Joona Väänänen

JAMSTACK – DYNAAMISESTI TOIMIVA STAATTINEN VERKKO-SIVUSTO

JAMSTACK – DYNAAMISESTI TOIMIVA STAATTINEN VERKKO-SIVUSTO

Joona Väänänen
Opinnäytetyö
Kevät 2019
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Joona Väänänen
Opinnäytetyön nimi suomeksi: JAMstack – dynaamisesti toimiva staattinen verkkosivusto
Työn ohjaaja: Lasse Haverinen
Työn valmistumislukukausi ja -vuosi: Kevät 2019
Sivumäärä: 37

Tämän opinnäytetyön tarkoituksena oli perehtyä modernin web-kehityksen JAMstack-arkkitehtuuriin ja luoda sen ideologian mukainen staattinen blogisivusto. Motiivina oli tutkia JAMstack-arkkitehtuuria web-kehityksen vaihtoehtona palvelinvetoiselle dynaamiselle ratkaisulle. Toteutuksen staattisen verkkosivuston tuli sisältää dynaamiselle verkkosivustolle tyypillisiä ominaisuuksia, kuten sisällönhallintajärjestelmä ja kommentointimahdollisuus.

Opinnäytetyössä käsiteltiin ensiksi JAMstack-arkkitehtuurin teoriaa, sen tuomia hyötyjä ja soveltuvuutta eri käyttökohteisiin. Seuraavaksi käsiteltiin blogisivuston toteutusta sekä kokonaisuutena että yksittäisesti sen komponentteja ja tekniikoita tarkastellen. Lopuksi arvioitiin blogisivuston toteutusta sekä testattiin sen suorituskykyä ja toiminnallisuutta.

Työntuloksena saavutettiin valmis julkaistu blogisivusto, joka täytti sille työn alussa määritellyt vaatimukset. Toteutuksen suorituskyky ja toiminnallisuus testattiin ja todennettiin hyväksi sekä ohjelmallisesti simuloiden että todellisessa käyttötilanteessa. Toteutuksessa onnistuttiin saavuttamaan käyttökuluiltaan ilmainen kokonaisuus. Johtopäätöksenä tehdyn tutkimus- ja kehitystyön pohjalta todettiin JAMstack-arkkitehtuurin olevan perusteltu vaihtoehto pienistä keskisuurille verkkosivustoille.

Asiasanat: ohjelmistoarkkitehtuuri, verkko-ohjelmointi, WWW-sivustot

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Software Development

---

Author: Joona Väänänen
Title of thesis: JAMstack – Dynamically Functional Static Website
Supervisor: Lasse Haverinen
Term and year when the thesis was submitted: Spring 2019
Pages: 37

The objective of this thesis was to research the modern web development architecture, JAMstack, and to create a static blog site according to its ideology. The motive was to analyse JAMstack architecture as an alternative option in web development to a server-based dynamic solution. The implemented static website was to include features typical to a dynamic website such as a content management system and an ability to comment.

In this thesis, first the theory of JAMstack architecture was addressed, along with the benefits brought by it and its applicability for various uses. Next the implementation of the blog site was analysed both as an entirety and singularly reviewing its components and techniques. In the end, the implementation of the blog site was deliberated and its performance and functionality tested.

As the result of this thesis, a complete published blog site was created that also fulfilled the requirements specified in the beginning. The performance and functionality of the implementation was tested and confirmed desirable by both programmatically simulating and in a real-world condition. The implementation was achieved to be operating cost-free. As a conclusion based on the done research and development work, JAMstack architecture was found to be an efficient option for small to medium-sized websites.

Keywords: software architecture, web development, websites

ALKULAUSE

Tämä opinnäytetyö tehtiin puhtaasti omasta mielenkiinnosta modernia web-kehitystä kohtaan, johon JAMstack tuo poikkeavan lähestymistavan. Blogisivuston taustalla toimiva GatsbyJS taas mahdollisti luontevan oppimisväylän Reactiin kuten myös GraphQL:ään. Ennen kaikkea kuitenkin tässä opinnäytetyössä luotu blogisivusto oli avovaimoni Hyerin pitkäaikainen haave, jonka myötä opinnäytetyön aiheen valinta oli viimeistään hyvin selvä. Kiitos siis hänelle ideasta ja niskaan hengittämisestä, kiitos myös opinnäytetyön ohjaajalle Lasse Haveriselle aktiivisesta tuesta ja rakentavasta palautteesta!

Oulussa 28.5.2019

Joona Väänänen

SISÄLLYS

SANASTO

API	Application Programming Interface, ohjelmointirajapinta
CDN	Content Delivery Network, sisällönjakeluverkko
CI/CD	Continuous Integration, Continuous Delivery, jatkuva integraatio, jatkuva toimitus
CMS	Content Management System, sisällönhallintajärjestelmä
JAMstack	JavaScript, APIs, Markup, modernin web-kehityksen arkkitehtuuri
PWA	Progressive Web Application, progressiivinen selainsovellus
JSON	JavaScript Object Notation, tiedostomuoto tiedonvälitykseen
JSON-LD	JavaScript Object Notation for Linked Data, JSON-tiedostomuoto linkitetyn datan tuella
JSX	JavaScript XML, syntaksilisäosa
LAMP	Linux, Apache, MySQL, PHP, ohjelmistokokoelma
Markdown	Merkintäkieli tekstin muotoiluun
MEAN	MongoDB, Express.js, AngularJS, Node.js, ohjelmistokokoelma
Open Graph	Facebookin metatagiprotokolla
SPA	Single-Page Application, yhden sivun sovellus
Twitter Cards	Twitterin metatagiprotokolla

1 JOHDANTO

Kun ollaan perustamassa verkkosivustoa, valitaan perinteisesti staattisen tai palvelinpuolen toteutukseltaan dynaamisen verkkosivuston väliltä. Siinä missä staattisessa verkkosivustossa sisältö pysyy samana, dynaamisessa se muuttuu jatkuvasti. Sisällön päivittymisen ohella verkkosivuston dynaamisuus tekee mahdolliseksi käyttäjän vuorovaikutuksen tämän kanssa. Nämä kaksi ominaisuutta ovat useimmissa käyttötapauksissa ehdottoman tarpeellisia.

On kuitenkin syitä, miksi olla valitsematta palvelintoteutukseltaan dynaamista ratkaisua. Tällainen toteutus vaatii aktiivisempaa ylläpitoa ja on alttiimpi tietoturvahyökkäyksille. Ennen kaikkea se kuitenkin edellyttää kapasiteettiinsa nähden korkeampaa suorituskykyä palvelinpuolelta, ja siten sen kustannukset ovat suuremmat.

Tässä opinnäytetyössä selvitetään, mitä on JAMstack-arkkitehtuuri ja miten sen mukaisesti on mahdollista toteuttaa staattinen verkkosivusto, joka kuitenkin täyttää dynaamisen verkkosivuston ominaispiirteet. Opinnäytetyön tavoitteena on luoda staattinen blogisivusto, joka sisältää muun muassa sisällönhallintajärjestelmän (CMS, Content Management System) blogikirjoitusten muokkaamista ja julkaisemista varten sekä blogikirjoituskohtaisen kommenttikentän lukijoiden kanssa keskustelemiseksi.

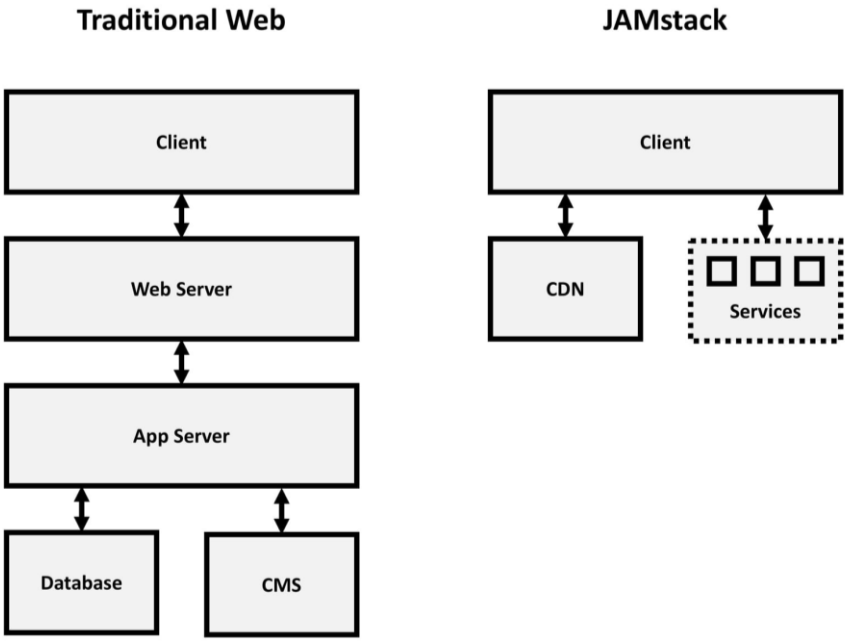
Blogisivuston kehitysympäristön tulee toimia automatisoidusti siten, että uutta sisältöä julkaistaessa staattiset sivut generoidaan ja sivusto kootaan ja jaetaan sisällönjakeluverkkoon julkiseksi. Sivustoon tulee myös tuoda sisältöä ja toiminnallisuutta ulkoisista palveluista dynaamisuuden ja vuorovaikutuksellisuuden lisäämiseksi. Päämääränä ja motiivina on palvelinvetoiselle dynaamiselle blogisivustolle toiminnallisuudeltaan kilpailukykyinen ratkaisu, joka on myös suorituskykyynsä ja kapasiteettiinsa nähden huomattavasti kustannustehokkaampi ylläpitää.

