOM (bijzondere operaties & middelen) en Bakens, de locatie van een auto doorsturen en volgen.

Vroeger was de CCU minder belangrijk, maar sinds de opkomst van computer crime is deze dienst afzonderlijk blijven groeien. Vroeger was er gerechtelijke, lokale en rijkswacht, waarna in 2001 alles werd samengevoegd en er twee aparte eenheden ontstonden: FCCU en RCCU (federale en regionale). Tot op heden is er nog steeds een tekort aan mankracht in de verschillende teams. Deze zijn onderverdeeld in vier groepen. Team Osint, met de focus op open source intelligence. Team telefonie, vooral bezig met het uitlezen van telefoons. Team onderzoek, is vooral bezig met gegevensdragers (USB-sticks, HDD's en andere opslagmedium). En als laatste Team Hacking, die meer gespecialiseerd is op het internet.

Sinds vier jaar zijn er ook LCCU's, dit zijn lokale computer crime units voor elke politiedienst. Zo kan een onervaren politieman weten wat hij moet doen bij situaties waarvoor hij niet werd opgeleid. Na een algemene uitleg ging Francis meer in op hoe ze juist tewerk gaan in specifieke scenario's. Het kopiëren van harde schijven bijvoorbeeld. Dit doen ze omdat een harde schijf van een laptop verwijderen tegenwoordig niet zo snel gedaan is. M.2 SSD's (zie bijlage) zitten bevestigd op het moederbord. Als hier iets fout loopt bij het ontkoppelen moeten zij ook de kosten dekken.

Voor het kopiëren van data gebruiken zij verschillende programma's. DD, DCFLOP, DDRESCUE en aanverwanten. Zo krijgen zij de mogelijkheid om volledige schijven te klonen of weg te schrijven naar een bestand. Het EFW-formaat is bij hun standaard om een kopie te comprimeren. Je kan ze op deze manier ook gemakkelijk hashen.

Hierna kwam er een uitleg over Caine Forensics. Dit is een linux distributie met veel verschillende tools die zij ter plaatse gebruiken. Caine staat voor Computer Aided Investigative Environment. Een van de tools is SleuthKit, een reeks cmd-tools die gewiste bestanden kunnen ophalen en terughalen met metadata. Ze zoeken op de header van een file, hiervoor gebruiken zij photorec. Zo kan je bijvoorbeeld zoeken op JPG-bestanden. Zo kunnen ze ook foto's terughalen die verwijderd zijn.

Bij de CCU werken zij met allerlei vormen van criminaliteit, ook illegale videobeelden van minderjarigen. Soms zijn deze niet terug te vinden op de zichtbare partities van een hardeschijf. Het gebruik van tools zoals MMLS, kunnen zij de structuur van de partities zien en ook alle verstopte bestanden. FLS laat toe om de gewiste bestanden te bekijken. Er zijn ook verschillende tools met een gui, hiervan gebruiken ze er verschillende. De meest aangehaalde was guymanager, gebruikt DD, om schijven te kopiëren en te comprimeren. Ze kunnen er ook een hashwaarde aan toekennen.

Deze hashwaarde is belangrijk voor de onderzoeksrechter, de tegenpartij moet diezelfde hashwaarde gebruiken, zodat er bewijs is dat ze met dezelfde kopie van de hardeschijf aan het werken zijn. Zo is er geen vervalsing mogelijk.

Om de gespreksgeschiedenis te bekijken op een gsm van een verdachte, maken ze gebruik van Ufed en XRY. Hiermee krijgen ze alles terug, van sms-berichten tot TikTok chats. Ze hebben hiervoor wel de code nodig van de gsm biometric. De code kunnen ze niet breken.

Na de uitleg over Caine Forensics kwam er een uitgebreide toelichting over dataretentie. De grootste bottleneck voor de CCU is dat ze alle info omtrent IP/telefoonnummers maximaal zes maanden mag bijhouden. Vaak is dit veel te kort, als ze aan providers nog data moeten terugvragen. Al deze opvragingen dienen ook te gebeuren via het ministerie van buitenlandse zaken en dit kan soms maanden duren. Ze mogen ook niet altijd deze zaken opvragen, hiervoor hebben ze een vordering nodig van de procureur of de onderzoeksrechter.

