



GraphQL vs REST-API – En jämförelse

Robin Blixter

This thesis is submitted to the Fakulteten of datavetenskaper at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Higher Education Diploma in Software Engineering with emphasis in Web Programming. The thesis is equivalent to 10 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Kontakt Information:

Författare:

Robin Blixter

E-mail: r.blixter89@gmail.com

University advisor:

Emil Folino

Department of Computer Science

Fakulteten of datavetenskaper
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

In today's society, large amounts of redundant data are sent over the Internet. This means an unnecessary power consumption that can be reduced with the help of modern technologies like GraphQL. Intersport Sweden runs a large e-commerce business and wanted help implementing a GraphQL server. GraphQL opens up the possibility to control from the client which data is to be retrieved from the server to the client. Representational state transfer (REST) is the most common way today to build APIs, the difference between REST and GraphQL will be investigated in the study.

A GraphQL server was implemented and used to make comparisons with focus on download size, electricity consumption and sustainability. The download size of the object returned from the server for each technology has been compared and converted to electricity consumption, which in turn has been converted into carbon dioxide emissions. A survey was also conducted to find out how popular GraphQL is among developers.

The result shows that a large reduction can be made when using GraphQL. Intersport Sweden can reduce their electricity consumption by 20.15 kWh per month when using GraphQL compared to REST. The reduced electricity consumption can then be converted to reduced carbon dioxide emissions, for Intersport Sweden this means 286 g less carbon dioxide emissions per month. The survey shows that GraphQL is popular among developers and also the technology that is preferred when there is a lot of data that needs to be handled. The most common technique, according to the study, is REST.

The conclusion of the study is that GraphQL should be used when an API consists of a lot of data - it gets faster, draws less power and saves on the environment.

Keywords: GraphQL, REST, API, electricity consumption, sustainability

Abstrakt

I dagens samhälle skickas det stora mängder av överflödiga data på internet. Detta innebär en onödig strömförbrukning som med hjälp av moderna tekniker som GraphQL går att reducera. Intersport Sverige driver en stor e-handel och ville ha hjälp med att implementera en GraphQL-server. GraphQL öppnar upp möjligheten att från klienten kunna styra vilken data som skall hämtas från servern. Representational state transfer (REST) är det vanligaste sättet idag för att bygga API:er, skillnaden mellan REST och GraphQL kommer att undersökas i arbetet.

En GraphQL-server implementerades och den användes för att göra jämförelser med fokus på nedladdningstorlek, strömförbrukning och hållbarhet. En nedladdningstorlek för respektive teknik har tagits fram, den har sedan gjorts om till strömförbrukning och som i sin tur har omvandlats till koldioxidutsläpp. Även en enkätundersökning har genomförts för att ta reda på hur populärt GraphQL är bland utvecklare.

Resultatet visar att en stor reduktion går att göras vid användning av GraphQL. Intersport Sverige kan minska sin strömförbrukning med 20,15 kWh per månad vid användning av GraphQL i jämförelse med REST. Den minskade strömförbrukningen går sedan att räkna om till reduktion av koldioxidutsläpp, för Intersport Sverige innebär detta 286 g mindre koldioxidutsläpp i månaden. Enkätundersökningen visar att GraphQL är populärt bland utvecklare och även den teknik som föredras när det är mycket data som behöver hanteras. Den vanligaste tekniken är enligt undersökningen REST.

Slutsats på arbetet är att GraphQL bör användas när ett API består av mycket data, det blir snabbare, drar mindre ström och sparar på miljön.

Nyckelord: GraphQL, REST, API, strömförbrukning, hållbarhet

Innehåll

Abstract	i
Abstrakt	ii
1 Inledning	1
1.1 Bakgrund	1
1.1.1 GraphQL	1
1.1.2 REST	3
1.1.3 Elasticsearch	3
1.2 Motiv och värde	4
1.3 Syfte	4
1.4 Avgränsa	4
2 Frågeställningar	5
2.1 Forskningsfråga 1	5
2.2 Forskningsfråga 2	5
2.3 Forskningsfråga 3	5
2.4 Forskningsfråga 4	5
3 Metod	7
3.1 GraphQL-server	7
3.2 Kundens REST-API	9
3.3 Forskningsfråga 1 och 2 - Överflödiga data	9
3.4 Forskningsfråga 3 - Hållbarhet	10
3.5 Forskningsfråga 4 - Enkätundersökning	10
4 Litteraturgenomgång	15
4.1 Inledning	15
4.2 GraphQL	15
4.2.1 Vetenskapliga artiklar och studier	15
4.2.2 Studie 1	16
4.2.3 Studie 2	17
4.3 Hållbarhet	18
5 Resultat	19
5.1 JSON-dokument	19
5.1.1 Klienten	19
5.1.2 Storlek och nedladdningstid	23

5.2	Hållbarhet	25
5.2.1	Vad innebär det i minskad koldioxidutsläpp?	25
5.3	Enkätundersökning	26
5.3.1	Svarsresultat	27
6	Analys och diskussion	34
6.1	Forskningsfråga 1	34
6.2	Forskningsfråga 2	34
6.3	Forskningsfråga 3	35
6.3.1	Alternativt sätt att räkna	35
6.4	Forskningsfråga 4	35
7	Slutsatser och Framtida arbete	37
7.1	Slutsatser	37
7.2	Framtida arbete	37
	References	39
A	Bilaga	41
A.1	JSON-dokument	41

1.1 Bakgrund

Denna studie fokuserar på hållbarhet inom internet. Det är mycket data som skickas över internet från olika servrar till klienter. Den data som inte används räknas som överflödigt och genererar en onödig strömförbrukning. Denna onödiga strömförbrukning skapar koldioxidutsläpp, som hade kunnat undvikas. Detta är något som går att lösa med nya tekniker som GraphQL.

Bakgrunden till denna studie är ett uppdrag för Intersport Sverige som driver en stor e-handel. De vill öppna upp nya möjligheter för sina utvecklare genom att implementera ett nytt Application Programming Interface (API) [12]. Deras webbplats består av dynamisk data som hämtas direkt från deras nuvarande API som är ett Representation State Transfer-API (REST-API) [7]. Det nya API:t skall kunna ställa klient-specificerade frågor mot servern, därför kommer det skapas med den moderna tekniken GraphQL.

1.1.1 GraphQL

GraphQL är ett frågespråk (query-language), som kan hämta data från olika källor (REST-API, databaser, text-filer eller andra tjänster som Elasticsearch) [14]. Det skapades av Facebook 2012 och släpptes som öppen källkod 2015. Enligt Facebook så skapades GraphQL på grund av att de såg brister i sitt dåvarande REST-API främst för de mobila applikationerna där all data som skickades från deras slutpunkter (end-points) inte behövdes, antingen var de tvungna att skapa ett nytt API för de mobila enheterna eller hitta en bättre lösning – den lösningen blev GraphQL [6]. En stor fördel med GraphQL jämfört med REST-API är att klienten bara får den data som efterfrågas.

Hur GraphQL fungerar

Enkelt förklarat så gör GraphQL det möjligt för klienten att ställa frågor (queries) till en databas. Till skillnad mot REST-API där klienten kallar på slutpunkter. Databasen i GraphQL definieras av ett schema som klienten har tillgång till. Listing 1.1 visar hur ett GraphQL-schema kan se ut.

```
1 type Book {  
2   id: String!  
3   title: String  
4   author: Author
```

```
5     pages: Int
6 }
7
8 type Author {
9     name: String
10    books: [Book]
11 }
12
13 type Query {
14     getBook(id: String!): Book
15     getBooks: [Book]
16     getAuthors: [Author]
17 }
```

Listing 1.1: Exempel på GraphQL-schema

Detta schema definierar typerna **Book** och **Author**. Med nyckelordet *type* definieras objekt-typer. Varje objekt har egna fält som även är typdefinerade. Objektet **Book**, består av fyra fält (id, title, author och pages). Där id är av typen sträng och utropstecknet betyder att fältet inte får vara tomt. Title är av typen sträng, author är av typen **Author** som är ett eget objekt. Pages är av typen **Int**. Nästa objekt **Author** består av fälten name och books. Author har ett namn av typen sträng. Fältet books är en lista av objektet **Book**. I alla scheman behöver det även finnas typen **Query**, där definieras vilka frågor som kan ställas av klienten, samt vilka argument som behöver skickas med.

Klienten kan sedan ställa frågor till servern genom att använda GraphQLs egna frågespråk. I Listing 1.2 hämtar klienten en bok med id-strängen och väljer att få tillbaka data från fälten title, author.name och pages.

```
1 query BookById{
2     getBook(id:"1234") {
3         title
4         author {
5             name
6         }
7         pages
8     }
9 }
```

Listing 1.2: GraphQL-fråga

Klienten får då tillbaka ett JSON-dokument som kan se ut som i Listing 1.3.

```
1 {
2     "data": {
3         "getBook": {
4             "title": "The Fellowship of the Ring"
5             "author": {
6                 "name": "J. R. R. Tolkien"
7             }
8             "pages": 423
9         }
10    }
11 }
```

Listing 1.3: JSON-dokument som returneras från servern

För att detta skall fungera så krävs det något som kallas för resolvers. Det är dessa resolver-funktioner som gör om klientens fråga till data som kan returneras. Resolvers är helt enkelt funktioner som hämtar objekten som efterfrågas från någon typ av databas. Det finns en till typ i GraphQL som kallas för *mutation*, den används när klienten skall skriva data till databasen. Denna studie kommer inte använda mutation och det är inget som kommer att tas upp här [5].

1.1.2 REST

Representational state transfer är ett IT-arkitekturbegrepp som skapades av Roy Fielding år 2000[11]. Fielding och hans kollegor ville skapa en standard där en server kunde prata med vilken annan server som helst i världen. De kom fram till sex begränsningar:

- Client-server: Det behövdes en tydligt separering på klient och server. Där klienten skickar en begäran (request) till en server som skickar tillbaka ett svar (response). [11]
- Stateless: Samspelet mellan klient och server behövde begränsas till att kommunikationen alltid skulle vara stateless, vilket innebär att varje begäran från klient till server skall innehålla all nödvändig information som krävs för att förstå begäran. [11]
- Cache: För att förbättra nätverkseffektiviteten la de även till cache-begränsningar. Dessa begränsningar kräver att svaret från servern märks som cachbart eller icke-cachbart. Om svaret är cachbart, kan klienten återanvända svarsdatan. [11]
- Uniform Interface: Definierar gränssnittet mellan klienter och servrar. Kortfattat innebär detta att HTTP-metoder skall användas (POST, GET, PUT, PATCH, DELETE). Samt att URI skall användas som resurs och en HTTP-response med status och body skall returneras. [11][22]
- Layered system: En klient skall inte veta om den är ansluten till slutservern eller om det är en mellanhand på vägen. Detta för att kommunikationen inte skall påverkas av en proxy eller någon form av lastbalansör mellan klient och server. Poängen är att förbättra systemets skalbarhet. [11]
- Code on demand: Denna begränsning är frivillig, den gör det möjligt att skicka tillbaka körbar kod från server till klient [11].

1.1.3 ElasticSearch

ElasticSearch är en opensource tjänst som hämtar all typ av data från olika källor. Elasticsearch är byggt på Apache Lucene. Fördelen med ElasticSearch är att det är lätt att skapa REST-apier, som är snabba och framför allt skalbara. ElasticSearch tar emot rådata och denna data indexeras in i ElasticSearch, sedan kan användarna hämta data genom att ställa komplexa frågor [9].

Intersport Sverige använder denna tjänst idag för att indexera sin data, vilket ger dem ett REST-API som snabbt hämtar data. När ett GraphQL-API implementeras i denna studie behöver all data hämtas från ElasticSearch.