2 JAMSTACK-ARKKITEHTUURI

JAMstack on modernin web-kehityksen arkkitehtuuri, joka perustuu JavaScript-selainskripteihin, uudelleenkäytettäviin ohjelmointirajapintoihin (API, Application Programming Interface) ja ennalta käännettyyn merkintäkieleen (markup). Lyhenne ”JAM” tulee sanoista JavaScript, API:t ja markup. JAMstack toimii vaihtoehtoisena ratkaisuna web-kehityksen ohjelmistokokoelmille, kuten LAMP (Linux, Apache, MySQL ja PHP) ja MEAN (MongoDB, Express.js, AngularJS ja Node.js). JAMstack-arkkitehtuurin on kehittänyt Netlifyn toimitusjohtaja Mathias Biilmann. (1; 2.)

2.1 Ideologia

Kaikki dynaaminen toiminnallisuus pyyntö-vastaussyklin aikana ajetaan JavaScript-ohjelmakoodilla täysin selainpohjaisesti. Palvelinpuolen prosessit ja tietokantatoiminnot on erotettu ulkoisten uudelleenkäytettävien ohjelmointirajapintojen avulla (kuva 1). Tämä tukee palvelimetonta arkkitehtuuria suosien tarvittaessa kutsuttavia pilvifunktioita koko ajan ajettavien palvelinkoneiden sijaan. Merkintäkieli käännetään ennalta julkaisun yhteydessä, tyypillisesti sisältösivustot sivugeneraattorilla ja web-sovellukset koontityökalulla. (1; 2.)



KUVA 1. Perinteisen web- ja JAMstack-arkkitehtuurin erot (3)

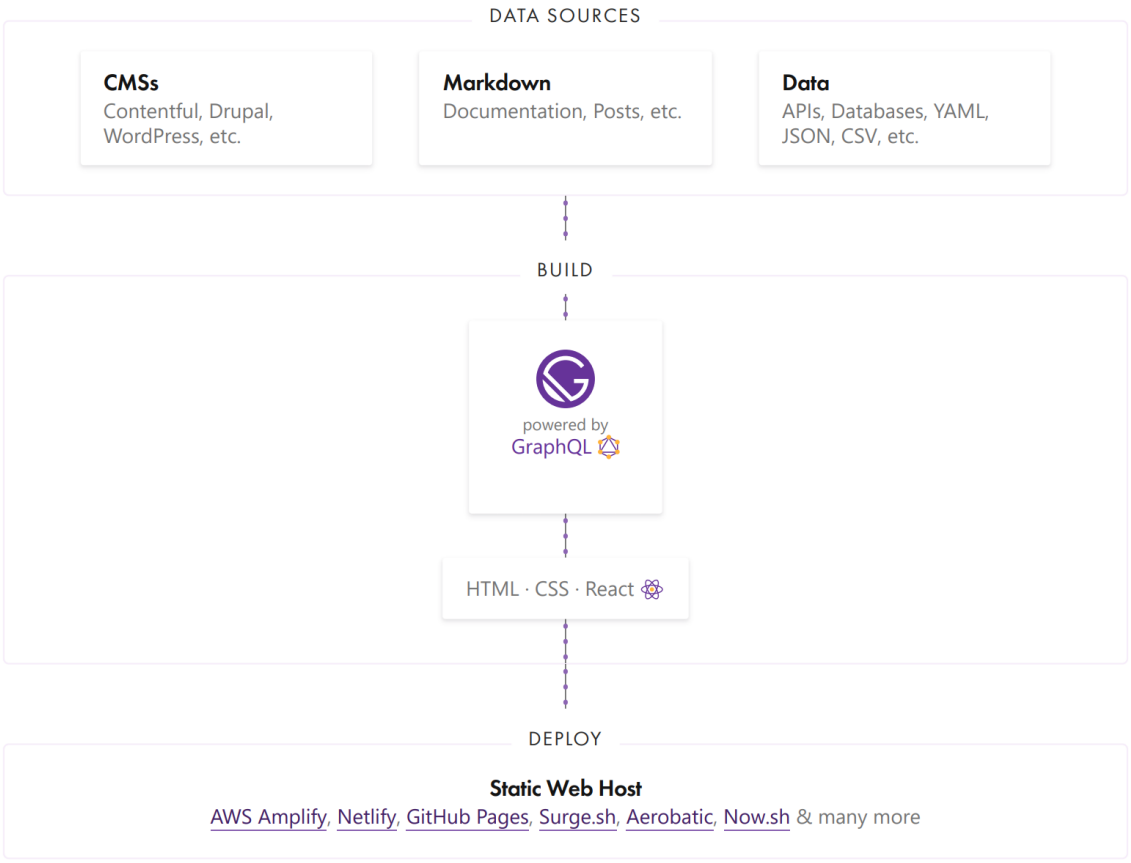
Kuvassa 1 perinteisellä web-arkkitehtuurilla (traditional web) viitataan ratkaisuihin, jotka ovat suoranaisesti riippuvaisia palvelinpuolen ajonaikaisesta ympäristöstä (esim. Node.js, Ruby) tai palvelinpuolen sisällönhallintajärjestelmästä (esim. Wordpress, Drupal). JAMstackin keskeisimpänä erona edellä mainittuihin on palvelinpuolen monimutkaisuuden väheneminen sen poistaessa tarpeen kiinteäkytköksiselle (tightly coupled) web- ja sovelluspalvelimelle sekä tietokannalle ja palvelinpuolen sisällönhallintajärjestelmälle. Sen sijaan web-sovellus jaetaan yhdenmukaisena ja staattisena sisällönjakeluverkon (CDN, Content Delivery Network) avulla ja erillisiä palveluja käytetään tarpeen mukaan suoraan selainohjelmasta. (3.)

Kuten mainittu, JAMstackia voidaan verrata LAMP- ja MEAN-ohjelmistokokoelmiin, mutta näistä poiketen se on ennemminkin kehitystapa kuin tietty kokoelma ohjelmistoja. LAMPia ja MEANia käytetään dynaamisten verkkosivujen ja -sovellusten kehittämiseen ja molemmissa toteutuksissa sivut renderöidään pyynnöstä ja sisältö haetaan tietokannasta. MEANilla on kuitenkin enemmän yhteistä JAMstackin kanssa. MEAN-kokoelman Angular.js toi suosioon yhden sivun sovellukset (SPA, single-page application), joissa ladataan vain yksi HTML-sivu, jota päivitetään dynaamisesti ohjelmistorajapintojen avulla. Se siirtää siis osan toiminnallisuudesta selainohjelmalle. Toteutustavaltaan JAMstack-sivusto on perinteisestä MEAN-sivustosta askel staattisempaan suuntaan, jossa oman kehitettävän ja ylläpidettävän palvelinohjelmiston ja tietokannan sijaan valtaosa sisällöstä kootaan ennalta ja vain sitä eritoten vaativat dynaamiset elementit toteutetaan ulkoisten palveluiden rajapintakutsuilla. (2; 4.)

2.2 Työnkulku

JAMstack ei määritä, millä tapaa HTML-dokumentit tulisi luoda. Yleisin lähestymistapa kuitenkin on rakentaa web-sovellus selainpuolen viitekehystä (esim. React, Vue.js tai Preact) käyttäen ja luoda eri sivujen generointia varten uudelleenkäytettävät komponentit ja mallit. Sivujen varsinainen sisältö haetaan pääasiallisesti koonnin aikana erillisistä lähteistä, kuten markdown-tiedostoista tai ul-

koisesta sisällönhallintajärjestelmästä. Lopulta HTML-dokumentit luodaan yhdessä komponenttien, mallien ja sisällön pohjalta staattisella sivugeneraattorilla (esim. GatsbyJS, Next.js, Hugo) (kuva 2). (5; 6.)



KUVA 2. GatsbyJS-sivugeneraattorin toimintaperiaate (7)

Yksi JAMstackin ydinperiaatteista on, että kaikki verkkosivuston ohjelmakoodi ja muu sisältö sijoitetaan keskitetysti Git-ohjelmavarastoon. Tämä tekee erittäin helpoksi modernin ohjelmistokehityksen jatkuvan integraation (CI, continuous integration) ja jatkuvan toimituksen (CD, continuous deployment) menetelmien toteuttamisen. Staattisiin verkkosivuihin erikoistunut julkaisualusta, kuten tässä työssä käytettävä Netlify, voidaan asettaa seuraamaan ohjelmavarastoa, kääntämään ohjelmakoodi ja julkaisemaan verkkosivusto sisällönjakeluverkkoon automaattisesti muutosten tapahtuessa. Julkaisu tapahtuu atomisena, eli kaikki muutokset viedään julki yhtäaikaisesti epäjohdonmukaisuuksien välttämiseksi. Julkaisun yhteydessä välimuisti mitätöidään muutosten välittömän voimaan astumisen varmistamiseksi. (2; 6.)