Om verwarring te vermijden heeft hij ook het verschil duidelijk gemaakt tussen de procureur en de onderzoeksrechter. Een procureur is iemand die zaken ter harte neemt voor het openbare ministerie. Die vertegenwoordigt het volk. Een onderzoeksrechter kan dwangmaatregelen opleggen (zoals huiszoekingen). Die staat dus boven een procureur. Een procureur kan ook een systeembeheerder vorderen. Dit wil zeggen dat die gegevens moet blootstellen van de baas, waar hij toegang toe heeft. Als die niet gehoorzaamt kan er een gevangenisstraf aan hem worden gegeven. Hij mag geen gegevens blootstellen waartoe hij geen toegang heeft.

Alleen de OGP (officier van de gerechtelijke politie) kan een toestel in beslag nemen. Het toestel moet dan ook direct in een toestand gebracht worden dat er geen extern contact mogelijk is. Gegevens op het toestel raadplegen kan met vordering van de onderzoeksrechter. Facebook laat ook toe data te downloaden aan de hand van gebruikersnaam en wachtwoord. Hiervoor moet de verdachte meewerken. Maar op deze manier kunnen ze ook alle data downloaden, inclusief chatberichten.

Afluisteren van telefoongesprekken kan ook enkel met vordering van de onderzoeksrechter. Alles wat relevant is moet ook volledig worden uitgetypt en om de vijf dagen moet er een verslag verstuurd worden naar de onderzoeksrechter. Na deze uitleg werd het wel duidelijk, alle persoonsgegevens mogen gebruikt worden, maar pas met toestemming van de onderzoeksrechter. Als de gebruiker voorwaarden heeft geaccepteerd bij externe communicatiemiddelen zoals facebook, zijn gegevens eveneens ook beschermd en kan facebook niets vrijgeven zonder zijn inloggegevens.

Tot slot kwam er nog een inlichting over huidige fenomenen die zich daar nu voordoen. De meest gekende: Ransomware, inmiddels al verouderd maar dit is nog steeds een probleem. Er zijn wel verschillende encryptiemethodes gekend en deze kunnen ook ontcijferd worden. Op nomoransom.org vinden we de gekende encryptiemethodes. Ook de meer gekende scams zijn nog steeds actief. Aankopen via challengecode, door zich voor te doen als bediende bij een bank en het slachtoffer verschillende combinaties te laten ingeven op de kaartlezer thuis.

Op het einde gaf hij informatie over de nodige opleiding om een job te kunnen uitoefenen bij de cybercrime unit en welke voordelen arbeiders daar hebben doordat ze voor de overheid werken. De presentatie was leerrijk, maar de job is toch niet voor mij weggelegd. Ik heb gekozen voor de richting web- en app development omdat ik full-stack developer wil worden. Dit valt buiten mijn interessegebied, maar kan misschien voor infrastructure-studenten wel een optie zijn. Na deze presentatie onthou ik hoe de cybercrime unit gemakkelijk een toestel of harde schijf kan blootstellen, maar legaal gezien niet bevoegd is om dit te doen, zonder toestemming van de procureur of onderzoeksrechter.

8.2 Verslag In The Pocket

De tweede sessie werd gegeven door Kenny Deriemaeker en zijn collega van In The Pocket. Na een korte inleiding over hun AR app die ze maakten, kwam Kenny van In The Pocket een voorstelling geven over hoe het project van AR in samenwerking met Woody (de vlaamse marktleider in kinderpyjama's) tot stand kwam. Ze kregen de opdracht om de diertjes die op de pyjama geprint stonden tot leven te brengen aan de hand van Augmented Reality, wanneer een mobiel toestel of tablet de camera richt op het diertje. Na een scan met de app verschijnt de alpaca of dodo die op de pyjama geborduurd is levensecht voor de gebruiker in de echte wereld. Het blijft je volgen en je kan hem verschillende taken laten uitvoeren zoals dansen of eten. De app wordt nu ook massaal gebruikt, aangezien de pyjama's elk jaar meer dan een miljoen keer aangekocht worden en de app gratis beschikbaar is. Op deze manier kan Woody nogmaals uitblinken en aantonen dat zij een waardige marktleider zijn bij het verkopen van pyjama's in Vlaanderen. Een combinatie van AI en AR werd gebruikt om deze app te ontwikkelen om de diertjes te herkennen en te laten verschijnen.