1.2 Motiv och värde

Kunden vill i framtiden bygga sin webbplats mer statisk med tekniker som GatsbyJS[13], vilket kan bli problematiskt med deras nuvarande API som innehåller mycket logik och där det inte går att styra hur mycket eller lite data som skall hämtas. Med ett nytt API där deras utvecklare själva kan justera den data som hämtas, blir det enklare att skapa statiska webbsidor, som i sin tur skapar en snabbare webbplats.

Ifall kunden vill skapa en ny webbplats för mobila enheter eller en applikation med ett eget API, som inte hämtar samma data som deras nuvarande API. Då kan de istället använda sig av GraphQL där front-end utvecklarna själva styr över vilken data som skall hämtas och inte behöver oroa sig för att hämta för mycket data. Det är lätt att lägga till ny data i GraphQL utan att förstöra eller tvinga utvecklarna att skapa en ny API-version, som är problematiskt med REST-API.

1.3 Syfte

Vi lever i en värld där internet är större än någonsin och det ständigt kommer nya tekniker, GraphQL är en av många. En stor fördel med GraphQL, som kommer vara fokusområdet för denna studie, är att det är möjligt att från klienten välja vilken data som skall hämtas från databasen och skickas från servern tillbaka till klienten. I den mer etablerade tekniken REST-API, finns inte möjligheten att göra detta från klienten. Att skicka en viss mängd data över internet, kostar en viss mängd i strömförbrukning. Det finns ett samband mellan storleken av datan som skickas och strömförbrukningen. Som det ser ut idag, så är det mycket överflödiga data som skickas från olika servrar till klienter runt om i världen. Denna studie kommer att ta reda på om det är möjligt att minska på storleken på den data som skickas tillbaka till klienten från servern. Detta bör även innebära en vinst för miljön, eftersom mindre strömförbrukning i sin tur bör innebära mindre koldioxidutsläpp. Studien tar reda på om så är fallet. Studien kommer även kunna hjälpa utvecklare som vill se jämförelser mellan REST-API och GraphQL-API samt hjälpa till att visa hur man kan göra för att implementera GraphQL och koppla samman med en annan datakälla. Studien ger också en inblick om vad andra utvecklare vet om GraphQL och vilken teknik som föredras.

1.4 Avgränsa

Det kommer att skapas ett GraphQL-API för att hämta Intersports produkter genom att skicka med produktnummer. Inom programmering finns uttrycket CRUD (Create Read Update Delete). I denna studie är det bara R:et, att hämta och läsa den data som finns i databasen. Det skapade GraphQL-API:t kommer inte att kopplas samman med en klient. De enda verktygen som används för att hämta och läsa datan är GraphQL Playground[21] och Postman[20].

Fokusområden

- Over-fetching – Hämtar för mycket data, får data som inte används.
- Hållbarhet - Genom att minska datan som skickas från server till klient så minskas strömförbrukningen.
- Popularitet bland utvecklare - Hur stor kunskap finns det kring GraphQL bland andra utvecklare. Vad tycker de om GraphQL?

2.1 Forskningsfråga 1

Vid användning av GraphQL, hur många rader färre JSON-data får klienten tillbaka vid en API-förfrågan?

GraphQL gör det möjligt för klienten att bara hämta den JSON-data som behövs. Arbetet skall ta reda på hur många rader JSON-data som kan plockas bort från klient-sidan.

2.2 Forskningsfråga 2

Vid användning av GraphQL, vad är den minskade storleken på det returnerade JSON-dokumentet?

Studien undersöker hur många kilobytes det går att reducera det returnerade JSON-dokumentet som skickas från servern.

2.3 Forskningsfråga 3

Går det att minska världens koldioxidutsläpp genom att byta till GraphQL?

Genom att minska den totala storleken på det returnerade objektet från servern bör det även innebära minskad strömförbrukning. Studien skall ta reda på om det är möjligt att minska världens koldioxidutsläpp genom att byta till GraphQL.

2.4 Forskningsfråga 4

Hur populärt är GraphQL bland utvecklare?

GraphQL är fortfarande en relativt ny teknik. En enkätundersökning kommer att

genomföras, som mål att ta reda på hur stor kännedom utvecklare runt om i världen har kring GraphQL, samt vilka tekniker som föredras.

3.1 GraphQL-server

För att kunna göra jämförelser mellan GraphQL och REST behövdes en GraphQL-server skapas, som hämtar samma data som det befintliga REST-API:t. Denna server skapades i Node.JS[18] som är JavaScript på server-sidan. Det vanligaste webb-ramverket för att implementera webb-applikationer och API:er för Node är Express.js[10], vilket används även i detta fall. För att skapa GraphQL-servern användes Apollo Server[3] som är en open-source lösning för att implementera GraphQL på server-nivå. Oftast används något som kallas Apollo Client för att sedan koppla samman servern med klienten. I detta arbetat kommer däremot inte en klient att skapas. Kunden har valt att helt och hållet gå över till TypeScript istället för JavaScript, därför användes ytterligare ett ramverk för att skapa GraphQL-servern - TypeGraphQL[24] som är ett ramverk för GraphQL-API. Det har många fördelar när man arbetar med TypeScript. Med hjälp av klasser och decorators går det att definiera egna GraphQL-scheman. I listing 3.1 initieras servern och schemat byggs upp i ApolloServer med hjälp av TypeGraphQLs buildSchema-funktion, där ProductResolver används.

```

1 import "reflect-metadata";
2 import express from "express";
3 import { ApolloServer } from "apollo-server-express";
4 import { buildSchema } from "type-graphql";
5 import { ProductResolver } from "../resolvers/ProductResolver";
6 import { ErrorInterceptor } from "../interceptor/ErrorInterceptor";
7
8 (async () => {
9   const app = express();
10
11   const apolloServer = new ApolloServer({
12     schema: await buildSchema({
13       resolvers: [ProductResolver],
14       validate: true,
15       nullableByDefault: true,
16       globalMiddlewares: [ErrorInterceptor]
17     }),
18     context: ({ req, res }) => ({ req, res })
19   });
20
21   apolloServer.applyMiddleware({ app, cors: false });
22   const port = process.env.PORT || 8000;
23   app.listen(port, () => {

```

```

24     console.log('server started at http://localhost:${port}/graphql
25         ');
26   });
  }());

```

Listing 3.1: Apollo-server-express

Inuti ProductResolver hämtas alla data som behövs från Elasticsearch. Detta görs genom att använda en Elasticsearch-klient för Node.js[8]. För att bygga upp frågor mot Elasticsearch användes ytterligare ett bibliotek, som heter elastic-builder[15]. Som hjälper till att bygga upp frågorna som ställs mot elastic-search. Dessa frågor blir ofta stora och komplexa, därför är det smidigt med en builder som gör detta åt utvecklaren med autocompletes. I listing 3.2 på raderna 1 - 9 är koden för Elasticsearch-klienten. Där ser vi searcher-funktionen som tar emot parametern bodyQuery, som är den query som byggs upp av Elastic-Builder. På raderna 11 - 39 är en funktion som hämtar produkt-saldo av en produkt. Den tar emot två parametrar (itemnumber och store), där itemnumber är produktnumret och store är ett id-nummer för den butik som sökningen skall leta efter produktsaldo.

```

1  // Elasticsearch searcher
2  async function searcher(bodyQuery: { toJSON: () => any }) {
3    const { body } = await client.search({
4      index: "productinfo-prod",
5      body: bodyQuery.toJSON()
6    });
7
8    return body;
9  }
10
11 export async function fetchSaldoByItemNumber(
12   itemnumber: string,
13   store: string
14 ) {
15   // Fetch saldo by itemnumber
16   const requestBody = esb.requestBodySearch().query(
17     esb
18       .hasChildQuery()
19       .type("saldo")
20       .query(
21         esb
22           .boolQuery()
23           .must([
24             matchQuery("store", `${store}`),
25             matchQuery("_routing", `${itemnumber}`)
26           ])
27       )
28       .innerHits(
29         esb
30           .innerHits("saldo")
31           .size(200)
32           .source(["quantityAvailable", "store", "instock"])
33           .sort(esb.sort("store.keyword", "asc"))
34       )
35   );
36   const body = await searcher(requestBody);
37

```

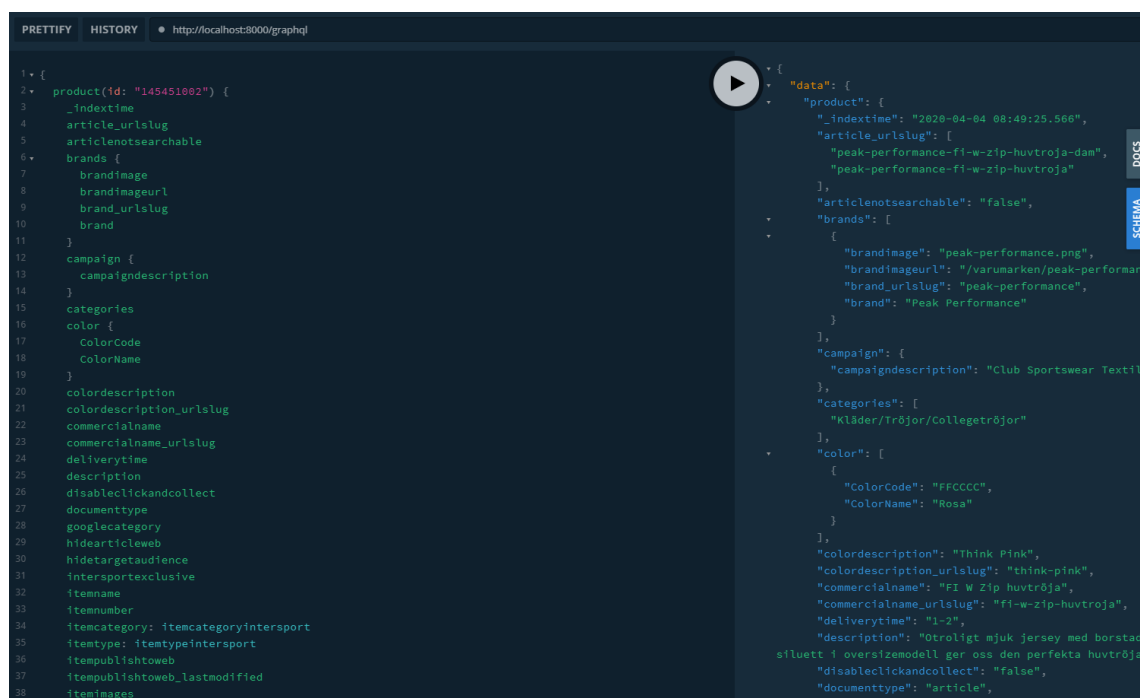
```

38   return body.hits.hits;
39 }

```

Listing 3.2: Elasticsearch Node Client

När servern är igång, är det möjligt att nå GraphQL Playground[21]. Där utvecklaren kan nå GraphQL-servern utifrån klientens perspektiv, med möjligheter att ställa frågor mot GraphQL. Där är det även möjligt att se GraphQL-schemat i sin helhet. I figur 3.1 skickas frågan *product* med id-numret *145451002* till GraphQL-API:t där flera fält som går att hämta har valts. Till höger om play-tecknet visas den returnerade datan.



Figur 3.1: GraphQL-playground

3.2 Kundens REST-API

Intersports REST-API är uppbyggt på Findwise i3[23] som är ett ramverk för att skapa sökdrivna applikationer. Data indexeras med hjälp av Elasticsearch. Findwise i3 och Elasticsearch erbjuder ett REST-API med sökning, komplettering och klickloggning. I denna studie är det bara sökning som används. En GET-request skickas till: `/rest/apps/webstore/searchers/productdetail?id={productID}`, vilket returnerar ett JSON-dokument med produktinformation (exempel finns i bilaga A på sida 41).