Asiakkaat ja muut ei-tekniset käyttäjät voivat tehdä muutoksia verkkosivulle erillisen sisällönhallintajärjestelmän (headless CMS) tai muun käyttöliittymän välityksellä ilman, että sen taustalla olevasta toteutuksesta tarvitsee olla tietoinen. Tällaisen järjestelmän (esim. Netlify CMS, Contentful) avulla käyttäjä voi luoda ja muokata tekstieditorissa sisältöä, joka tallennettaessa ladataan ohjelmavarastoon ja käännetään asianmukaiselle sivulle sivugeneraattorilla. (2; 6.)

2.3 Hyödyt

JAMstack tuo useita etuja niin käyttäjälle kuin kehittäjällekin. Koska sisältö on staattista, se voidaan jakaa sisällönjakeluverkkoon, jonka myötä latausajat lyhenevät ja resurssit ovat helpommin skaalattavissa. Lyhyemmät latausajat tarkoittavat myös parempaa hakukoneoptimointia, jota edesauttaa ennestään staattisten sivujen yksinkertainen sivuarkkitehtuuri. Ilman dynaamista palvelinohjelmistoa, sen liitännäisiä ja tietokantaa hyökkäyspinta-ala pienenee ja tietoturvallisuus kasvaa. (6.)

Kehittäjän näkökulmasta minimaalinen palvelintoteutus tarkoittaa ylläpidollisten töiden ja työvaiheiden vähenemistä. Kehittäjäkokemusta edistää myös löyhän kytkennän (loose coupling) tuomat vapaudet ja sen myötä paremmin kohdennettavissa oleva kehitystyö. Lisäksi koko verkkosivuston ja sen sisällön keskitetty sijainti ohjelmavarastossa mahdollistaa helposti toteutettavan CI/CD-putken. (2; 6.)

Suorituskyky

JAMstack-arkkitehtuurin tehokkuus on helppo ymmärtää, kun sitä verrataan tietokantapohjaisen dynaamisen verkkosivuston toimintaan, jossa jokainen yksittäinen sivulataus on monivaiheinen ja monimutkainen prosessi. Esimerkiksi WordPress-sivustolla käyttäjän ladatessa verkkosivun palvelin tekee kyselyjä MySQL-tietokannasta, käsittelee tietoja PHP-palvelinohjelmalla, yhdistää ne aktiiviseen teemaan ja liitännäisiin ja lopulta generoi HTML-dokumentin esitettäväksi käyttäjän selaimessa. Tyypillinen WordPress-sivusto tekee 30–100 kyselyä tietokantaan jokaisella sivulatauksella asennettujen liitännäisten määrän mukaisesti. (6; 8.)

On selvää, että jopa yksinkertainen blogisivusto on liian monimutkainen täysin käsin kirjoitettavaksi, ja siksi modulaarisuus ja automatisoitu koonti on tarpeen, mutta sen ei tarvitse välttämättä tapahtua palvelinpuolella. Nykypäivänä web-selaimet ovat suorituskykyisempiä ja on nopeampaa suorittaa logiikkaa paikallisesti. Staattiset sivugeneraattorit poistavat palvelinohjelman tarpeen ja sen sijaan, että sisältöä haetaan ja generoidaan pyyntökohtaisesti, tehdään se kaikki ennalta ja jaetaan sisällönjakeluverkkoon. (8.)

Hakukoneoptimointi

Staattiset verkkosivut ovat hakukoneystävällisempiä niiden nopeiden latausaikojen ja usein myös yksinkertaisempien URL-osoitteiden ja sivuarkkitehtuurin johdosta. Tyypillisesti dynaaminen verkkosivusto käyttää URL-uudelleenkirjoitusta URL-osoitteiden kääntämiseen selkokielisiksi. Staattisella verkkosivustolla tämä ei ole tarpeen, sillä URL-osoitteet kuvastavat alusta pitäen verkkosivuston tiedostojen fyysistä sijaintia. Näin hakukoneiden robottien sivuston selaus ja indeksointi helpottuu. Hakukoneet myös suosivat nopeasti latautuvia verkkosivustoja. Esimerkiksi Google pisteyttää verkkosivustot niiden latausaikojen mukaan ja suosii paremmin suoriutuvia hakutuloksissaan. (6; 9.)

Tietoturva

JAMstack-verkkosivuston hyökkäyspinta-ala on huomattavasti pienempi kuin tietokantapohjaisella dynaamisella verkkosivustolla. Tyypillisimpiä hyökkäysvektoreita, kuten palvelimella ajettavaa tietokantaa, liitännäisiä ja palvelinohjelmaa, ei ole. Ohjelmointirajapintojen kautta käytettävät ulkoiset palvelut ovat ainoa potentiaalinen hyökkäyskohde, mutta niiden ollessa irtonaisia ja eristettyjä verkkosivustosta on onnistuneen hyökkäyksen riski pieni. Tietoturvallisuus siis perustuu käytettävien ulkoisten palveluiden tietoturvaan. (6; 10.)

2.4 Soveltuvuus

JAMstack soveltuu parhaiten sellaisten sivustojen kehitykseen, joiden valtaosaa sisällöstä ei ole välttämätöntä päivittää pyyntökohtaisesti, kuten blogi-, portfolio-, uutis- ja yrityssivuilla. Esimerkiksi blogikirjoituksiin on vain harvoin tarvetta tehdä muutoksia enää sen julkaisun jälkeen, joten pyyntökohtaisen renderöinnin sijaan

on täysin perusteltua koota julkaisukohtaiset sivut ennalta. On tärkeää huomioida, että sivuston päivitettävyys on suoranaisesti yhteydessä sen laajuuteen, sillä sisällön kasvaessa kasvaa myös koontiaika. JAMstack on siis varteenotettava vaihtoehto pienistä keskisuurille sivustoille, joille sisällönhallintajärjestelmäpohjainen toteutus olisi ylenpalttinen ratkaisu. (6.)

Yksi tiettävästi laajimmista JAMstack-pohjaisista sivustoista on smashingmagazine.com. Kolmen vuoden takaisessa blogikirjoituksessaan Stefan Baumgartner kertoo, että he päätyivät JAMstackiin siirtyessään jakamaan 2000-sivuisen verkkosivustonsa osioittain omiin sisältöpaketteihin ja ohjelmavarastoihin, jonka myötä koontiajoiksi saavutettiin 2–5 minuuttia. Koontiaika koostui pääasiassa kuvien automatisoidusta käsittelystä ja optimoinnista. (11.)

Myöskään verkkokaupat eivät loppujen lopuksi ole niin dynaamisia, etteikö JAMstack niihinkin taipuisi. Yleisesti ottaen verkkokauppojen ainoa jatkuvasti päivitettävä tieto on tuotesaatavuus. Hintamuutoksia tuskin taas on tarvetta tehdä useammin kuin korkeintaan kerran päivässä. Loppu sisältö muuttuu tätäkin harvemmin. Esimerkiksi Snipcart tarjoaa staattisille sivuille JavaScript-pohjaisen ostoskori- ja kassajärjestelmän, hallintapaneelin varastotilanteen ja tilausten seurantaan sekä ohjelmointirajapinnan muihin järjestelmiin integraatiota varten. (6; 12.)

JAMstack ei ole pelkästään ohjelmistokehittäjille vaan tukee myös asiakasystävällisiä toteutuksia. Staattisia sisällönhallintajärjestelmiä on useita mistä valita, jotka mahdollistavat sisällön luomisen ja muokkaamisen käyttöliittymän avulla. Staattisten sivujen verkkoisännöinti on myös edullisempaa ja usein jopa ilmaista. Ilman kiinteäkytköksistä palvelinpuolta vähenevät myös ylläpidolliset kulut. Mahdollisia kuluja tulee sen sijaan ulkoisten palveluiden käytöstä ”maksa kun käytät”-periaatteella (”pay-as-you-go”). (13.)

3 BLOGISIVUSTON TOTEUTUS

Opinnäytetyön tavoitteena oli luoda JAMstack-arkkitehtuurin mukainen yhden kirjoittajan blogisivusto. Tähän lähestymistapaan päädyttiin useasta syystä. Ensiksikin, kun sivusto kootaan ennalta, palvelinpuolelta vaaditaan vähemmän, jonka ansiosta verkkoisännöintikulut ovat pienemmät tai jopa olemattomat. Staattisen sivuston verkkoisännöinti ei vaadi muuta kuin tallennustilan sen tiedostoille. Toki myös palvelinvetoisia sivustoja, kuten WordPress-sivustoja, on mahdollista ylläpitää vähin kuluin, mutta silloin sen suorituskyky ja käyttökapasiteetti on hyvin rajoittunutta. Tämä vaikuttaa negatiivisesti etenkin käyttökokemukseen ja voi myös estää käytettävyyden kokonaan suuren kävijäpiikin kohdalla. Staattinen sivusto taas voidaan jakaa kokonaisuudessaan sisällönjakeluverkkoon, jolloin sen toimintavarmuus on suurempi. JAMstackin valintaan vaikutti myös sivuston staattisuuden myötä pienentynyt hyökkäyspinta-ala, kuten on käsitelty kappaleessa 2.2.