In the pocket focust zich op de user experience en heeft veel aandacht besteed om te beslissen op welke manier de applicatie het eenvoudigst is om te gebruiken bij hun doelpubliek, namelijk kinderen. Het is duidelijk hoe belangrijk user tests zijn. Ook bij in the pocket hebben ze de applicatie voortdurend bijgestuurd om die zo gebruiksvriendelijk mogelijk te maken. De enige manier om te zorgen voor een ideale gebruikerservaring is door het te gaan testen bij kinderen die zulke pyjama's aankopen en op basis hiervan veranderingen door te voeren.

Ze kregen van Woody telkens meerdere afbeeldingen per karakter. Aan de hand van de afbeelding die op de T-shirt stond moet het dier herkend worden om dan het 3D-object weer te geven in de omgeving. Hiervoor hebben ze gebruik gemaakt van machine learning en een grote hoeveelheid aan training data. Er zijn 63 variaties van de pyjama's die gelinkt zijn aan drie verschillende Woody karakters. Het AI-team maakte gebruik van Google's TensorFlow library om een machine learning model te bouwen dat het dier herkent.

Eens het dier herkend werd moest een geanimeerd 3D-object tevoorschijn komen, waarmee de kinderen kunnen interacteren. Aan de hand van de tekeningen die zij kregen van Woody hebben zij deze ook zelf getekend en geanimeerd. Daarna konden ze beginnen met de functionaliteiten van de app te integreren en ervoor zorgen dat de dieren konden rondlopen in een ruimte, door gebruik te maken van Unity. Er wordt dan gebruik gemaakt van Augmented Reality om de app je omgeving te laten begrijpen. Op IOS wordt gebruik gemaakt van ARCore en bij android wordt gebruik gemaakt van ARKit om een vlak te vinden waarop het Woody karakter tevoorschijn komt. Een navigation mesh, die zorgt voor de detectie van de omgeving zorgt ervoor dat het Woody karakter weet waar die kan lopen en tegelijk ook obstakels zoals muren of meubels vermijdt.

Het is belangrijk om een efficiënte manier te vinden om de animaties en verplaatsingen van het object te kunnen testen, zodat niet telkens opnieuw gebouwd en gedeployd moet worden. In the pocket heeft hiervoor zelf een oplossing gevonden, een eigen AR-simulator. Op die manier kunnen ze zonder telkens opnieuw te moeten bouwen de aanpassingen en features testen, hoewel project MARS dit nu ook mogelijk maakt. MARS is een Mixed and Augmented Reality Studio. Deze unity extension geeft creators de mogelijkheid om mixed reality en augmented reality applicaties te bouwen die interacteren met de real-world omgeving.

Voor In The Pocket was het een hele uitdaging, waarbij ze hun niveau naar omhoog hebben getrokken. De AR-applicatie is van hoog niveau en Kenny vergelijkt deze zelfs met de Pokémon Go App. Hij vertelt dat de Woody applicatie zelfs van hoger niveau is en meer het gevoel geeft dat het karakter effectief aanwezig is in de omgeving, doordat die minder statisch beweegt en de gebruiker volgt in de omgeving. In the Pocket en Woody zijn beiden blij met de samenwerking en de resultaten die deze met zich

meebracht. Wat me het meeste bijbleef was hoe ze erin slagen om goed in contact te staan met de klant om zo flexibel tot het gewenste resultaat te komen. Net zoals op mijn stagebedrijf is het belangrijk om snel feedback te ontvangen zodat de klant tevreden is en snel features kunnen worden uitgebreid of aangepast. Ook het usertesten bleek van groot belang, zo hebben ze UI-changes moeten doen om de ervaring voor de kinderen te optimaliseren.