3.3 Forskningsfråga 1 och 2 - Överflödiga data

När ett nytt API är implementerat kommer jämförelserna startas. Hela det returnerade JSON-dokumentet som kommer från det befintliga REST-API:t kommer att

undersökas. Detta för att ta reda på vilken data som används av klienten, vilka rader som är överflödiga och skulle kunna plockas bort i GraphQL. För att göra detta krävs det att, tillsammans med kunden, gå igenom klienten och kartlägga vilken data som behövs vid uppvisning av en produkt på webbplatsen.

En applikation kommer att användas för att jämföra storlek på den data som skickas från servern till klienten vid efterfrågan av data. Postman[20], är en API-klient som gör det möjligt att skicka GraphQL och REST-förfrågningar direkt via applikationen.

3.4 Forskningsfråga 3 - Hållbarhet

När forskningsfråga 1 och 2 är besvarad finns det en siffra på hur mycket mindre den data som returneras till klienten är. Den reducerade storleken bidrar till minskad strömförbrukning. Ett medelvärde för kostnaden av dataöverföring över internet kommer att behövas. Detta värde kommer att hämtas från relevanta studier som hittas, mer om detta i litteraturgenomgången. För att räkna ut den minskade strömförbrukningen vid användning av GraphQL för varje inkommande förfrågan, behöver en uträkning göras. Kostnaden av dataöverföring skall multipliceras med den minskade storleken på JSON-dokumentet, resultatet skall sedan multipliceras med antal requests som sker till API:t. När detta är gjort skall det beräknas hur mycket den minskade strömförbrukningen innebär i minskad koldioxidutsläpp. För detta används den senaste mätningen från European Environment Agency[1], där Sveriges, och hela Europas, koldioxidutsläpp från elproduktion är uppmätt.

3.5 Forskningsfråga 4 - Enkätundersökning

Data kommer samlas in genom en enkätundersökning. Enkätundersökningen skapas i Google Form. Det kommer vara en kort enkät med frågor kring utvecklarnas erfarenhet inom programmering, deras kännedom om REST och GraphQL, samt vilken teknik de föredrar. Målet är att enkäterna skall nås av alla olika utvecklare, ingen specifik grupp. Enkäterna skickas ut på olika sociala medier: Reddit, Facebook och Discord. Frågorna till enkäten visas nedan i figur 3.2 - figur 3.5.

GraphQL vs. REST

In this survey, I will find out how much developers know about GraphQL

**Obligatorisk*

Education: What is your highest level diploma/degree you have acquired? *

☐ Associate's degree

☐ Bachelor's degree

☐ Master's degree

☐ Doctoral degree

☐ None

In which country do you live? *

Ditt svar _____

What role do you currently most identify with? *

☐ Student

☐ Employee

☐ Consultant

☐ Manager

☐ University teacher or professor

☐ Retiree

☐ Övrigt: _____

Figur 3.2: Enkätfrågor - sida 1/4

How long have you been programming? *

- ☐ < 1 year
- ☐ 1-2 years
- ☐ 2-3 years
- ☐ 3-5 years
- ☐ > 5 years

What does programming mean to you? *

- ☐ Professional work
- ☐ Hobby outside of work

Have you heard about GraphQL, before this? *

- ☐ Yes
- ☐ No

Have you used GraphQL? *

- ☐ Yes
- ☐ No

Figur 3.3: Enkätfrågor - sida 2/4

What type of application did you use GraphQL with?

Only answer this if you have used GraphQL.

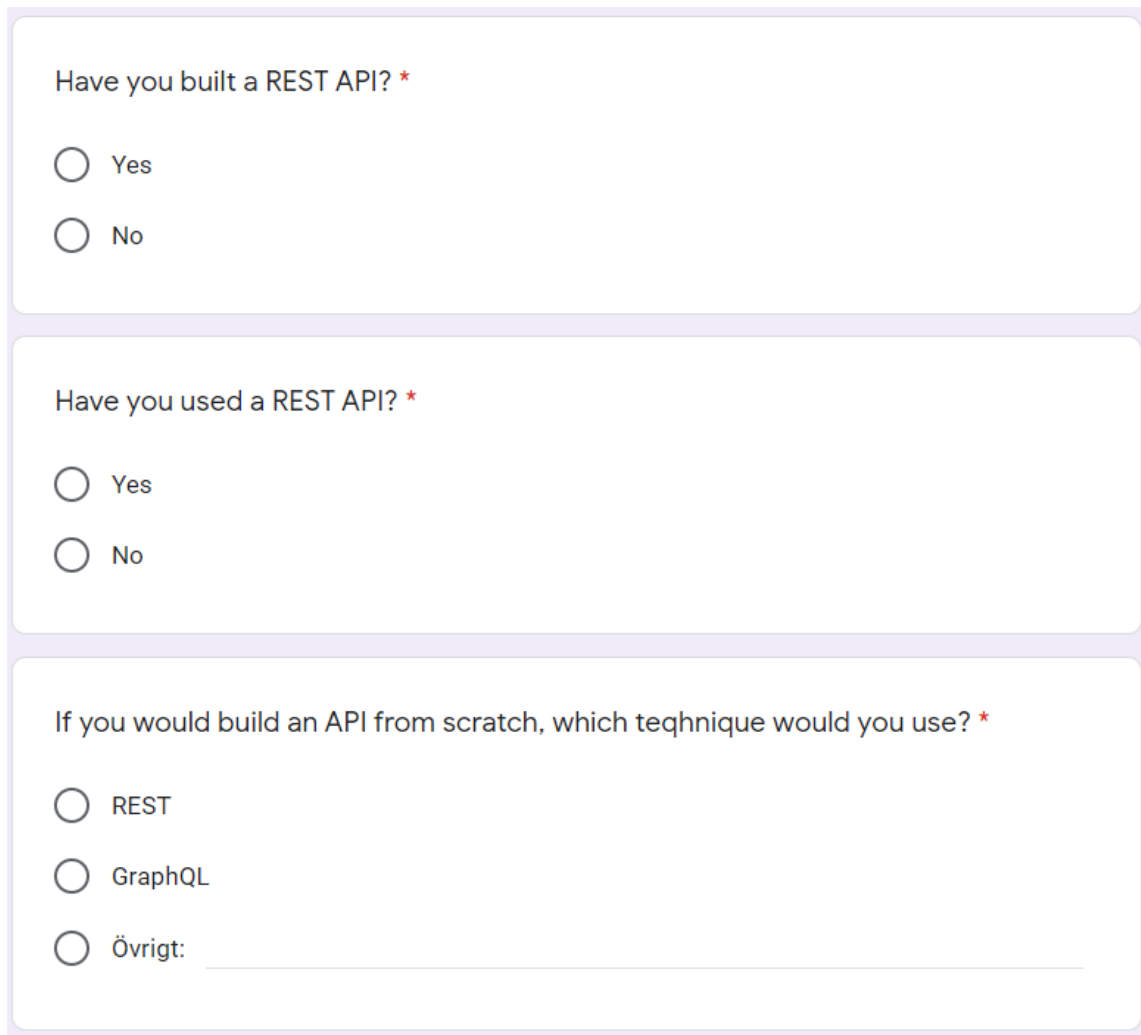
- ☐ Blog
- ☐ E-commerce
- ☐ Social network
- ☐ Streaming service
- ☐ Native App
- ☐ Övrigt: _____

Which programming languages did you use with GraphQL?

Only answer this if you have used GraphQL.

- ☐ JavaScript
- ☐ Python
- ☐ PHP
- ☐ C#
- ☐ Java
- ☐ Perl
- ☐ Ruby
- ☐ Rust
- ☐ Swift
- ☐ Go
- ☐ Övrigt: _____

Figur 3.4: Enkätfrågor - sida 3/4



Have you built a REST API? *

☐ Yes

☐ No

Have you used a REST API? *

☐ Yes

☐ No

If you would build an API from scratch, which technique would you use? *

☐ REST

☐ GraphQL

☐ Övrigt: _____

Figur 3.5: Enkätfrågor - sida 4/4

4.1 Inledning

Vetenskapliga källor inom GraphQL är inte många i antal. Framför allt inte inom specificerade områden som till exempel hållbarhet. Detta beror främst på att det fortfarande är en relativt ny teknik. *“Since it is a new technology, papers about GraphQL are not common in the scientific literature. Therefore, for such emerging technologies, a grey literature tends to provide a better coverage of relevant documents than a traditional literature review.”*[5]

För att hitta litteratur kommer främst BTHs egna biblioteksverktyg som kallas för Summon användas, där går det att hitta litteratur från hela världen. Sökord som används:

- graphql
- graphql energy
- rest api
- energy consumption of data transfer
- electricity intensity

4.2 GraphQL

4.2.1 Vetenskapliga artiklar och studier

Vid sökning efter litteratur på GraphQL hittades främst två studier inom samma område som detta arbete. Dessa två studier kommer att sammanfattas och användas som material i arbetet. Studie 1: ‘Experiences on Migrating RESTful Web Services to GraphQL’ där det görs jämförelser mellan REST och GraphQL, deras främsta mål med studien är att ta reda på hur mycket det går att minska JSON-dokumentet som returneras från servern vid användning av GraphQL jämfört med REST. Studie 2: ‘A GraphQL approach to Healthcare Information Exchange with HL7 FHIR’ även här görs det jämförelser mellan REST och GraphQL, där de tittar på svarstorleken och även laddningstiden från servern vid respektive teknik.

4.2.2 Studie 1

Den mest relevanta studie inom ämnet ‘GraphQL och storlek på returnerad data’, är ‘Migrating to GraphQL: A Practical Assessment’[5]. Denna studie är uppdelad i tre delar. Först gjordes en grå litteraturs-undersökning, där de gick igenom bloggar, tutorials och liknande webb-artiklar och filtrerade sedan ut sådant som inte var trovärdigt. Dessa trovärdiga artiklar fick sedan svara på två forskningsfrågor:

“RQ1: What are the characteristics and benefits of GraphQL?”

“RQ2: What are the main disadvantages of GraphQL?”

Resultat på RQ1

- GraphQL är starkt typbestämt vilket bidrar till bättre verktygsstöd. Med bättre error-meddelanden, redan innan en fråga skickas iväg till servern.
- GraphQL erbjuder klientspecifierade frågor. Vilket innebär att en GraphQL-fråga får tillbaka exakt det den ber efter.
- GraphQL har en hierarkisk datamodell, detta öppnar upp möjligheten för att hämta data från flera källor i en enda förfrågan.
- GraphQL tar bort behovet av att öka versionsnummer på API:t, eftersom det går att lägga till nya fält utan förstöra något.

Resultat på RQ2 (nackdelar)

- GraphQL har inte support för att dölja information. Det går inte att lägga till privata fält, vilket innebär att alla fält är tillgängliga för klient-applikationen.
- GraphQL-frågor tenderar att vara svårare att implementera, vilket kan vara mer tidskrävande i stora API:er.
- GraphQL har svårare för att cacha data, eftersom alla frågor mot servern kan se olika ut. Det krävs att utvecklaren har implementerat cachning på ett korrekt sätt på server-sidan.
- GraphQL kan ha en negativ effekt på prestandan. GraphQL-servern behöver processera komplexa frågor, med djupa nestingar, vilket kan vara väldigt krävande. Utvecklaren behöver se till att det finns begränsningar på hur dyra frågorna får vara.