Arkkityyppisen mallin lisäksi valinnanvaraa on myös koontityökaluissa ja sivugeneraattoreissa. Käytetyin näistä on Jekyll, joka toimii myös moottorina GitHub Pages -palvelulle, jonka avulla käyttäjät voivat isännöidä verkkosivustoja suoraan Git-ohjelmavarastoista (14). Tässä työssä päädyttiin kuitenkin GatsbyJS-sivugeneraattoriin sen suosittuuden, ympärillä olevan aktiivisen yhteisön ja näiden myötä laajan tuen ja lisäosatarjonnan johdosta. Valintaan vaikutti vahvasti myös henkilökohtainen mielenkiinto perehtyä React-kirjastoon, jota GatsbyJS käyttää sivustojen kehittämiseen.

Blogisivusto kehitetään React-kirjastoa ja GraphQL-kyselykieltä käyttäen, josta kootaan staattinen verkkosivusto GatsbyJS-sivugeneraattorilla. Ohjelmakoodi varastoidaan Git-ohjelmavarastoon, jonka muutosten yhteydessä Netlify-palvelualusta automaattisesti kääntää ja julkaisee uuden version sivustosta. Blogikirjoitusten luontiin ja muokkaamiseen käytetään Netlify CMS -sisällönhallintajärjestelmää. Dynaaminen kommentointiosio tuodaan staattiselle sivustolle ulkoisen Disqus-palvelun avulla. Seuraavissa luvuissa käsitellään blogisivuston toteutuksen keskeisiä komponentteja ja niiden toiminnallisuutta.

3.1 React-kirjasto

React on Facebookin kehittämä ja käyttämä JavaScript-kirjasto käyttöliittymien kehittämiseen. React painottaa komponenttipohjaista kehittämistä, jonka etuna on mahdollisuus päivittää vain tarpeellinen elementti koko sivun uudelleenlatauksen sijaan. Komponentit ovat muista riippumattomia ja määrittävät osan käyttöliittymän ulkoasua ja toiminnallisuutta. Komponentit voivat olla niin kutsuttuja tilallisia komponentteja (stateful components), jotka sisältävät tilamuuttujia, joiden muutokset aiheuttavat komponenttien uudelleenrenderöinnin. React-komponentit kehitetään useimmiten käyttäen JavaScriptin JSX-syntaksilisäosaa, joka yhdistää HTML-koodin JavaScriptiin. (15.)

3.2 GraphQL-kyselykieli

GraphQL on kyselykieli ohjelmointirajapinnoille ja palvelinpuolen ajonaikainen ympäristö kyselyjen suorittamiseen. GraphQL:llä voidaan hakea vain oleellista ja välttämätöntä tietoa, ja se on tehtävissä yhdellä kyselyllä, vaikkakin lähteitä olisi useita. Kysely lähetetään merkkijonona samaa formaattia mukaillen, jossa siihen vastataan JSON-tiedostomuodossa (kuva 3). Myös GraphQL on Facebookin kehittämä ja sen aktiivisessa käytössä. (16.)

```
{
  markdownRemark
  (
    fields:
    {
      slug:
      {
        eq: "/blog/ensimmaiset-kuukaudet-suomessa/"
      }
    }
  )
  {
    frontmatter {
      title
      date
      (
        formatString:"MMMM DD, YYYY",
        locale: "fi"
      )
      tags
    }
  }
}
```

```
{
  "data": {
    "markdownRemark": {
      "frontmatter": {
        "title": "Ensimmäiset kuukaudet Suomessa",
        "date": "toukokuu 01, 2019",
        "tags": [
          "hyerisuomessa"
        ]
      }
    }
  }
}
```

KUVA 3. Esimerkki GraphQL-kysely ja JSON-vastaus

3.3 GatsbyJS-sivugeneraattori

GatsbyJS on kehitysympäristöön asennettava sivugeneraattori, jolla voidaan koota staattisia PWA-sovelluksia (progressive web app). PWA:lla tarkoitetaan verkkosovellusta, joka toimii mobiilisovelluksen tavoin ja on käytettävissä myös offline-tilassa. GatsbyJS hyödyntää modernin web-kehityksen teknologioita, kuten React ja GraphQL. Sen toiminnallisuutta voidaan lisätä ja muuttaa asentamalla lisäosia (plugins). Renderöitävien staattisten sivujen sisältöä voidaan hakea useista eri lähteistä, kuten markdown- ja CSV-tiedostoista sekä sisällönhallintajärjestelmistä. (17.)

GatsbyJS-projektit noudattavat tiettyä kansiorakennetta (kuva 4). Olennaisin kansio on *src*, joka sisältää verkkosovelluksen käyttöliittymän muodostavan ohjelmakoodin. Kansion *src/pages* React-komponenteista muodostetaan automaattisesti niiden tiedostonimiä vastaavat sivut. Blogisivuston yleistä tietoa sisältävä about-sivu luodaan näin. (18.)

KUVA 4. Blogisivuston GatsbyJS-projektin pelkistetty tiedostorakenne

Toistuvalle sisällölle, kuten blogikirjoituksille, on kuitenkin perustellumpaa luoda sivut ohjelmallisesti ja käyttää tähän näille yhteistä sivupohjaa (template). Sivupohja on React-komponentti, johon haetaan ja sijoitetaan sisältöä ulkoisesta lähteestä GraphQL-kyselyllä. Esimerkiksi blogikirjoitukset tallennetaan kansioon *src/pages/blog* markdown-tiedostoina, jotka toimivat näille yhteisen sivupohjan ulkoisena sisällön lähteenä. Markdown-tiedostojen sisällön jäsentelyyn GraphQL-kentiksi käytetään lisäosaa gatsby-transformer-remark. Sivupohjat sijaitsevat kansiossa *src/templates*. Kansio *src/components* ei ole osa tiedostorakenteen standardia, mutta sitä on käytetty blogisivuston keskitettynä sijaintina eri sivuilla toistuville elementeille uudelleenkäyttöä varten (kuva 5). Kansion *static* sisältöä ei prosessoida millään tavalla, vaan se julkaistaan sellaisenaan verkkosivustolla. (18.)

```
import React from 'react';
import Img from 'gatsby-image'
import Slider from 'react-slick';
import 'slick-carousel/slick/slick.css';
import 'slick-carousel/slick/slick-theme.css';

const InstaFeed = ({ edges }) => {
    var settings = {
        ...
    };

    return (
        <div>
            {typeof window !== 'undefined' ?
                edges.length > 0 &&
                    <Slider{...settings}>
                    {edges.map(edge => (
                        <a
                            href={"https://www.instagram.com/p/${edge.node.id}/"}
                            target="_blank"
                            rel="noopener noreferrer"
                        >
                            <Img fixed={edge.node.localFile.childImageSharp.fixed}/>
                        </a>
                    ))}
                    </Slider>
            : null}
        </div>
    );
};

export default InstaFeed;
```

KUVA 5. Blogisivuston React-komponentti Instagram-kuvakarusellille

GatsbyJS-projektin tiedosto *gatsby-config.js* on sivuston keskeinen konfiguraatiotiedosto. Siinä määritellään muun muassa sivuston metadata ja käytettävät

Gatsby-lisäosat. Tiedostossa *gatsby-node.js* voidaan toteuttaa Gatsbyn ohjelmointirajapintoja ja täten kustomoida sivuston koontiprosessia. Ohjelmallinen sivujen luonti tapahtuu GatsbyJS:n rajapintametodien *onCreateNode* ja *createPages* avulla. Metodia *onCreateNode* kutsutaan, kun mikä tahansa sisältöobjekti (node) luodaan, ja siksi tällöin on otollista luoda myös polkutunnukset (slug). Metodia *createPages* kutsutaan nimenomaisesti, jotta voidaan luoda sivuja sivupohjista. Esimerkiksi blogikirjoitusten tapauksessa haetaan ensin kaikki markdown-tiedostot kansiosta *src/pages/blog* GraphQL-kyselyllä. Sitten jokaiselle kyselyn vastauksen sisältöobjektille luodaan sivu aiemmin määriteltyyn polkutunnukseen käyttäen sivuille yhteistä sivupohjaa (kuva 6). (19.)

```
const path = require("path")
const { createFilePath } = require(`gatsby-source-filesystem`)

exports.onCreateNode = ({ node, getNode, actions }) => {
  const { createNodeField } = actions
  if (node.internal.type === `MarkdownRemark`) {
    const slug = createFilePath({ node, getNode, basePath: `pages` })
    createNodeField({
      node,
      name: `slug`,
      value: slug,
    })
  }
}

exports.createPages = ({ graphql, actions }) => {
  const { createPage } = actions
  return graphql(`
    {
      allMarkdownRemark(
        sort: { order: DESC, fields: [frontmatter___date] }
        limit: 2000
      ) {
        edges {
          node {
            fields {
              slug
            }
          }
        }
      }
    }
    `
  ).then(result => {
    const posts = result.data.allMarkdownRemark.edges
    posts.forEach(({ node }, index) => {
      createPage({
        path: node.fields.slug,
        component: path.resolve(`./src/templates/blog-post.js`),
        context: {
          slug: node.fields.slug,
          prev: index === 0 ? null : posts[index - 1],
          next: index === result.length - 1 ? null : posts[index + 1],
        },
      })
    })
```

KUVA 6. Polkutunnusten ja sivujen luonti blogikirjoituksille

Kun sivu kootaan, GatsbyJS generoi sivujen HTML-dokumentit sekä ajonaikaisen JavaScript-ympäristön, joka ajetaan, kun selain on ladannut sivuston alustavan HTML-koodin. JavaScript-ohjelmakoodi ja CSS-tyylimäärittelyt jaetaan automaattisesti sivu- ja komponenttikohtaisesti erillisiin tiedostoihin. CSS voidaan myös sisällyttää HTML-koodiin. Näin voidaan ladata vain sivukohtaiset välttämättömät resurssit renderöintiä varten. Tätä hyödynnetään edelleen sivujen sisällä lataamalla sisältöä vasta vierityksen edetessä. Lisäksi ladatulle sivulle linkitettyjen sivujen tiedostoja ladataan ja säilötään ennalta taustalla, jolloin käyttäjän näkökulmasta sivustoa selatessa sivut latautuvat välittömästi. (20.)