Geraadpleegde bronnen:

<https://www.inthepocket.com/>

<https://www.inthepocket.com/work/woody>

8.3 Project 4 – Installatiehandleiding

Installatiehandleiding REST server

Clone deze repository lokaal. Open de Package-Manager en installeer volgende packages:

```
PM> Install-Package Microsoft.EntityFrameworkCore
PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer
PM> Install-Package Swashbuckle.AspNetCore -Version 5.0.0-rc4
PM> Install-Package MongoDB.Driver
PM> install-package Microsoft.AspNetCore.Identity.EntityFrameworkCore
```

Nu staat het programma klaar voor gebruik en rest ons nog de nodige connecties aan te passen. Zorg dat SQL Server en MsSQL geïnstalleerd zijn.

Download	URL
SQL-Server	https://www.microsoft.com/en-us/sql-server/sql-server-downloads
MsSQL	https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15
MongoDb	https://docs.mongodb.com/manual/installation/
MongoCompass	https://www.mongodb.com/download-center/compass

Indien zowel SQL-Server als MsSQL, MongoDB en MongoCompass geïnstalleerd staan, kunnen we de database en collecties aanmaken.

Eerst maken we een Mongo database aan genaamd 'WeatherDB', indien u een andere naamgeving gebruikt, vergeet deze dan ook niet aan te passen in de config.json. We beginnen met het importeren van de collections in onze Mongo Database.

Deze bevinden zich onder weatherdata. Hierin zitten de metingen en voorspellingen, genomen op 17 maart 2017. Importeer de JSONs als volgt per collectie.

- days: daily_14.json
- currents: weather_14.json
- hours: hourly_14.json
- cities: cities_14.json

Hierna maken we de SQL-database aan voor de users. Maak een database aan onder naam userdb en pas in de appsettings.json de connectionstring aan naar deze database. Dan voeren we een update-database uit, waardoor de tabel aangemaakt wordt om de users op te slaan.

```
PM> Update-database
```

Eens dit gebeurd is kunnen we de API runnen. We kunnen testen of alles goed verlopen is door te surfen naar volgende url: <https://localhost:5001/>

Zonder in te loggen hebben we geen toegang om de actions uit te voeren.

Een post van login of register kan uitgevoerd worden om een JWT-token te krijgen. Waarna we deze toevoegen bovenaan rechts 'Authorize'. Vul het veld in met 'Bearer ', gevolgd door de JWT-token die register of login teruggaf. Nu kunnen we de front-end opstarten, zie installatie handleiding repository WeatherPWA.

Installatiehandleiding GraphQL server

Clone deze repo lokaal op je device. Open het powershell script (dit kan ook via de terminal in VSCode). Om de apollo-server op te starten moeten we alle nodige packages installeren. Dit kan met volgend commando:

```
npm install
```

Nu staat het programma klaar voor gebruik en rest ons nog de nodige connecties aan te passen. Zorg dat MongoDB en MySQL geïnstalleerd zijn.

Het kan zijn dat u voor MySql te installeren eerst python moet installeren. Let op dat u de gevraagde versie installeert.

Download	URL
MongoDb	https://docs.mongodb.com/manual/installation/
MongoCompass	https://www.mongodb.com/download-center/compass
MySql	https://dev.mysql.com/downloads/installer/
Python	https://www.python.org/downloads/

Indien zowel MongoDB als MySql geïnstalleerd staan, kunnen we de database en collecties aanmaken. Dit kan via MongoCompass of de Mongo commandline.

Eerst maken we een database aan genaamd 'WeatherDB', indien u een andere naamgeving gebruikt, vergeet deze dan ook niet aan te passen in de config.json. We beginnen met het importeren van de

collections in onze Mongo Database. Deze bevinden zich onder weatherdata. Hierin zitten de metingen en voorspellingen, genomen op 17 maart 2017.

Importeer de JSONs als volgt per collectie.

- days: daily_14.json
- currents: weather_14.json
- hours: hourly_14.json
- cities: cities_14.json

Hierna maken we de SQL-database aan voor de users. We moeten ook een user connection aanmaken zodat we niet met de root verbinden. Doe dit eerst en geef er rechten aan. Maak een database aan onder naam userdb. Pas de username, password en database aan in de config.json.