Del två

I den andra delen av studien (Migration Study), strävar de efter att utvärdera två viktiga egenskaper som är kopplade med resultatet av den första delen av studien:

1. *“Clients can precisely request the data they need from servers (due to the support to client-specific queries).”*
2. *“Clients rely on a single endpoint to retrieve the data they need (due to a hierarchical data model).”*

Fortsatt i den andra delen av studien, skall de migrera sju klient-applikationer som är baserade på REST-API:er till GraphQL. De skall sedan svara på två forskningsfrågor med de nya klient-applikationerna. Deras två forskningsfrågor lyder:

1. “*RQ3: When using GraphQL, what is the reduction in the number of API calls performed by clients?*”
2. “*RQ4: When using GraphQL, what is the reduction in the number of fields of the JSON documents returned by servers?*”

Resultat på RQ3

GraphQL erbjuder en hierarkisk datamodell, som tillåter klienter att hämta data från flera källor i en enda förfrågan. Däremot så hittade de väldigt få tillfällen att implementera den typen av frågor. Detta på grund av att de flesta klient-funktionerna hämtar data från en enda REST-slutpunkt.

Resultat på R4

Vid användning av REST, får klienterna tillbaka stora JSON-dokument, där bara ett fåtal av fälten används. Vilket kallas för over-fetching. Skillnad mot GraphQL där klienten bara frågar efter de fält som behövs. I studien innebär detta en reducering från 93,5 till 5.5 i antal JSON-fält returnerade från REST jämfört med GraphQL - en minskning på 94%.

Del tre

I den tredje delen av studien skulle forskningsfrågan RQ5 besvaras:

“*RQ5: When using GraphQL, what is the reduction in the size (in bytes) of the JSON documents returned by servers?*”

Här var målet att använda mer realistisk data. De hämtade en lista över artiklar som publicerades i två nyligen och relevanta programvarukonferenser. Sedan valdes sju artiklar som förlitar sig på GitHub för att skapa en dataset med data om öppen källkodsprojekt. Fördelen är att dessa artiklar tydligt beskriver vilken data som används. Det blir därför lätt att bara hämta den data som behövs. Resultatet av detta blir att REST i genomsnitt får tillbaka ett JSON-dokument som är 9.8 MB stort jämfört med GraphQL som får tillbaka ett dokument som är 86 KB. Detta innebär en reducering på 99%.

4.2.3 Studie 2

‘A GraphQL approach to Healthcare Information Exchange with HL7 FHIR’[17] är en studie som gör en jämförelse på REST och GraphQL. Den syftar till att utvärdera huruvida en migrering från REST till GraphQL är skalbar och välfungerande. För att utvärdera detta görs det först ett test för att jämföra svarstorleken och svarstiden. Sedan görs det även ett utvärderingstest. Deras mål med det första testet var att se hur mycket extra information som hämtas från slut-punkterna vid hämtning av information genom REST, samt hur mycket av den informationen som används av

mobilapplikationen. Resultatet blev att ungefär 50% av informationen används inte av mobilapplikationen. På utvärderingstestet syns det tydligt en ökning i genomströmning och responstiden är betydligt snabbare när endast den data som behövs hämtas vid användning av GraphQL. “*The result is interesting to related stakeholders as the throughput and response time are directly associated with operating costs and performance of the system, which in turn are associated with the user adherence.*” Användning av GraphQL kan alltså ha en direkt påverkan på kostnad och prestanda av systemen.

4.3 Hållbarhet

Forskningsartikeln *Profiling Energy Efficiency and Data Communications for Mobile Internet of Things* tar upp ämnet energieffektivitet för datakommunikation, med fokus på mobiltelefoner. De skapar en prototyp för att hämta data över cloud, som är så energisnål som möjligt. De går in på varför de väljer att använda GraphQL: “*Mobile and web applications are heavily using HTTP RESTful services and ad hoc endpoints to obtain data nowadays. The core problem with this approach is that the response data is entirely decided by the server. The server-side application might be responding to the client application with more data than required. For example, a simple client application showing current weather is getting more information than what the application displays on the screen. Facebook is one of the most energy consuming applications due to user-engaged usage, according to our measurements. GraphQL is designed as remedy to the above-mentioned problem. It is a query data language for the API and a server-side runtime for executing queries. It provides a complete and understandable description of the data in the API, giving clients the power to ask for exactly what they need and nothing more.*”[16]

Electricity Intensity of Internet Data Transmission: Untangling the Estimates[4], är en studie som fokuserar på att få fram medelvärdet på elintensiteten för överföring av data över internet. De beskriver att elintensitet brukar mätas i kilowattimmar per Gigabyte (kWh/GB). Studien analyserar tidigare studier inom ämnet och försöker få fram det mest trovärdiga medelvärdet. Enligt denna studie så bör vi vara nere på under 0.01 kWh/GB år 2020. Däremot är inte elintensiteten för mobildata inräknad i dessa studier. Enligt studien Evaluating the Energy Consumption of Mobile Data Transfer[19], så är medelvärdet för 3G mobile broadband access network 2.9 kWh/GB och 37 kWh/GB för 2G. Studien, som är från 2018, säger att baserat på deras ekvation kan elintensiteten för mobildata vara under 0.1 kWh/GB år 2020 i Finland.

Vid en google-sökning på ‘c02 emissions web’ hittades ett blogginlägg av utvecklarer Danny Van Kooten, där han undersöker möjligheten kring att minska sitt koldioxidfotavtryck genom att reducera storleken på sina webbplatser. Enligt honom så lyckades han med detta genom att ta bort 20kb från sitt Wordpress-plugin: “*Just last week I reduced global emissions by an estimated 59.000 kg CO2 per month by removing a 20 kB JavaScript dependency in Mailchimp for WordPress. There’s no way I can have that kind of effect in other areas of my life.*”[25]

5.1 JSON-dokument

Kunden använder sitt REST-API för att hämta data till sina produkter. Detta sker varje gång en användare går in på en specifik produkt på kundens e-handel. I bilaga A på sida 41 är hela JSON-dokumentet som returneras vid hämtning av en produkt med produktnummer: ‘145451002’.

5.1.1 Klienten

Samma produkt som hämtades i listing A.1, visas på webbplatsen i figur 5.1 på sida 20. Tabell 5.1 på sida 25 - 26, går igenom alla fält som kommer från REST-API:t och kollar vilka som används av klienten eller inte. Tabellen har tre olika kolumner, på vänster sida är kolumnen för **Fält**, som representerar ett fält i JSON-dokumentet. Kolumnen i mitten, **Används**, kommer få värdet *Ja* eller *Nej* beroende på om fältet används av klienten. **Nummer/Syfte** som är den sista kolumnen ger en förklaring till var fältet används på klienten, om det står en siffra är den kopplad till samma siffra på figur 5.1. Denna sammanställning har kunden hjälpt till att skapa.

Ett JSON-dokument har en hierarkisk struktur. Under kolumnen **Fält** i tabellen 5.1, finns det rader där det står till exempel ‘**brands.campaign**’. Det innebär den översta nivån i hierarkin (brands) och sedan nivån under brands i hierarkin (campaign). Detta har delats upp eftersom att fälten på nivå två i hierarkin kan ha olika syften i klienten. Det är även så att vissa fält i nivå två inte används. I GraphQL går det att styra vilka fält i de olika hierarki-nivåerna som skall hämtas. Detta innebär att det inte bara är den översta nivån i JSON-dokumentets hierarki som kommer räknas i antal fält i denna sammanställning.

HEM > KLÄDER > TRÖJOR > COLLEGETRÖJOR > FI W ZIP HUVTRÖJA 1

2

3

FI W Zip huvtröja 4
Peak Performance | Dam | Rosa

5 6 7 8 **799 kr** 1199 kr 9 10 **-33%**

11

THINK PINK

Välj storlek

XS	S	M	L
XL			

12

Medlemspris!
Logga in för att få medlemspris. Logga in 13

Lägg i varukorg 14

Boka & hämta i butik 15

Se butikssald 16 Storleksguide

- ✓ Fri frakt & fria returer till butik
- ✓ Hemleverans 79kr
- ✓ 2-6 helgfria dagars leveranstid
- ✓ Öppet köp i 365 dagar

Betala snabbt och smidigt med Klarna **Klarna.**

Leveranssätt 17

Produktinformation 18

Otröligt mjuk jersey med borstad baksida som kombineras med en avslappnad siluett i oversizemodell ger oss den perfekta huvtröjan för en modern livsstil.

- Mjukt material 19
- Avslappnad siluett

Kategorier: Dam, Sportswear, Peak Performance, Kläder, Tröjor, Collegetröjor 20

Artikeldetaljer > 21

Andra köpte också

49 kr Shorty Running ankelstr...

350 kr Amplified TR huvtröja

49 kr Shorty Running ankelstr...

99 kr Smellwell doftpåse

79 kr No show ankelsocka 3-p...

69 kr Shaftless ankelsocka 3-...

Relaterade produkter

2

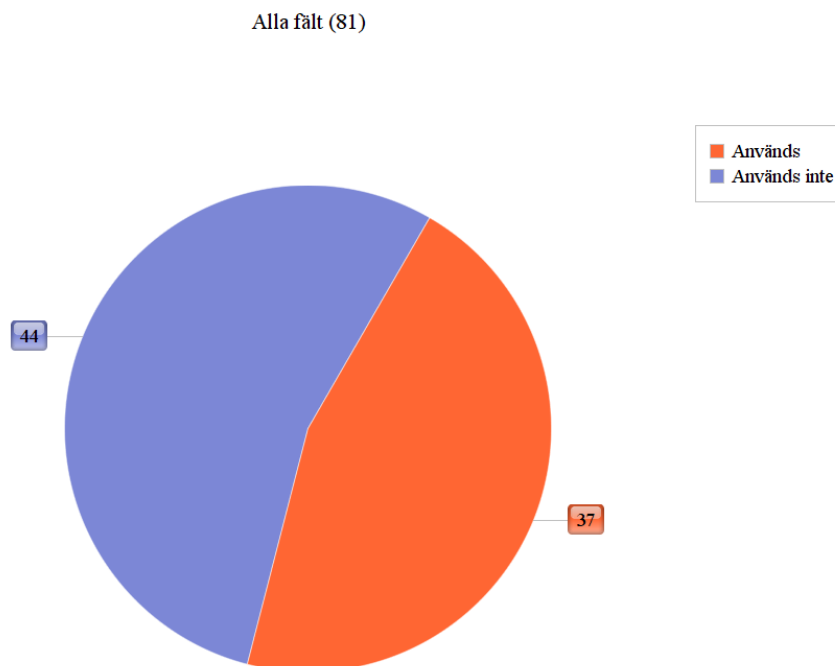
Figur 5.1: Visning av en produkt på webbplatsen.