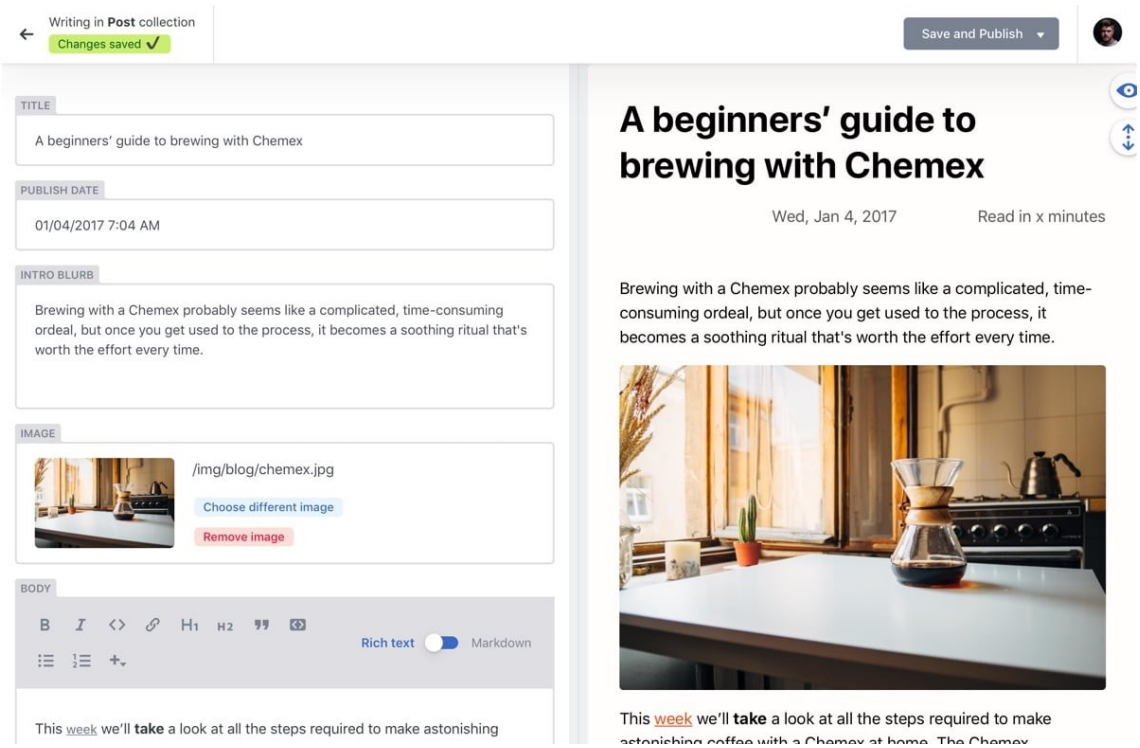
3.4 Netlify-palvelualusta

Netlify on palvelualusta modernien staattisten sivujen verkkoisännöintiin. Se tarjoaa esimerkiksi jatkuvan toimituksen Git-käynnisteisen koonnin avulla, maailmanlaajuisen sisällönjakeluverkon, täyden DNS-hallinnan ja automatisoidun HTTPS-käyttöönoton. Netlify on erinomainen ratkaisu staattisella sivugeneraattorilla, kuten GatsbyJS:llä, koottavalle sivustolle, sillä se voidaan asettaa generoimaan ja toimittamaan sivusto automaattisesti ohjelmavaraston muutosten yhteydessä. Edellä mainitut ominaisuudet sisältyvät ilmaiseen palvelutasoon. (21.)

Palvelun käyttöönotossa luodaan uusi sivusto Git-ohjelmavaraston pohjalta ja valtuutetaan Netlifylle tähän lukuoikeudet. Toimitusasetuksissa (deploy settings) määritetään käytettävä ja toimitettava ohjelmavaraston haara, ajettava koontikomento ja sivuston julkaisukansio. Tämän jälkeen sivusto on jo toimitettavissa ja haettavissa netlify.com-alitunnisteen verkko-osoitteesta. Sivustolle voidaan asettaa oma mukautettu verkkotunnus DNS-asetuksista. Blogisivuston toteutuksessa Netlify siis ajaa koontikomennon Git-ohjelmavaraston muutosten yhteydessä, jolloin GatsbyJS kokoaa tämän sisällön pohjalta uuden version verkkosivustosta, jonka Netlify sitten toimittaa sen sisällönjakeluverkkoon. (21.)

3.5 Netlify CMS -sisällönhallintajärjestelmä

Netlify CMS on avoimen lähdekoodin sisällönhallintajärjestelmä Git-ohjelmavaraston sisällön ja tiedostojen muokkaamiseen. Se tuo staattiselle verkkosivustolle sisällön muokkaamista varten käyttöliittymän, jossa jokainen julkaistu muutos tallennetaan ohjelmavarastoon ohjelmointirajapintakutsulla. Eri sisältötyypeille määritetään ennalta kokoelmat (collections) ja niiden tietokentät (fields), joita voidaan muokata käyttöliittymän komponenttien, kuten tekstieditorin, avulla (kuva 7). Sisältö voidaan tallentaa joko markdown-, JSON-, YAML- tai TOML-tiedostomuodossa. Blogisivuston toteutuksessa käyttöliittymällä luodut blogikirjoitukset käännetään markdown-tiedostomuotoon ja tallennetaan Git-ohjelmavarastoon ohjelmointirajapintakutsulla. (22; 23.)



KUVA 7. Esimerkki blogikirjoituksen muokkaamisesta Netlify CMS:llä (24)

Netlify CMS -sisällönhallintajärjestelmän käyttöönotto ja sisällytys Gatsby-verkkosivustoon edellyttää sen lisäosien asentamista ja rekisteröimistä verkkosovellukseen. Lisäksi tätä varten tulee luoda konfiguraatiotiedosto, jossa määritellään ohjelmavaraston sijainti ja nimi, verkkosovelluksen keskeiset hakemistot sekä kokoelmat eri sisältötyypeille. Jotta yhteys ohjelmavarastoon on mahdollinen, tulee sen sijaita esimerkiksi GitHub-palvelussa. Käyttäjä tulee myös autentikoida sisällönhallintasivulle pääsyä varten, joka on helpoimmin toteutettavissa Netlifyn palveluilla Git Gateway ja Netlify Identity. Näiden avulla voidaan hallita, autentikoida ja asettaa käyttöoikeuksia verkkosivuston ylläpitäjille. (23.)

3.6 Disqus-kommentointipalvelu

Disqus on kolmannen osapuolen palvelu, jolla voidaan tuoda kommentointi- ja muita yhteisöllisyysominaisuuksia verkkosivustoille. Käyttäjä voi sen avulla jättää kommentteja sisältösivuille joko kirjautuneena sosiaalisen median käyttäjätileillään tai kirjautumatta vieraana sekä jakaa sisältöä ulkoisissa palveluissa. Disqus tarjoaa myös työkaluja keskustelun moderointiin ja ylläpidollisiin tehtäviin. Disqusilla on virallinen tuki Reactille, joka muun muassa mahdollistaa kommenttikentän

reaaliaikaisen uudelleenlatauksen uuden kommentin saapuessa. Lisäksi sen kaikki komponentit ladataan lazy loading -toimintoa käyttäen eli vasta vierityksen edetessä ja elementin ollessa esillä, eivätkä ne siten vaikuta sivujen latausaikoihin. (25; 26.)

Palvelu voidaan käyttöönottaa Gatsby-verkkosivustolla asentamalla tämän virallinen React-paketti. Disqus-kommenttiosio lisätään sisältösivuille tuomalla paketin DiscussionEmbed-komponentti näiden sivupohjaan. Komponentille annetaan parametrina konfiguraatio-objekti, jossa määritellään verkkosivuston tunnus (shortname) sekä sisällön yksilöivät tiedot, kuten otsikko ja ID-tunniste. Näin Disqus luo jokaiselle sisältösivulle omat kommenttiketjut. (26.)

4 TOTEUTUKSEN ARVIOINTI JA TESTAUS

Tässä luvussa tarkastellaan julkaistua blogisivustoa ja arvioidaan työntulosta työn alussa asetettuihin lähtökohtiin ja tavoitteisiin nähden. Ensiksi esitellään blogisivuston toteutus ja keskeisimmät ominaisuudet yleisellä tasolla. Sitten testataan sivuston toimintaa ja suorituskykyä Chrome-selaimen kehittäjän työkaluilla Lighthouse ja Performance. Lopuksi käsitellään sivuston kehitystyötä henkilökohtaisten kokemusten pohjalta ja arvioidaan JAMstack-arkkitehtuurin soveltuvuutta käytettyyn tarkoitukseen.