Hierna voeren we de migrations uit, waardoor de tabel aangemaakt wordt om de users op te slaan. Open het powershell script in de root folder en voer volgende commando's uit:

```
npm install -g sequelize-cli  
sequelize db:migrate
```

Eens dit gebeurd is kunnen we de server runnen via npm run serve of yarn serve.

We kunnen testen of alles goed verlopen is door te surfen naar volgende url: <https://localhost:4000/graphql/>

Zonder in te loggen hebben we geen toegang om de queries uit te voeren. Rechts kan je "docs" openen en zien welke queries of mutations beschikbaar zijn. De enige die toegankelijk is zonder JWT-token is current.

De eenvoudigste vorm om een query te testen is als volgt:

```
{current(cityname: "Brussels"){time}}
```

Andere queries hebben een JWT-token nodig, wat niet kan via deze interface. We moeten gebruik maken van postman als we andere zaken willen testen. Nu kunnen we de front-end opstarten.

8.4 Project 4 – Gebruikershandleiding

Eens u de installatie hebt voltooid, kan u de PWA gebruiken.

Wanneer u opent in de browser komt u eerst op het volgende scherm. Dit is de loginpagina. U kan niet navigeren naar andere pagina's tot wanneer u succesvol bent ingelogd. Als u niet ingelogd bent klikt u op Sign Up, hier kan u een account aanmaken. Indien u niet aan de criteria voldoet ziet u een foutmelding.

Sign in

Email

tom@gmail.com

Password

.....

Login

Sign Up



Register

Email

jan@gmail.com

Name

Jan

Password

.....