Fält	Används	Nummer/Syfte
_indextime	Nej	-
article_urlslug	Ja	Slug
articlenotsearchable	Nej	-
brand	Nej	-
brandTaxonomy	Nej	-
brand_urlslug	Nej	-
brandimage	Nej	-
brandimageurl	Nej	-
brands.brand	Ja	5
brands.brand_urlslug	Ja	Slug
brands.brandimage	Ja	2
brands.brandimageurl	Ja	2
campaign.campaigndescription	Nej	-
campaign.campaignid	Nej	-
campaign.campaigntag	Nej	-
categories	Ja	1
color	Nej	-
colordescription	Ja	11
colordescription_urlslug	Ja	11
colors.colordescription	Ja	11
colors.colordescription_urlslug	Ja	11
colors.hex	Nej	-
colors.itemimage	Ja	11
colors.name	Ja	7
colors.name_urlslug	Nej	-
commercialname	Ja	4
commercialname_urlslug	Ja	Slug
deliverytime	Nej	-
description	Ja	18
disableclickandcollect	Ja	15
documenttype	Nej	-
dynamic_facets.filterkey	Nej	-
dynamic_facets.filtervalue	Nej	-
googlecategory	Nej	-
hidearticleweb	Nej	-
hidetargetaudience	Nej	-
intersportexclusive	Nej	-
itemcategory	Nej	-
itemimages	Nej	-
itemimages_png	Nej	-
itemname	Ja	21
itemnumber	Ja	21
itemproducttype	Ja	21

itempublishweb	Nej	-
itempublishweb__lastmodified	Nej	-
itemtype	Ja	Relaterade produkter
itempublishwebarticle	Nej	-
itemurl	Nej	-
maingroup	Nej	-
media (several fields)	Ja	3
onearticleinlistings	Ja	Logik för bundles
onearticleinlistingspresent	Ja	Logik för bundles
popularity	Nej	-
price.price	Ja	8
price.pricetype	Ja	10
price.regularprice	Ja	9
productname	Nej	-
productname__urlslug	Nej	-
productnumber	Ja	17
productsizerange	Ja	12
producttype	Nej	-
salable	Nej	-
salabletext	Nej	-
saleable	Ja	14
saleabletext	Ja	14
salescontrol.salescontroldescription	Ja	10 + 13
salescontrol.salescontrolype	Ja	10 + 13
season	Nej	-
shortdescription	Nej	-
sites	Nej	-
sizes (several fields)	Ja	12
sizes.saldo	Ja	15 + 16
sort__commercialname	Nej	-
sportTaxonomy	Nej	-
sports	Nej	-
sports__urlslug	Nej	-
supplieritemnumber	Ja	17
targetGroupTaxonomy	Nej	-
targetaudience	Ja	6
usp	Ja	19
valid	Nej	-

Tabell 5.1: Sammanställning av fälten som används av klienten.

Grafen i figur 5.2 visar vilka fält som används och inte används. Det är totalt 81 fält och bara 37 av dessa fält används. Det är möjligt att plocka bort 44 fält och

**Figur 5.2:** Statistik från tabellen

jämföra storleken på det returnerade JSON-dokumentet. Det innebär en minskning på 54%.

5.1.2 Storlek och nedladdningstid

Som verktyg för att mäta storlek och responstid användes programmet Postman. I Postman skickas en GET-request till slutpunkten för att hämta produktdetaljer. Storleken och nedladdningstiden på det som skickas tillbaka från Postman visas i figur 5.3.

Status: 200 OK Time: 146ms Size: 7.32 KB

Figur 5.3: Nedladdningstid och storlek från REST-API:t

I Postman skickas även en förfrågan till den nyskapade GraphQL-servern. Där bara de rader som klienten behöver hämtas från servern. Listing 5.1 visar hur en sådan fråga ser ut.

```
1 {  
2   product(id: "145451002") {  
3     article_urlslug  
4     brands {
```

```
5     brandimage
6     brandimageurl
7     brand_urlslug
8     brand
9   }
10  categories
11  colordescription
12  colordescription_urlslug
13  colors {
14    itemimage
15    colordescription
16    colordescription_urlslug
17    name
18  }
19  commercialname
20  commercialname_urlslug
21  description
22  disableclickandcollect
23  itemname
24  itemnumber
25  itemtype: itemtypeintersport
26  media
27  price {
28    price
29    pricetype
30    regularprice
31  }
32  productsizerange
33  saleable
34  saleabletext
35  salescontrol {
36    salescontroldescription
37    salescontroltype
38  }
39  sizes {
40    itemskuean
41    itemskunumber
42    itemskusizeid
43    itemskusizename
44    itemskusizename_urlslug
45    saldo {
46      quantityAvailable
47      store
48    }
49  }
50  supplieritemnumber
51  targetaudience
52  usp
53 }
54 }
```

Listing 5.1: GraphQL-query för att hämta produktinformation

I figur 5.4 visas storleken och tiden för requesten. Det tar 88ms att få JSON-dokumentet och requesten har en storlek på 2.9 KB. Detta visar klart och tydligt att det är möjligt att minska storleken på det JSON-dokument som klienten efterfrågar

vid användning av GraphQL. I detta fall minskar response-storleken med 4,42 kB (7,32 kB - 2,9 kB). Nedladdningstiden är även kortare vid användning av GraphQL, det går 58 ms snabbare.

Status: 200 OK Time: 88ms Size: 2.9 KB

Figur 5.4: Nedladdningstid och storlek från GraphQL-API:t

5.2 Hållbarhet

De studier som finns inom ämnet har olika medelvärden för kostnaden av dataöverföring över internet. Det kan skilja mycket på typen av nätverk som används[25]. Räckvidden är från 37 kWh/GB för 2G-nätet[19] till 0,01 kWh/GB för det fasta bredbandsnätet[4]. 3G-nätet används till en viss del i Sverige och den har ett värde på 2.9 kWh/GB enligt Energy Consumption of Mobile Data Transfer[19], samma studie säger att strömförbrukningen bör vara så låg som 0,1 kWh/GB år 2020 (i Finland). Ett värde behöver sättas och 0,50 kWh per GB används i denna studie, eftersom det är över medelvärdet för det fasta bredbandsnätet samtidigt som det är under 3G-nätets medelvärde.

Vi har ett medelvärde på 0,50 kWh per GB. För att omvandla detta till kostnaden per kB istället, som blir enklare att arbeta med, behöver detta divideras med en miljon:

$$\frac{0,5kWh}{1000000} = 0,0000005kWh/kB \quad (5.1)$$

Intersports webbplats hade **9 120 492** unika sidvisningar per månad (2019). Här används de unika sidvisningarna istället för de totala sidvisningarna, eftersom förfrågningarna till REST API:t annars kan vara cachade i webbläsaren. Som utgångspunkt för denna studie, räknas varje sidvisning som en produktvisning - en förfrågan till REST-API:t för att hämta produktdetaljer. För att räkna ut hur mycket elförbrukningen minskar för varje kB som tas bort från det returnerade JSON-dokumentet, behövs antal unika sidvisningar multipliceras med medelvärdet för elförbrukningen per kB. Sedan behövs det resultatet multipliceras med den minskade storleken på response-objektet vid användning av GraphQL (4,42 kB). Uträkningen som visas i 5.2, blir att vid användning av GraphQL sparar kunden 20,15 kWh per månad.

$$(0,0000005kWh \times 9120492) \times 4,42 = 20,15 kWh \quad (5.2)$$

5.2.1 Vad innebär det i minskad koldioxidutsläpp?

Sveriges utsläppsintensitet för koldioxid ligger på 13,3g per kWh, enligt senaste mätningen från European Environment Agency[1]. Nästa steg blir att räkna ut hur mycket Intersport kan dra ner på världens koldioxidutsläpp per månad genom att använda sig av GraphQL. Vi vet att Intersport sparar 20,15 kWh per månad och den

siffran skall multipliceras med Sveriges utsläppsintensitet för koldioxid. I uträkning 5.3 ser vi resultatet.

$$20,15kWh \times 13,3g = \mathbf{268\ g} \quad (5.3)$$

Intersport kan därmed minska på sitt koldioxidavtryck med 268 g per månad.

Vad innebär detta per år?

Under år 2019 hade Intersports webbplats totalt 109 445 901 unika sidvisningar. För att räkna ut vad det innebär i energiförbrukning per kB skall de unika sidvisningarna multipliceras med medelvärdet för kWh per kB. Sedan behöver det resultatet även multipliceras med den storlek som JSON-dokumentet minskade vid användning av GraphQL. 5.4 visar uträkningen.

$$(0,0000005 \times 109445901) \times 4,42 = \mathbf{241,88\ kWh} \quad (5.4)$$

Vid användning av GraphQL kan Intersport minska sin elförbrukning med 241,88 kWh. Nästa fråga är hur mycket detta innebär i minskad koldioxidutsläpp. Som tidigare räknas detta ut genom att multiplicera den sparade elförbrukningen med Sveriges utsläppsintensitet för koldioxid. Uträkningen och resultatet kan ses i 5.5.

$$241,88 * 13,3 = \mathbf{3217\ g} \quad (5.5)$$

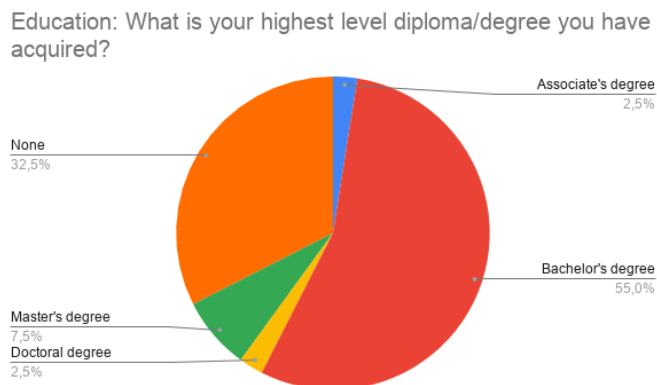
I minskad koldioxidutsläpp innebär detta 3217 g. På ett år kan Intersport reducera koldioxidutsläppen med över 3,2 kg. Dessa resultat visar att det är möjligt att minska världens koldioxidutsläpp genom att använda GraphQL istället för REST.

5.3 Enkätundersökning

Enkäten har skickas ut till utvecklare på olika sociala medier (Reddit, Facebook, Gitter). Målet har varit att försöka få så många utvecklare som möjligt att svara på enkäten. Helst olika typer av utvecklare, eftersom målet är att se vilka som känner till GraphQL och inte, kan det eventuellt baseras på hur länge man har programmerat? Är det till största del nya utvecklare som använder GraphQL?

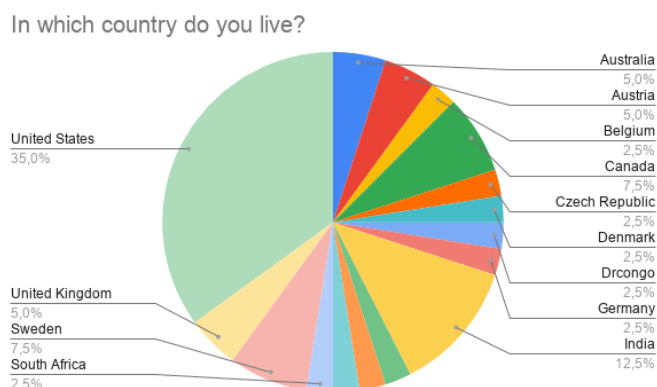
5.3.1 Svarsresultat

Totalt svarade 40 utvecklare på enkäten. Det är främst på undersidor för utvecklare på Reddit som data har samlats in. Figur 5.5 visar vad de 40 deltagarna har för



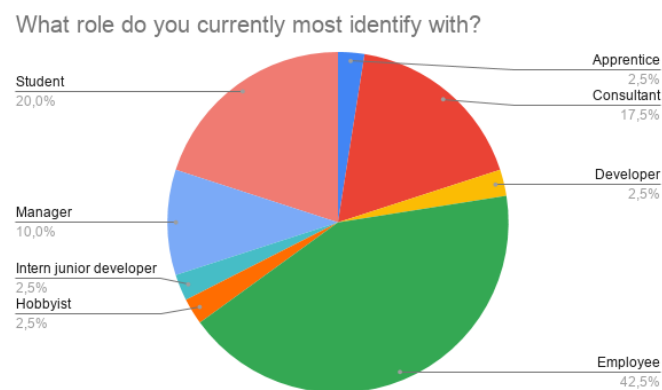
Figur 5.5: Vilken är den högsta högskoleutbildning du har?

högsta utbildning. Majoriteten (55%) av deltagarna har en Bachelor's degree, eller kandidatexamen som det heter på svenska. Diagrammet visar att en stor del av deltagarna (32,5%) inte har någon högskoleutbildning.