4.1 Blogisivuston esittely

Julkaistu blogisivusto löytyy verkko-osoitteesta https://hyericho.com/. Etusivu vie suoraan blogiin, jossa esitetään viimeisimmät blogikirjoitukset. Nämä sivutetaan siten, että jokaisella sivulla esitetään kolme blogikirjoitusta. Etusivulle tuodaan myös blogikirjoittajan Instagram-tilin uusimmat kuvajulkaisut karusellielementtiin. Nämä molemmat kaksi sisältötyyppiä muodostetaan kokonaisuudessaan JAMstackin ideologialle ominaisesti jo koontivaiheessa. (Kuva 8.)

KUVA 8. Kuvankaappaus blogisivuston etusivusta mobiilinäkymässä

Blogikirjoituksille luodaan omat sivunsa, joihin tuodaan sisältö Netlify CMS -sisällönhallintajärjestelmällä luoduista markdown-tiedostoista. Blogikirjoitukset sisältävät pääasiassa tekstiä ja kuvia, jotka esitetään määritellyn sivupohjan mukaisesti. Jokaiselle julkaisulle luodaan keskustelu Disqus-palveluun, joka esitetään sivun loppuosassa. Julkaisulle on määritelty myös tagit eli avainsanat, joiden avulla voidaan hakea ja selata tietyntyyppistä sisältöä. Sivustolla etenemistä helpottavat generoidut navigaatiolinkit sisältösivuilla. (Kuva 9.)

KUVA 9. Disqus-komponentti blogikirjoituksen yhteydessä

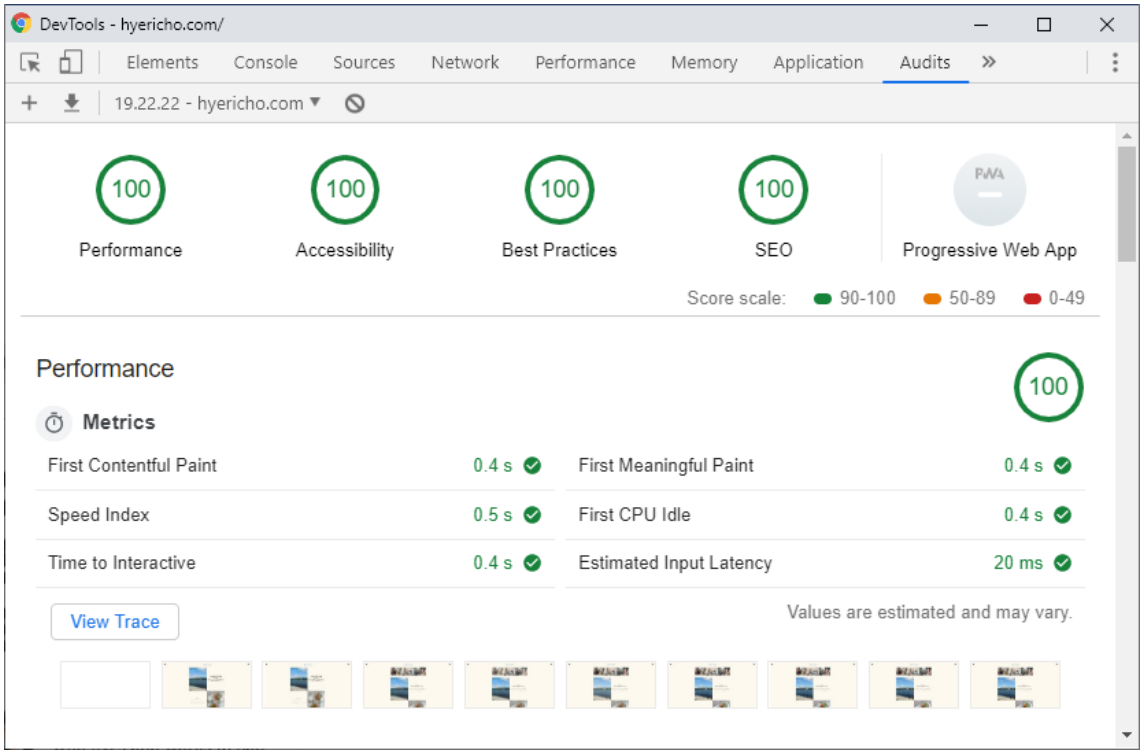
Jokaisella sivulla toistuu yhtenäinen alisivupohja. Se määrittelee sivuston laajuiset tyylittelyt ja komponentit, kuten logon, sivupalkki-valikon ja sosiaalisen median linkki-ikonit. Kaikille sivuille tuodaan myös hakukoneoptimointi-komponentti, jossa jäsennetään sivun tietoja hakukoneille ja sosiaalisen median sivustoille sopivaksi. Metatiedot esitetään dokumentin head-osiossa muun muassa standardien JSON-LD-, Open Graph- ja Twitter Cards -muodoissa. (27.)

Sivuston kaikki kuvat ladataan React-komponentilla gatsby-image, joka optimoi kuvia monilla tavoin. Se lataa kuvat lazy loading -toiminnolla, jonka myötä ensilatausaika (initial load) nopeutuu ja sivu on käytettävissä, vaikkakin kuvia vielä ladattaisiin. Kuvan viemä tila ja sijainti varataan jo latauksen aikana, jolloin kuvasta esitetään sumennettu matalan tiedostokoon versio. Kuvista generoidaan koonnin aikana useita eri kokoja ja niistä ladataan sopivin päätelaitteen tyypin ja näytön resoluution mukaan. (28.)

4.2 Toiminnan ja suorituskyvyn testaus

Google Lighthouse on avoimen lähdekoodin testaustyökalu, joka mittaa ja pisteyttää verkkosivun suorituskykyä, hakukoneoptimointia, parhaiden käytänteiden noudattamista, käytettävyyttä ja PWA-ominaisuuksia. Se mittaa yli 75 metriikkaa, joista se koostaa kokonaisvaltaiset pisteytykset. Pisteytyksen ohella se myös raportoi löydetyistä ongelmakohdista ja toimenpiteistä niiden korjaamiseksi. Lighthouse löytyy integroituna Chromen kehittäjän työkaluista. (29.)

Blogisivuston etusivulla ajettuna Lighthouse-testi antaa täydet pisteet jokaisella osa-alueella, mutta ei kuitenkaan täytä kaikkia PWA-sivuston vaadittuja ominaisuuksia (kuva 10). Tämä johtuu offline-tuen puutteesta, joka on tarkoituksenmukaisesti ainakin toistaiseksi poistettu käytöstä. Tämä todettavasti esti ajoittain uuden sisällön lataamisen ja esitti sen sijaan vanhentuneen version verkkosivustosta. Testin tulokset ovat samat myös sivuston mobiiliversiossa.



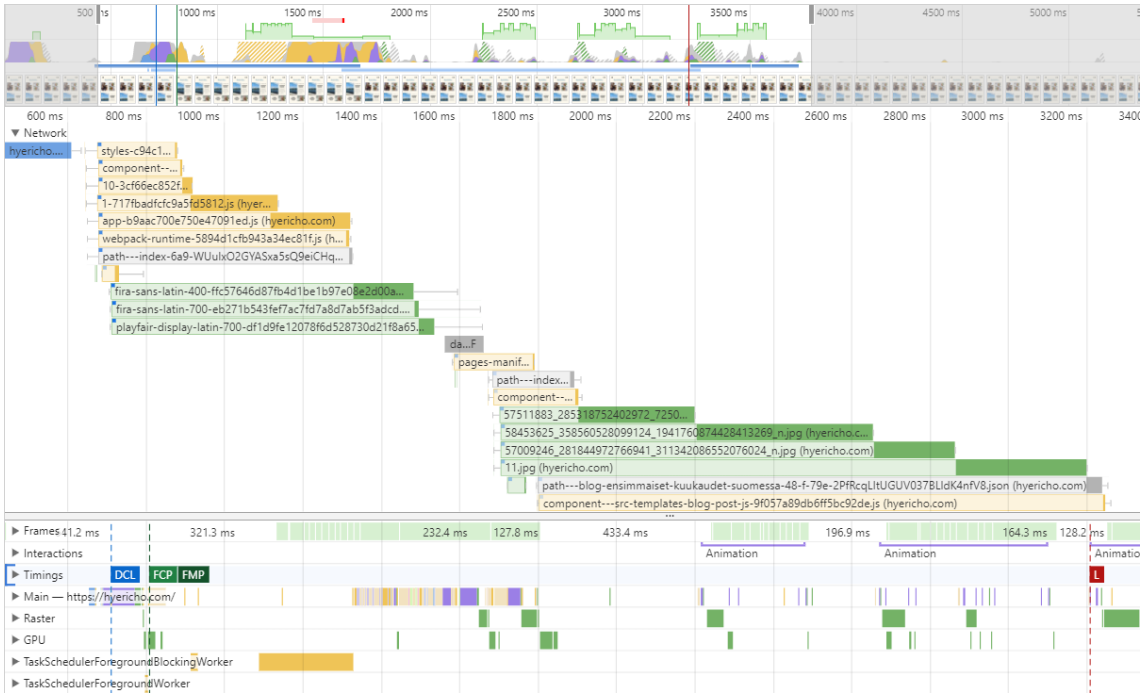
KUVA 10. Lighthouse-testin tulokset blogisivuston etusivulla

Blogikirjoitussivuilla pisteytys ja suorituskyvyn metriikat pysyvät muuten samoina, mutta parhaiden käytänteiden noudattamisen osalta pisteet laskevat 86:een. Ra-

portista käy ilmi, että pisteitä laskeva ongelmakohta on Disqus-komponentin sivunlatausta potentiaalisesti hidastava injektointitapa. Väitteen paikkansapitävyys on kuitenkin varsin kyseenalainen, sillä komponentti ladataan lazy loading -toiminnolla eikä se estä sivun latausta tai käytettävyyttä.