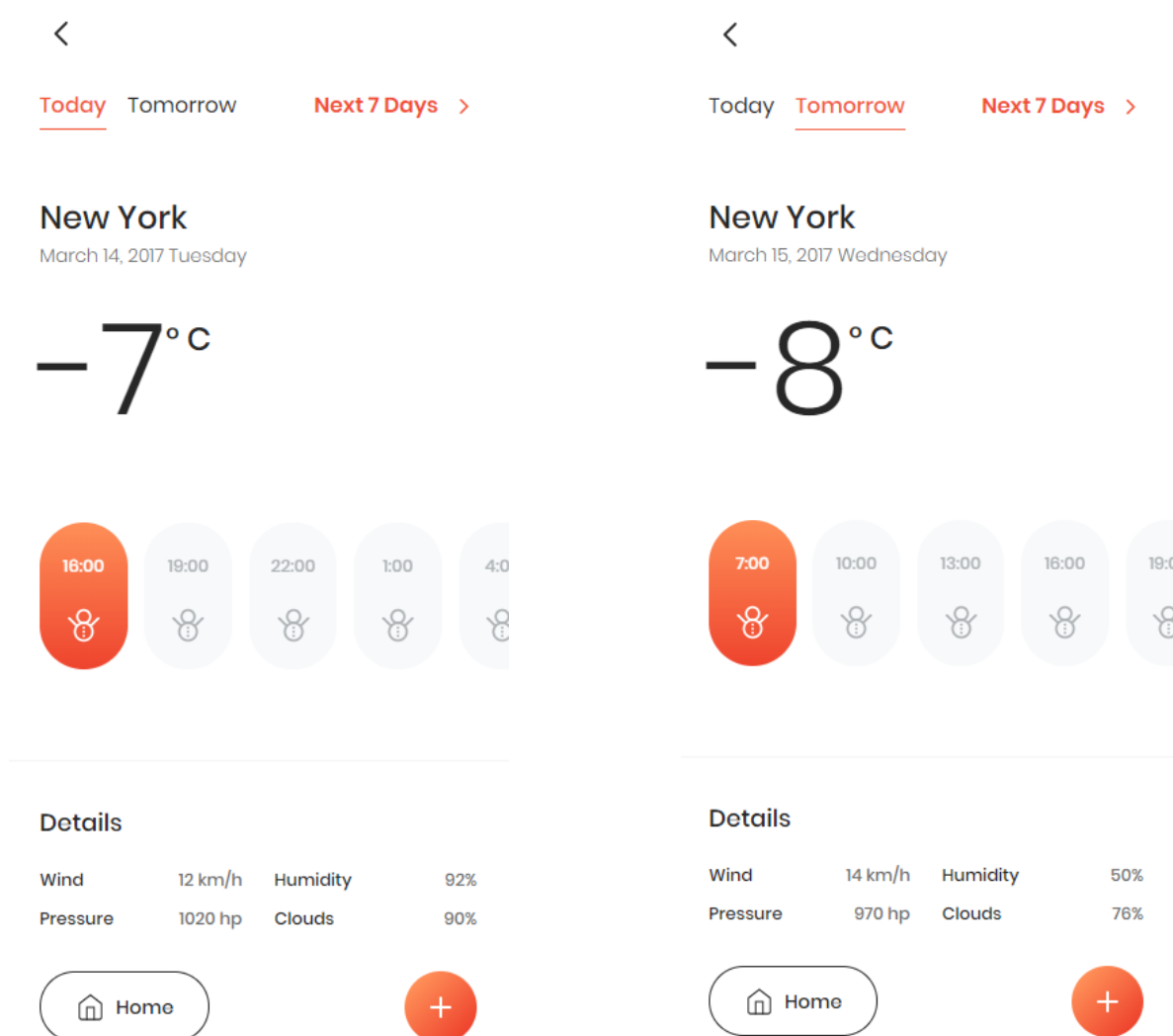
User already exists

Register

Dit is de homepagina. Hier kan u zoeken op een stad en erop klikken om er het weerbericht van te zien. U ziet ook telkens de vijf laatst bezochte steden en de locaties die u zelf hebt toegevoegd.

The image displays two versions of a weather application's home screen. Both screens feature a search bar at the top with a magnifying glass icon and a user profile icon. The left screen has the search bar containing the text 'City'. Below it, the 'Added Locations' section shows three cards: 'New Yekepa' (LR) with a temperature of 27°C, 'Brushy Creek' (US) with 4°C, and 'Bus' (TZ) with 29°C. The 'Recently Viewed' section lists four locations: 'Brugge' (BE) at 9°C, 'Januaria' (BR) at 28°C, 'Brushy Creek' (US) at 4°C, and 'Dobrush' (BY) at 7°C. The right screen shows the search bar with 'new y' entered, which has triggered a dropdown menu with suggestions: 'New Yekepa', 'New York', 'West New York', and 'East New York'. The 'Added Locations' and 'Recently Viewed' sections on the right screen are identical to those on the left screen.

Wanneer u op een locatie klikt komt u op het volgende scherm en kan u kiezen voor Today of Tomorrow:



Zo ziet u de gegevens van die dag. U kan zijdelings scrollen om de uursvooruitzichten te zien. Als u op het plusje klikt wordt deze locatie toegevoegd aan "Added Locations", waarna u de locatie kan terugvinden op de homepage. Als u op Home klikt navigeert u terug naar de Homepage. Wanneer u het overzicht wil zien van de komende week klik u bovenaan rechts op 'Next 7 days'.

Hier ziet u alle gegevens van elke weekdag. Ook hier kan de locatie worden toegevoegd. Om uit te loggen kan u bovenaan rechts op de homepagina naar de profielpagina gaan. Hier ziet u enkel het account waarmee je bent ingelogd. Als u op 'Sign Out' klikt wordt u teruggestuurd naar de loginpagina.

<

Next 7 days

Wednesday

-8°C
-10°C

Wind	14 km/h	UVI	3.3
Humidity	50%	Clouds	76%

Thu

☁ 0%

☔ 15 km/h

-7°C
-12°C

Fri

☁ 4%

☔ 4 km/h

0°C
-2°C

Sat

☁ 100%

☀ 10 km/h

5°C
-2°C

Sun

☁ 0%

☔ 16 km/h

0°C
-9°C

Mon

☁ 99%

☔ 12 km/h

1°C
4°C

Tue

☁ 0%

☔ 14 km/h

5°C
-6°C

Wed

☁ 56%

☔ 5 km/h


7°C
-4°C

Home

+

<

Hi, welcome to WeatherApi



Email

joeri@gmail.com

[→ Sign Out]

Home