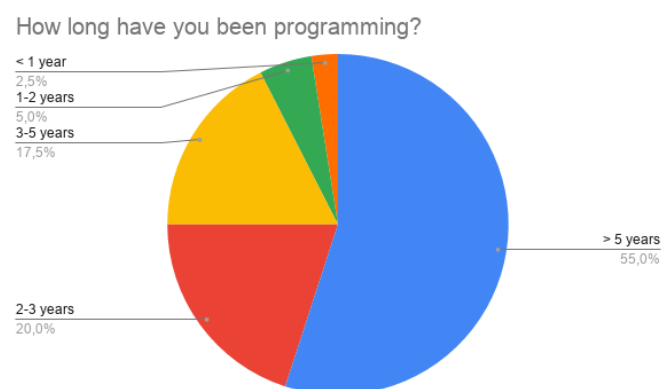
Figur 5.6: I vilket land bor du?

Figur 5.6 visar var utvecklarna som deltagit i enkäten bor i världen. Fjorton av deltagarna som är den största gruppen bor i USA (35%). Den nästa största gruppen är Indien (12,5%) med fem deltagare. På delad tredje plats finns Sverige (7,5%) med tre personer och Canada (7,5%) med lika många. I övrigt är den en stor spridning med personer från stora delar av världen.



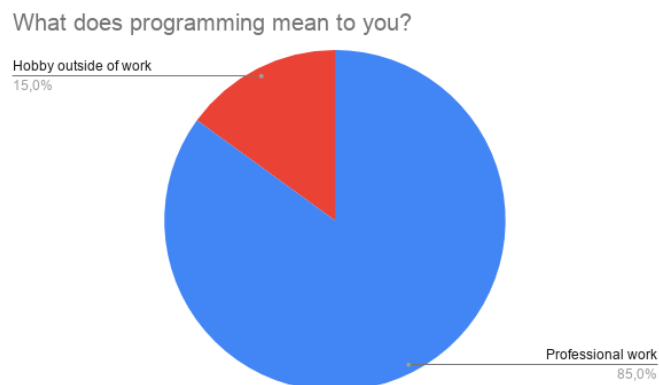
Figur 5.7: Vilken roll passar bäst in på dig?

Denna fråga handlar om vilken roll deltagaren själv känner stämmer bäst in. En stor del av deltagarna är anställda (42,5%), det är även en relativt stor grupp som är studenter (20%). Figur 5.7 visar hela resultatet.



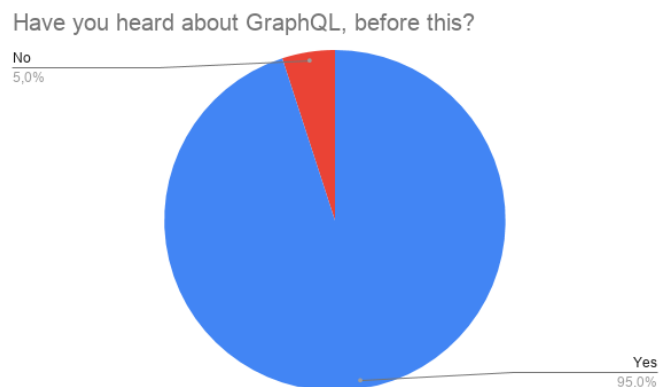
Figur 5.8: Hur länge har du programmerat?

Utvecklarna fick svara på frågan om hur länge de har programmerat, där det tydligt finns en majoritet som har programmerat i över 5 år (55%). Den nästa största gruppen är 2-3 år (20%) följt av 3 - 5 år (17,5%). Diagrammet i figur 5.8 visar hela resultatet.



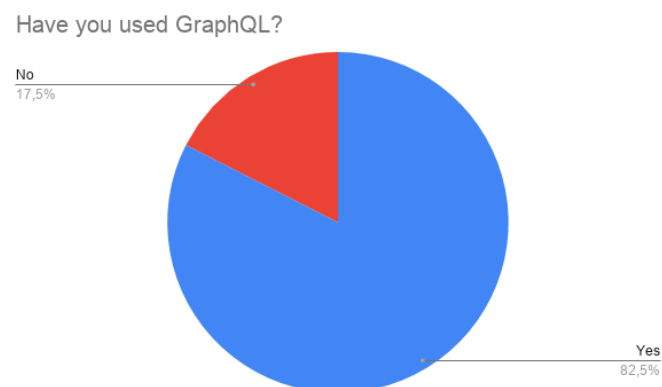
Figur 5.9: Vad betyder programmering för dig?

Nästa fråga handlar om vad programmering innebär för utvecklarna. Om det är deras professionella yrke eller om det är en hobby de sysslar med på fritiden. 85% av deltagarna använder programmering i sitt arbetsliv (figur 5.9). Samtidigt som 15% av utvecklarna som deltog i undersökningen, har programmering som en hobby.

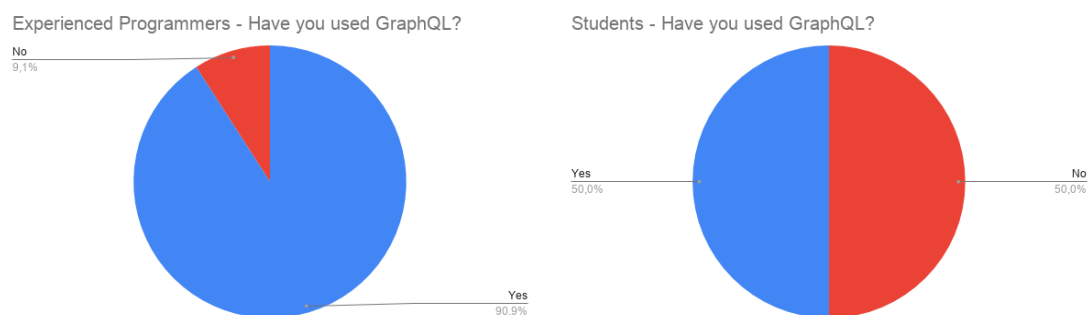


Figur 5.10: Har du hört talas om GraphQL tidigare?

Här börjar frågorna om GraphQL och REST. Figur 5.10 visar resultatet på frågan om utvecklaren har hört talas om GraphQL tidigare. Som diagrammet visar har nästan alla (95%) deltagare hört talas om GraphQL innan de gjorde enkätundersökningen. Det är endast två av deltagarna som aldrig hört talas om GraphQL, enligt frågan om deras roll är den ena av dem student och den andra lärling. Studenten som är från Sverige har programmerat i 2 - 3 år. Lärlingen som är från Skottland har programmerat i 1 - 2 år.

**Figur 5.11:** Har du använt GraphQL?

Som fortsättning på den föregående frågan, undersöks hur många av deltagarna som har använt sig av GraphQL tidigare. Figur 5.11 visar att det är en stor grupp (82,5%) som har gjort det. Majoriteten av de som inte använt sig av GraphQL är studenter (4 st). De övriga som inte använt GraphQL är en lärling, en konsult och en anställd. Konsulten har programmerat i över 5 år, likaså har en av studenterna gjort, de övriga har programmerat i mindre än 5 år.

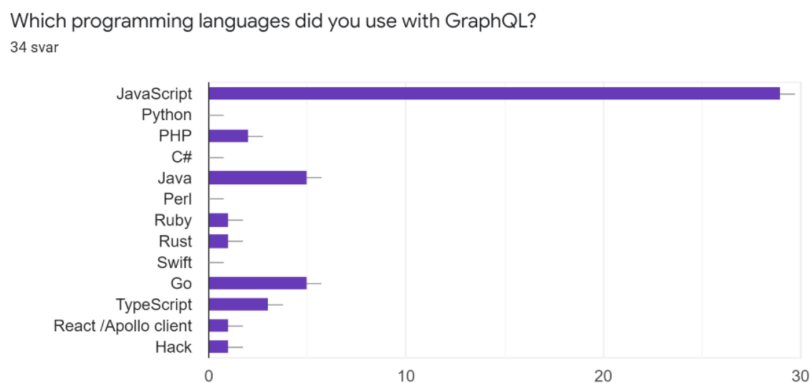
**Figur 5.12:** Erfarna utvecklare och studenter

Figur 5.12 visar samma fråga som föregående. I det vänstra diagrammet har alla deltagare som programmerat i minst 5 år valts ut. I det högra diagrammet har alla studenter valts ut. Antal som har använt GraphQL av de erfarna programmerarna blev 90,9% och för studenterna blev det 50%.



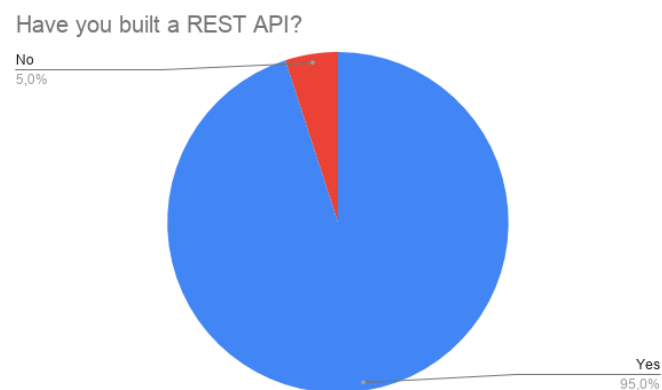
Figur 5.13: Vilken typ av applikation använde du GraphQL med?

De deltagare som använt GraphQL, fick svara på vilken typ av applikation de använt GraphQL till (figur 5.13). Det svarsalternativ som har fått det största svarsantalet är **Native App** (34,4%), som innebär ett mjukvaruprogram utvecklat för en specifik plattform, oftast handlar det om en mobilapplikation. Näst flest svar fick **Blog** (28,1%), följt av **E-commerce** (25%) och **Social network** (21,9%).



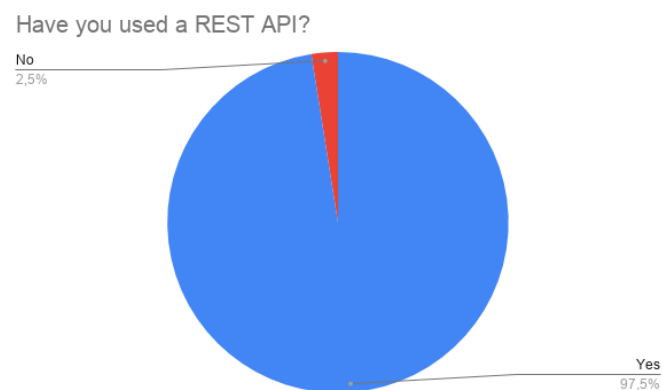
Figur 5.14: Vilka programmeringsspråk använde du ihop med GraphQL?

Deltagarna som har använt GraphQL fick frågan om vilka programmeringsspråk de använde ihop med GraphQL. Figur 5.14 visar att de allra flesta har använt GraphQL tillsammans med JavaScript (85,3%). På andra plats hamnade Java (14,7%) och Go (14,7%), följt av TypeScript (8,8%).



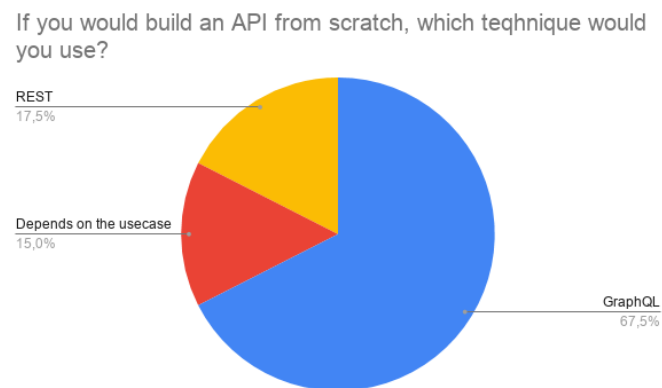
Figur 5.15: Har du byggt ett REST-API?