Lighthouse-testi ajettiin Windows 10 -käyttöjärjestelmän kannettavalla tietokoneella VDSL2-yhteyden langattomassa verkossa. Google Analyticsin statistiikkojen mukaan keskivertokäyttäjä käyttää kuitenkin Android-älypuhelinta ja oletettavasti selaa verkkosivustoa matkapuhelinverkkoyhteydellä. Käy siis järkeen mitata vielä suorituskykyä mukaillen edellä kuvailtua oletetusti yleisintä käyttötapausta.

Seuraavaksi suorituskykyä testattiin Android-laitteella LTE-verkkoyhteydellä. Testi suoritettiin etätestaamalla Android-laitetta Chromen kehittäjän työkaluilla. Välimuisti poistettiin käytöstä testin ajaksi ensivierailun latausajan mittaamiseksi. Testin tuloksista ilmeni, että sivunlatauspyynnöstä laskien HTML-dokumentin lataus ja jäsentely valmistui ajankohdalla 285,7 ms, ensimmäisen merkityksellisen sisällön (first contentful paint, first meaningful paint) renderöinti ajankohdalla 383,9 ms ja kaiken sisällön lataaminen ajankohdalla 2785,5 ms (kuva 11).



KUVA 11. Suorituskykytestin tulokset Android-laitteella LTE-yhteyttä käyttäen

4.3 Kokemukset ja arviointi

Henkilökohtaisesta näkökulmasta JAMstack-arkkitehtuurin toteutukset ja sivugeneraattorit eivät tarjoa yhtä lailla käyttövalmista tuotetta kuin palvelimelle asennettavat sisällönhallintajärjestelmät kuten WordPress. Ne siis todennäköisesti vaativat kehittäjältä enemmän, ja ilmiselvinäkin pidetyt ominaisuudet, kuten sisältösivujen ohjelmallinen luonti, sivunavigaatio ja hakukoneoptimointi, eivät tule käyttövalmiina, vaan ne pitää hyvin pitkälti kehittää kokonaan itse. Sivugeneraattoreille, kuten myös GatsbyJS:lle, on tarjolla kuitenkin valmiita projektipohjia eri käyttötarkoituksiin blogisivustoista verkkokauppoihin. Näiden lisäksi kattava lisäosatarjonta tuo myös joitain ominaisuuksia ja toimintoja valmiina paketteina sovellukseen asennettavaksi. Hyvänä kääntöpuolena tälle toki on sen tuomat vapaudet ja mahdollisuus aloittaa kehitys puhtaalta pöydältä ja täten välttää ominaisuuksien liika ahtaminen.

GatsbyJS-sivugeneraattori

Kokemukset käytetystä sivugeneraattorista GatsbyJS ovat hyvin positiiviset. Kehitystyö tämän kanssa oli hyvin vaivatonta perusteellisten dokumentaatioiden ansiosta, ja tätä helpotti edelleen paikallisessa kehitysympäristössä ajettava kehitystila, jossa ohjelmakoodin muutokset olivat nähtävissä selaimessa reaaliajassa. Maininnan arvoisia ovat ehdottomasti myös GatsbyJS:n sisäänrakennetut suorituskykyä edistävät ominaisuudet, kuten ohjelmakoodin pilkkominen, kuvien optimointi, kriittisten tyylimäärittelyjen rivikohtainen sisällytys, lazy loading -toiminto ja resurssien ennalta taustalla lataaminen (30).

Netlify-palvelualusta

Keskeisessä roolissa blogisivuston toteutuksessa on toki myös Netlifyn palvelualusta. Se tarjoaa paljon ilmaiseksi, ja vaikkakin staattinen sivusto olisi ollut mahdollista isännöidä muuallakin kuluitta, niin Netlifyn ominaisuudet kuten Git-käynnisteinen CI/CD-putki ja maailmanlaajuinen sisällönjakeluverkko tekivät valinnasta helpon. Käytännössä alun käyttöönoton ja konfiguraation jälkeen sivuston

koonnista ja toimituksesta ei ole tarvinnut suoranaisesti huolehtia, vaan sen sijaan on riittänyt sisällönhallintajärjestelmän käyttö blogikirjoituksia julkaistaessa ja Git-komentojen ajo ohjelmakoodia muokatessa.

Muut palvelut

Kokemukset toteutuksen komponenteista Netlify CMS ja Disqus ovat myös hyvät. Netlify CMS tuo helpon sivustoon sisäänrakennetun hallintapaneelin sisällön luomiseen ja muokkaamiseen. Käyttöliittymän myötä teknisen osaamisen vaatimus madaltuu eikä sisällönluojan ole tarpeen käyttää suoranaisesti markdown-kuvauskieltä tai Git-ohjelmavarastoa. Disqus taas mahdollistaa reaaliaikaisen kommentoinnin muuten staattisella sivustolla. Se tuo myös muita yleisöä aktivoivia ominaisuuksia, kuten reaktiopainikkeet, ja työkaluja kommenttien moderointiin. Tarpeen vaatiessa kommenttiketjut voidaan viedä palvelusta XML-tiedostoksi, mikäli palvelusta halutaan siirtyä toiseen. Myös Disqus on blogisivuston käyttötarkoituksessa ilmainen.

Toteutuksen rajoitteet

Netlifyn ilmainen palvelutaso sisältää 100 Gt dataliikennettä kuukaudessa (31). Ylärajan paremmin hahmottamiseksi voidaan tätä arvioida sivun katseluina ja kävijämääränä. Viime kuukauden aikana dataa on kulunut 917 Mt ja sivun katseluja ollut 4 298. Samassa suhteessa 100 Gt siis vastaisi noin 480 000 sivun katselua, ja keskimääräisen katseltujen sivujen luvun ollessa 4,68 tämä vastaisi hieman yli 100 000:ta vierailua. Lisäksi vierailukohtainen tiedonsiirto pienentynee ennestään vastikään käyttöönotetun laaja-alaisemman selaimen välimuistiin tallennettavan tiedon myötä. Nykyisellä toteutuksella blogisivusto on ylläpidettävissä täysin ilmaiseksi edellä mainittuun ylärajaan asti.

Toinen mahdollinen rajoite blogisivuston toteutukselle on Git-ohjelmavaraston suuruus. Käytettävä palvelu GitHub kertoo, että on suositeltavaa pitää ohjelmavarastot alle 1 Gt:n kokoisina. Varsinainen yläraja on kuitenkin 100 Gt. Blogisivuston ohjelmavaraston tallennustilaa vievät pääasiassa kuvatiedostot. Tähänastisten julkaistujen blogikuvien tiedostokokojen keskiarvosta laskettuna projektin 1 Gt:n koko täyttyisi vasta reilun 9 000 kuvan kohdalla, joten tämäkin on

varsin kaukainen huolenaihe. Tarvittaessa kuvat voitaisiin kuitenkin pitää ohjelmavarastosta kokonaan erillään ja isännöidä esimerkiksi Netlifyn juuri tähän tarkoitetussa Git LFS -pohjaisessa (Git Large File Storage) Large Media -palvelussa. (32; 33.)

5 YHTEENVETO

Opinnäytetyön tarkoituksena oli perehtyä modernin web-kehityksen JAMstack-arkkitehtuuriin ja luoda sen ideologian mukainen blogisivusto. Tämän myötä opinnäytetyössä tutkittiin myös staattisen verkkosivuston soveltuvuutta eri käyttötarkoituksiin, mutta etenkin blogisivuston toteutukseen. Staattisen blogisivuston tuli sisältää sisällönhallintajärjestelmä ja kommenttikenttä sekä tuoda ja esittää sisältöä ulkoisista lähteistä. Sivuston kehitysympäristön tuli olla automatisoitu ja toteuttaa jatkuvan integraation ja toimituksen mallia. Työn tavoitteena oli saavuttaa mahdollisimman suorituskykyinen toteutus mahdollisimman vähin käyttökuluin. Opinnäytetyössä käsiteltiin ja testattiin blogisivuston toteutusta sekä tarkasteltiin siinä käytettäviä työkaluja ja palveluja.

Työn tuloksena on asetettuja tavoitteita vastaava blogisivuston toteutus, joka sisältää kaikki edellä määritellyt ominaisuudet. Toteutus on testatusti suorituskyvyltään kiitettävää tasoa etenkin, kun huomioidaan sen käyttöönoton ja ylläpidon ilmaisuus. Lisäksi sen kapasiteetti riittää hyvin pitkälle tulevaisuuteen blogisivuston sisällön ja vierailukertojen kasvaessa.