Nu kommer ett par frågor om REST-API. Den första tar reda på om deltagaren har utvecklat ett REST-API. Figur 5.15 visar att de flesta har gjort det (95%). De två deltagare som inte har byggt ett REST-API är en student från Serbien som har programmerat i mindre än ett år, samt en anställd med en kandidatexamen från Kanada som har programmerat i 2 - 3 år. Den anställda från Kanada svarade ja på frågan om han använt GraphQL.



Figur 5.16: Har du använt ett REST-API?

Den andra frågan om REST undersöker om utvecklaren har använt ett REST-API. Figur 5.16 visar att nästan alla har gjort det. Den enda som inte använt REST är studenten från Serbien.



Figur 5.17: Om du hade byggt ett API från grunden, vilket teknik hade du använt?

Den avslutande frågan av undersökningen handlar om vilken teknik deltagarna hade valt om de själva byggt ett API från grunden. Figur 5.17 visar en sammanställning som gjorts på svaren. Det svarsalternativ som fick flest röster blev GraphQL (67,5%) följt av REST (17,5%). Flera av deltagarna valde att välja det tredje alternativet, där de själva fick fylla i svaret. Där de skrev att det beror på användningsområdet, dessa svar räknades samman och hamnade under ‘Depends on the usecase’ i diagrammet. Exempel på vad utvecklarna svarade:

“Depends on requirements. For micro-services REST helps keep things in check, when you’ve got something that aggregates data across services GraphQL is the way to go.”,

“Graphql if I am the main consumer. REST if I have to allow outsiders to consume it. Swagger and the like are industry standards while graphql is still in its infancy.”,

“REST if it is a simple API, GraphQL if it is a more complicated API.”,

“Depends on context, massively.”,

“It depends on the application. For some, I think a few REST endpoints is enough. For more data, GraphQL is better.”

6.1 Forskningsfråga 1

Vid användning av GraphQL, hur många rader färre JSON-data får klienten tillbaka vid en API-förfrågan? Om en rad innebär ett fält blev svaret 44 rader färre JSON-data, eller en minskning på 54%. Detta visar att det är möjligt att minska antal rader JSON-data som klienten får tillbaka vid användning av GraphQL. Antal rader som det går att minska kan skilja mycket beroende på vilken typ av applikation det är som används. Om Intersport hade haft en mobilapplikation där det krävdes ännu färre rader JSON-data, som det ofta gör i mobilappar, hade siffrorna med stor sannolikhet sett ännu större ut.

I studie 1 från från litteraturgenomgången blev resultatet på samma forskningsfråga en reducering på hela 94%. Den stora skillnaden mellan dessa två studier är typen av applikationer som har testats. I studie 1 användes applikationer som hämtade data från GitHub's egna REST-API. GitHub's API är stort och det är något som redan är implementerat. Utvecklarna av dessa applikationer har själva inte skapat GitHub's REST-API och har därför inte kunna styra vilken data som skall returneras. Ur den synvinkeln är resultatet från denna studie, med en reducering på 54%, mer realistiskt. Eftersom Intersports REST-API har skapats i syfte för att visa upp produkter till deras klient.

6.2 Forskningsfråga 2

Vid användning av GraphQL, vad är den minskade storleken på det returnerade JSON-dokumentet? Den minskade storleken blev 4,42 kB, vilket är en minskad storlek på 60%. I de två studier som gicks igenom i litteraturgenomgången, har den minskade storleken varit den största fördelen med GraphQL, i studien 'Experiences on Migrating RESTful Web Services to GraphQL' reducerades storleken i genomsnitt med 99% [5]. Resultatet i denna studie med en minskning på 60% vid användning av GraphQL är mer realistiskt eftersom det är kopplat till en riktig e-handel i produktion. Detta är ett resultat som med stor sannolikhet kan återspeglas i andra företag som använder sig av REST-API:er. Om majoriten skulle gå över till tekniker som GraphQL skulle det bidra till en stor minskning i strömförbrukning och minskat koldioxidutsläpp. Enligt studien "A GraphQL approach to Healthcare Information Exchange with HL7 FHIR" bidrar det även till en minskad responstid och minskad driftkostnad, samt en förbättring av prestandan[17].

I jämförelsen som gjordes mellan REST och GraphQL, blev även nedladdningstiden mindre. Det gick 58 ms snabbare att få datan via GraphQL jämfört med REST. Det finns en anledning till att detta inte var med som en forskningsfråga. Dessa jämförelser går inte att förlita sig på på grund av att GraphQL körs på en lokal server, samtidigt som REST-API:t ligger på ett helt annan server, de har olika förutsättningar vilket kan påverka svarstiden.

6.3 Forskningsfråga 3

Går det att minska världens koldioxidutsläpp genom att byta till GraphQL? Ja det går att minska världens koldioxidutsläpp genom att byta till GraphQL. Däremot är det nästan omöjligt att säga med hur mycket. Det är många faktorer som spelar in och det är svårt att hitta ett bra medelvärde för kostnaden av dataöverföring över internet. I denna studie valdes ett medelvärde på 0,50 kWh per GB. Som nämntes i resultatet så används 3G-nät fortfarande till en viss del i Sverige och den har en förbrukning på 2.9 kWh/GB. Däremot så är hela världen på väg mot betydligt lägre siffror, kollar vi på Evaluating the Energy Consumption of Mobile Data Transfer [19], är strömförbrukningen för mobildata under 0.1 kWh/GB år 2020 i Finland. Hur mycket Sverige skiljer sig från Finland är svårt att säga, inga studier har hittats på detta. Även det fasta bredbandsnätet bör ha siffror så låga som 0.01 kWh/GB år 2020, enligt studien Electricity Intensity of Internet Data Transmission: Untangling the Estimates [4]. Ett antagande kan göras att siffrorna på strömförbrukningen av dataöverföring bör vara lägre än 0,50 kWh per GB.

Till frågan om vad Sveriges CO₂-utsläppsintensitet ligger på, så användes European Environment Agency [1]. Där de har räknat ut koldioxidutsläppet från den offentliga elproduktionen. Dessa siffror är hämtade från 2016 och är den senaste datan som hittades. Hur mycket den datan stämmer överens med dagens värden är okänt, därför är det osäkert hur trovärdiga siffrorna är. Syftet med detta arbete var att se om det var möjligt att minska sitt koldioxidsavtryck genom att använda GraphQL, och det har visat sig vara möjligt.

6.3.1 Alternativt sätt att räkna

CO₂-utsläppsintensitet för elproduktionen inom EU ligger på 295,5g per kWh[1]. Ifall Intersport skulle haft sin server någon annanstans inom EU och eventuellt haft kunder från hela EU hade resultatet sett annorlunda ut. Ett räkneexempel finns i 6.1.

$$241,88 * 295,5 = \mathbf{71\ 475,54\ g} \quad (6.1)$$

Det minskade koldioxidutsläppet blir alltså i detta fall istället på drygt 71 kg per år. Vilket visar att det kan skilja mycket beroende på var i världen man bor.

6.4 Forskningsfråga 4

Hur populärt är GraphQL bland utvecklare? 40 utvecklare svarade på enkäten. Fler deltagare hade varit önskvärt för ett tydligare resultat. De flesta deltagarna hade

hört talas om GraphQL tidigare och en stor majoritet hade även använt GraphQL tidigare. Här visade det sig att det var de mest erfarna deltagarna som främst hade använt GraphQL. Användningsområdet enligt enkäten var främst för nativa-appar. Vilket låter logiskt, eftersom det oftast är då man vill kunna välja vilken data som hämtas av klienten. Eftersom overfetching är så pass vanligt förekommande fenomen, främst inom mobila-appar. Det vanligaste programmeringsspråket för GraphQL är JavaScript, vilket även det är logiskt, då det finns oerhört många guider, videos och blogginlägg om hur implementationen av en GraphQL-server görs i JavaScript. Industristandarden för en GraphQL-implementering är Apollo, som är skapat i JavaScript [2].

97,5% hade använt ett REST-API jämfört med GraphQL där det var 82,5%. Vilket tyder på att REST-API är det vanligaste sättet och förmodligen även det sättet utvecklare först lär sig att använda.

Majoriteten av de som svarade hade valt att skapa ett API i GraphQL istället för REST (eller något annat sätt). Detta tyder på att de flesta kan se fördelarna med GraphQL. Även om det ibland räcker med REST, som en deltagare sa: “*REST if it is a simple API, GraphQL if it is a more complicated API.*” GraphQL verkar för de flesta som kommenterade vara det bästa sättet för mycket data.

Kapitel 7

Slutsatser och Framtida arbete

7.1 Slutsatser

Vid jämförelse med REST innebär användning av GraphQL en stor reduktion i antal rader och storlek i det returnerade JSON-dokument. REST-API:er är ofta designade att förse klienten med all tillgänglig data, vilket innebär att överflödigt data skickas så fort klienten inte använder all data som kommer från servern. Detta blir som tydligast i mobil-applikationer där det som regel krävs mindre data från klienten än till exempel en desktop-applikation. GraphQL öppnar upp möjligheten att från klienten styra vilken data som skall hämtas från servern, vilket är den stora fördelen. GraphQL bör användas när ett API består av mycket data, det blir snabbare, drar mindre ström och sparar på miljön.

En GraphQL-server implementerades och användes för att jämföra med kundens egna REST-API. Resultatet av jämförelsen visade att användning av GraphQL minskar antal JSON-fält som returneras från servern med 54% och minskad storlek med 60%.

Studien visar att vid användning av GraphQL går det att minska på världens koldioxidutsläpp. Mindre data som skickas från server till klient innebär reducerad strömförbrukning. För kunden innebär det en reduktion med 20,15 kWh per månad. Den minskade strömförbrukningen går sedan att räknas om till minskad koldioxidutsläpp. Sveriges koldioxidutsläpp från den offentliga elproduktionen ligger på 13,3g per kWh, kunden kan minska på sitt koldioxidavtryck med 268 g per månad eller drygt 3,2 kg per år.

Studien utförde en enkätundersökning där 40 utvecklare runt om i världen svarade på frågor kring GraphQL och REST. Enkätundersökningen visar att GraphQL är populärt, nästan alla enkätdeltagare hade hört talas om GraphQL och använt det. REST hade något högre siffror vid användning av tekniken. Vid frågan om vilken teknik som utvecklarna föredrog fick GraphQL en stor majoritet.

7.2 Framtida arbete

Fokus på detta arbete har varit om det är möjligt att minska storleken av det returnerade dokumentet, samt vad det innebär i minskad strömförbrukning och för miljön. Som fortsättning på arbetet bör det även göras ordentliga jämförelser på nedladdningstid, genom att ladda upp ett GraphQL-API och REST-API på samma server för att skapa lika förutsättningar.

I framtida arbete borde det även fokuseras på under-fetching, vilket händer när en slutpunkt på ett REST-API inte har all data som krävs och ytterligare en förfrågan behöver göras till en annan slutpunkt. I GraphQL går det att att lösa i samma förfrågan, genom den hierarkiska datamodellen som tillåter klienter att hämta data från flera källor i en enda förfrågan. Detta bör ha stor påverkan på nedladdningstiden samt strömförbrukningen.

Ett REST-API med flera slutpunkter bör jämföras med ett GraphQL-API som hämtar samma data. I detta arbete har den implementerade GraphQL-servern bara kunnat hämta data för en produkt baserat på id-nummer, denna server borde utökas för att hämta mer data som möjliggör tydligare jämförelser.