PWA-sovellukselle ominaista offline-toimintoa ei vielä toistaiseksi kuitenkaan käyttöönotettu blogisivustolla, jotta vältyttäisiin vanhentuneen sisällön esittämisestä käyttäjälle. Tämän toiminnon lisääminen voisikin olla yksi jatkokehitysmahdollisuuksista, mutta ensisijaisen ratkaisun sijaan sen tulisi toimia vain varasuunnitelmana internet-yhteyden puutteelle. Blogisivustoon voitaisiin lisätä myös toinenkin PWA-toiminto, nimittäin push-ilmoitukset, joilla voidaan tiedottaa palvelun tilaajia uuden sisällön saatavuudesta ilman, että web-sovelluksen tarvii olla aktiivinen. Tämä voisi olla mahdollinen toteuttaa asettamalla webhook-toiminto blogisivuston uuden version julkaisutapahtumaan.

JAMstack on varsin tuore lähestymistapa verkkosivustojen toteutukseen. Tämän takia se ei tarjoa kovinkaan käyttövalmiita ratkaisuja, ja sitä tukevia palveluita ja työkaluja on rajallinen määrä. Kehittämisen varaa siis on ja tämä näkyy paikoittain myös asiakkaille suunnatuissa käyttöliittymissä. Kehitystyö vaatii siis enemmän kuin monoliittisen arkkitehtuurin sovelluksen, kuten WordPressin, asennus.

Tämän opinnäytetyön esittelemä JAMstack-arkkitehtuurin mukainen toteutustapa on hyödynnettävissä myös toteutun blogisivuston ulkopuolella erityyppisillä verkkosivustoilla. Sama perusideologia pätee käyttökohteesta riippumatta, oli kyseessä sitten blogisivusto tai verkkokauppa. Käytettävät palvelut voivat toki olla erit, kuten myös työkalut ja tekniikat. Kuten todettu, JAMstack ei ole kokoelma ohjelmistoja vaan tapa kehittää.

Opinnäytetyön myötä voidaan todeta JAMstackin olleen erinomainen toteutustapa blogisivuston toteutukseen käytettyine työkaluineen. Blogisivuston käyttäjäkokemusta edistää myös GatsbyJS ja sen tuoma natiivisovellukselle ominainen toimintatapa, jossa sisältö latautuu näennäisen välittömästi sitä selatessa. JAMstack tekee mahdolliseksi etenkin pienten ja keskisuurten sivustojen toteutuksen vähin käyttökuluin ilman, että niiden toiminnasta on tarpeen tinkiä.

LÄHTEET

1. JAMstack. Saatavissa: https://jamstack.org/. Hakupäivä 4.4.2019.

2. Myers, Astasia 2018. The JAMstack: It's Pretty Sweet. Memory Leak. Saatavissa: https://medium.com/memory-leak/the-jamstack-its-pretty-sweet-e0834e4e6bb7. Hakupäivä 4.4.2019.

3. Bull, Matthew 2017. JAMstack. LifeinTECH. Saatavissa: https://www.lifeintech.com/2017/12/20/jamstack/. Hakupäivä 4.4.2019.

4. WTF is JAMstack?. Saatavissa: https://jamstack.wtf/. Hakupäivä 13.4.2019.

5. conceptanext 2017. JAMstack Review. Concepta. Saatavissa: https://conceptainc.com/blog/what-you-should-know-about-jamstack/. Hakupäivä 16.4.2019.

6. Bennet, Tom 2017. Go static: 5 reasons to try JAMstack on your next project. Builtvisible. Saatavissa: https://builtvisible.com/go-static-try-jamstack/. Hakupäivä 4.4.2019.

7. GatsbyJS. Saatavissa: https://www.gatsbyjs.org/. Hakupäivä 15.4.2019.

8. Copes, Flavio 2018. What is the JAMstack? Saatavissa: https://flaviocopes.com/jamstack/. Hakupäivä 4.4.2019.

9. Sanchez-Olvera, Alex 2018. What (the Hell) Is the JAMstack? Medium. Saatavissa: https://medium.com/@alexsanchezdesigns/what-the-hell-is-jamstack-5ef002963f26. Hakupäivä 5.4.2019.

10. Lichter, Alexander 2019. Going JAMstack with Netlify and Nuxt. Blog.Lichter.io. Saatavissa: https://blog.lichter.io/posts/going-jamstack-with-netlify-and-nuxt/. Hakupäivä 5.4.2019.

11. Baumgartner, Stefan 2016. Using A Static Site Generator At Scale: Lessons Learned. Smashing Magazine. Saatavissa: https://www.smashingmagazine.com/2016/08/using-a-static-site-generator-at-scale-lessons-learned/. Hakupäivä 16.4.2019.

12. Conforti, Filippo 2018. How to Build Ecommerce Websites in 2019 (Hint: Static). Commerce Layer. Saatavissa: https://commercelayer.io/blog/how-to-build-ecommerce-websites-in-2019/. Hakupäivä 17.4.2019.

13. Dionne, Mathieu 2019. New to JAMstack? Everything You Need to Know to Get Started. Snipcart. Saatavissa: https://snipcart.com/blog/jamstack. Hakupäivä 17.4.2019.

14. Dionne, Mathieu 2018. Picking the Best Static Site Generator for Your Next Project. Snipcart. Saatavissa: https://snipcart.com/blog/choose-best-static-site-generator. Hakupäivä 5.5.2019.

15. Chand, Swatee 2019. What Is React? – Unveil The Magic Of Interactive UI With React. Edureka. Saatavissa: https://www.edureka.co/blog/what-is-react/. Hakupäivä 6.5.2019.

16. Khachatryan, Grigor 2018. What is GraphQL?. Devgorilla. Saatavissa: https://medium.com/devgorilla/what-is-graphql-f0902a959e4. Hakupäivä 6.5.2019.

17. NS, Ajay 2018. Why you should use GatsbyJS to build static sites. freeCodeCamp. Saatavissa: https://medium.freecodecamp.org/why-you-should-use-gatsbyjs-to-build-static-sites-4f90eb6d1a7b. Hakupäivä 17.4.2019.

18. Gatsby Project Structure. GatsbyJS. Saatavissa: https://www.gatsbyjs.org/docs/gatsby-project-structure/. Hakupäivä 7.5.2019.

19. Building with Components. GatsbyJS. Saatavissa: https://www.gatsbyjs.org/docs/building-with-components/. Hakupäivä 17.4.2019.

20. Herchel, Mike 2018. Why is Gatsby so Fast? The PRPL Pattern. Saatavissa: https://herchel.com/2018-01-10-why-is-gatsby-fast-prpl/. Hakupäivä 7.5.2019.

21. Hosting on Netlify. GatsbyJS. Saatavissa: https://www.gatsbyjs.org/docs/hosting-on-netlify/. Hakupäivä 24.4.2019.

22. Sourcing from Netlify CMS. GatsbyJS. Saatavissa: https://www.gatsbyjs.org/docs/sourcing-from-netlify-cms/. Hakupäivä 28.4.2019.

23. Add to Your Site. Netlify CMS. Saatavissa: https://www.netlifycms.org/docs/add-to-your-site/. Hakupäivä 28.4.2019.

24. Netlify CMS. Saatavissa: https://www.netlifycms.org/. Hakupäivä 28.4.2019.

25. Oday, Saeed 2019. How does Disqus work? Disqus. Saatavissa: https://help.disqus.com/what-is-disqus/how-does-disqus-work. Hakupäivä 28.4.2019.

26. Riebesell, Janosh 2019. Gatsby Blog with Disqus comments. janosh.io. Saatavissa: https://janosh.io/blog/disqus-comments. Hakupäivä 28.4.2019.

27. Adding an SEO component. GatsbyJS. Saatavissa: https://www.gatsbyjs.org/docs/add-seo-component/. Hakupäivä 2.5.2019.

28. Working With Images In Gatsby. GatsbyJS. Saatavissa: https://www.gatsbyjs.org/docs/working-with-images/. Hakupäivä 2.5.2019.

29. Kumar, Chandan 2019. How to test your Site with Google Lighthouse? Geekflare. Saatavissa: https://geekflare.com/google-lighthouse/. Hakupäivä 2.5.2019.

30. gatsbyjs/gatsby: Build blazing fast, modern apps and websites with React. GitHub. Saatavissa: https://github.com/gatsbyjs/gatsby. Hakupäivä 9.5.2019.

31. Morey, Todd 2019. Netlify and bandwidth. Netlify. Saatavissa: https://www.netlify.com/blog/2019/02/26/netlify-and-bandwidth/. Hakupäivä 9.5.2019.

32. What is my disk quota?. GitHub Help. Saatavissa: https://help.github.com/en/articles/what-is-my-disk-quota. Hakupäivä 9.5.2019.

33. Large Media. Netlify. Saatavissa: https://www.netlify.com/docs/large-media/. Hakupäivä 9.5.2019.

34. Oday, Saeed 2019. Comments Export. Disqus. Saatavissa: https://help.disqus.com/developer/comments-export. Hakupäivä 13.5.2019.