Koldioxidutsläppet kan skilja mycket beroende på var i världen servern är, därför bör det även göras liknande studier i andra delar av världen.

Referenser

- [1] European Environment Agency. Overview of electricity production and use in europe. <https://www.eea.europa.eu/data-and-maps/indicators/overview-of-the-electricity-production-2/assessment-4>, 2019. Accessed 2020-04-16.
- [2] Apollo. GraphQL playground. <https://www.apollographql.com/docs/apollo-server/testing/graphql-playground/>, 2020. Accessed 2020-02-14.
- [3] Apollo. Introduction. <https://www.apollographql.com/docs/apollo-server/>, 2020. Accessed 2020-02-14.
- [4] Joshua Aslan, Kieren Mayers, Jonathan G. Koomey, and Chris France. Electricity intensity of internet data transmission: Untangling the estimates. *Journal of Industrial Ecology*, 22(4):785–798, 2018.
- [5] Gleison Brito, Thais Mombach, and Marco T. Valente. Migrating to graphql: A practical assessment. pages 140–150. IEEE, 2019.
- [6] Lee Byron. GraphQL: A data query language. <https://graphql.org/>, 2020. Accessed 2020-02-14.
- [7] World Wide Web Consortium. 3.1.3 relationship to the world wide web and rest architectures. <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest>, 2004. Accessed 2020-05-04.
- [8] Elasticsearch. Elasticsearch node.js client. <https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/current/index.html>, 2020. Accessed 2020-04-06.
- [9] Elasticsearch. What is elasticsearch. <https://www.elastic.co/what-is/elasticsearch>, 2020. Accessed 2020-02-21.
- [10] Express.js. Express. <https://expressjs.com/>, 2020. Accessed 2020-02-14.
- [11] Roy Thomas Fielding. Representational state transfer. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. Accessed 2020-05-28.
- [12] Sharon Fisher. Os/2 ee to get 3270 interface early. https://books.google.se/books?id=YToEAAAAMBAJ&pg=PA6&dq=application+programming+interface&redir_esc=y&hl=en#v=onepage&q=application%20programming%20interface&f=false, 1989. Accessed 2020-05-04.

- [13] GatsbyJS. Fast in every way that matters. <https://www.gatsbyjs.org/>, 2020. Accessed 2020-06-06.
- [14] GraphQL. GraphQL. <https://engineering.fb.com/core-data/graphql-a-data-query-language/>, 2020. Accessed 2020-02-14.
- [15] Suhas Karanth. elastic-builder. <https://www.npmjs.com/package/elastic-builder>, 2020. Accessed 2020-04-06.
- [16] Xiaoqiang Ma, Peramanathan Sathyamoorthy, Edith C.-H Ngai, Xiping Hu, and Victor C. M. Leung. Profiling energy efficiency and data communications for mobile internet of things. In *Web Engineering*. Hindawi, 2017.
- [17] Suresh Kumar Mukhiya, Fazle Rabbi, Violet Ka I Pun, Adrian Rutle, and Yngve Lamo. A graphql approach to healthcare information exchange with hl7 fhir. *Procedia Computer Science*, 160:338–345, 2019.
- [18] Node.js. Introduction. <https://nodejs.org/en/>, 2020. Accessed 2020-02-14.
- [19] Hanna Pihkola, Mikko Hongisto, Olli Apilo, and Mika Lasanen. Evaluating the energy consumption of mobile data transfer—from technology development to consumer behaviour and life cycle thinking. *Sustainability*, 10(7):2494, 2018.
- [20] Postman. Postman api client. <https://www.postman.com/product/api-client>, 2020. Accessed 2020-04-16.
- [21] Prisma. GraphQL playground. <https://github.com/prisma-labs/graphql-playground>, 2020. Accessed 2020-04-06.
- [22] ReadMe. The history of rest apis. <https://blog.readme.com/the-history-of-rest-apis/>, 2016. Accessed 2020-05-29.
- [23] Johan Sjöberg. Findwise i3. <https://findwise.com/sv/teknologi/findwise-i3>, 2020. Accessed 2020-05-28.
- [24] TypeGraphQL. Introduction. [urlhttps://typegraphql.com/docs/introduction.html](https://typegraphql.com/docs/introduction.html), 2020. Accessed 2020-04-09.
- [25] Danny van Kooten. Co2 emissions on the web. https://dannyvankooten.com/website-carbon-emissions/?fbclid=IwAR3bSi6BVR5fhNAoJG5lDrVDMest6IUimv0Qf_dtscifZlEAYOXybgLZWuI, 2020. Accessed 2020-02-21.

A.1 JSON-dokument

```
1 {
2   "_indextime": "2020-03-25 11:18:36.600",
3   "article_urlslug": [
4     "peak-performance-fi-w-zip-huvtroja-dam",
5     "peak-performance-fi-w-zip-huvtroja"
6   ],
7   "articlenotsearchable": "false",
8   "brand": "Peak Performance",
9   "brandTaxonomy": "Peak Performance/Kläder/Tröjor/Collegetröjor",
10  "brand_urlslug": "peak-performance",
11  "brandimage": "peak performance.png",
12  "brandimageurl": "/varumarken/peak performance",
13  "brands": [
14    {
15      "brand": "Peak Performance",
16      "brand_urlslug": "peak-performance",
17      "brandimage": "peak performance.png",
18      "brandimageurl": "/varumarken/peak performance"
19    }
20  ],
21  "campaign": [
22    {
23      "campaigndescription": "Club Sportswear Textil",
24      "campaignid": "1892-00",
25      "campaigntag": "1892"
26    }
27  ],
28  "categories": ["Kläder/Tröjor/Collegetröjor"],
29  "color": [
30    {
31      "ColorCode": "FFCCCC",
32      "ColorName": "Rosa"
33    }
34  ],
35  "colordescription": "Think Pink",
36  "colordescription_urlslug": "think-pink",
37  "colors": [
38    {
39      "colordescription": "Think Pink",
40      "colordescription_urlslug": "think-pink",
41      "hex": "FFCCCC",
```

```

42     "itemimage": "145451002000_10.jpg",
43     "name": "Rosa",
44     "name_urlslug": "rosa"
45   }
46 ],
47 "commercialname": "FI W Zip huvtröja",
48 "commercialname_urlslug": "fi-w-zip-huvtroja",
49 "deliverytime": "1-2",
50 "description": "Otroligt mjuk jersey med borstad baksida som
    kombineras med en avslappnad siluett i oversizemodell ger oss
    den perfekta huvtröjan för en modern livsstil.",
51 "disableclickandcollect": "false",
52 "documenttype": "article",
53 "dynamic_facets": {
54   "filterkey": "Egenskaper",
55   "filtervalue": "Med luva"
56 },
57 "googlecategory": "212",
58 "hidearticleweb": "false",
59 "hidetargetaudience": "false",
60 "intersportexclusive": "false",
61 "itemcategory": ["Tröjor"],
62 "itemimages": [
63   "145451002000_10.jpg",
64   "145451002000_20.jpg",
65   "145451002000_30.jpg"
66 ],
67 "itemimages_png": [
68   "145451002000_10.png",
69   "145451002000_20.png",
70   "145451002000_30.png"
71 ],
72 "itemname": "PEA FI WZIP H",
73 "itemnumber": "145451002",
74 "itemproducttype": "produkt",
75 "itempublishtoweb": "true",
76 "itempublishtoweb_lastmodified": "2019-03-28 13:10:00",
77 "itemtype": ["Collegetröjor"],
78 "itemunpublishwebarticle": "false",
79 "itemurl": "1454510/02",
80 "maingroup": "Sportswear Textil",
81 "media": [
82   {
83     "description": "",
84     "id": "675269",
85     "name": "145451002000_10",
86     "order": "0",
87     "targetlink": "",
88     "type": "productimage",
89     "url": "/productimages/",
90     "videourl": ""
91   },
92   {
93     "description": "",
94     "id": "675270",
95     "name": "145451002000_20",

```



```

96     "order": "1",
97     "targetlink": "",
98     "type": "productimage",
99     "url": "/productimages/",
100    "videourl": ""
101  },
102  {
103    "description": "",
104    "id": "675271",
105    "name": "145451002000_30",
106    "order": "2",
107    "targetlink": "",
108    "type": "productimage",
109    "url": "/productimages/",
110    "videourl": ""
111  }
112 ],
113 "onearticleinlistings": "false",
114 "onearticleinlistingspresent": "false",
115 "popularity": "0.0000082040018027",
116 "price": {
117   "price": "799",
118   "pricetype": "CampaignPrice",
119   "regularprice": "1199"
120 },
121 "productname": "PEA FI WZIP H",
122 "productname_urlslug": "pea-fi-wzip-h",
123 "productnumber": "1454510",
124 "productsizerange": ["XS", "S", "M", "L", "XL"],
125 "producttype": "produkt",
126 "salable": "1",
127 "salabletext": "JA säljbar med saldo",
128 "saleable": "1",
129 "saleabletext": "JA säljbar med saldo",
130 "salescontrol": {
131   "salescontroldescription": "CLUB",
132   "salescontroltype": "CLUB"
133 },
134 "season": "SS19",
135 "shortdescription": "Otroligt mjuk jersey med borstad baksida som
    kombineras med en avslappnad siluett i oversizemodell ger oss
    den perfekta huvtröjan för en modern livsstil.",
136 "sites": "Intersport.se",
137 "sizes": [
138   {
139     "itemskuean": "5713113431825",
140     "itemskunumber": "145451002010",
141     "itemskusizeid": "XS",
142     "itemskusizename": "XS",
143     "itemskusizename_urlslug": "xs",
144     "saldo": [
145       {
146         "quantityAvailable": "4",
147         "store": "266"
148       }
149     ]
150   }
151 ]

```

```

150 },
151 {
152   "itemskuean": "5713113431832",
153   "itemskunumber": "145451002020",
154   "itemskusizeid": "S",
155   "itemskusizename": "S",
156   "itemskusizename_urlslug": "s",
157   "saldo": [
158     {
159       "quantityAvailable": "0",
160       "store": "266"
161     }
162   ]
163 },
164 {
165   "itemskuean": "5713113431849",
166   "itemskunumber": "145451002030",
167   "itemskusizeid": "M",
168   "itemskusizename": "M",
169   "itemskusizename_urlslug": "m",
170   "saldo": [
171     {
172       "quantityAvailable": "0",
173       "store": "266"
174     }
175   ]
176 },
177 {
178   "itemskuean": "5713113431856",
179   "itemskunumber": "145451002040",
180   "itemskusizeid": "L",
181   "itemskusizename": "L",
182   "itemskusizename_urlslug": "l",
183   "saldo": [
184     {
185       "quantityAvailable": "0",
186       "store": "266"
187     }
188   ]
189 },
190 {
191   "itemskuean": "5713113431863",
192   "itemskunumber": "145451002050",
193   "itemskusizeid": "XL",
194   "itemskusizename": "XL",
195   "itemskusizename_urlslug": "xl",
196   "saldo": [
197     {
198       "quantityAvailable": "0",
199       "store": "266"
200     }
201   ]
202 }
203 ],
204 "sort_commercialname": "fi w zip huvtröja",
205 "sportTaxonomy": "Sportswear/Kläder/Tröjor/Collegetröjor",

```

```
206   "sports": ["Sportswear"],
207   "sports_urlslug": "sportswear",
208   "supplieritemnumber": "G54514236",
209   "targetGroupTaxonomy": "Dam/Kläder/Tröjor/Collegetröjor",
210   "targetaudience": ["Dam"],
211   "usp": ["Mjukt material", "Avslappnad siluett"],
212   "valid": "true"
213 }
```

Listing A.1: JSON-dokument för en produkt

