Pauliina Väisänen

Modernin verkkokehityksen työkalut ja tekniikat

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

5.4.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Pauliina Väisänen Modernin verkkokehityksen työkalut ja tekniikat 39 sivua 5.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Projektipäällikkö Elina Forsberg Lehtori Ilkka Kylmäniemi

Insinöörityön tavoitteena oli tutkia moderneja verkkokehitystyökaluja ja -tekniikoita ja niiden käyttöä verkkosivuston kehityksessä. Tarkoituksena oli koota kokoelma työkaluista, jotka helpottavat verkkokehityksen työnkulkua ja vähentävät manuaalisen työn määrää. Insinöörityön lopputuotteena syntyi uudet nykyaikaiset verkkosivut, joita on helppo ylläpitää niin asiakkaan kuin kehittäjän toimesta. Verkkosivut kehitettiin helsinkiläisen yliopiston osana toimivalle kesäkoululle.

Kesäkoulun verkkosivusto haluttiin uudistaa, koska yliopiston graafinen ilme muuttui ja sen myötä myös kesäkoulun sivustolle haluttiin päivittää yhteneväinen ilme. Vanha verkkosivusto ei myöskään ollut nykyajan standardien mukainen, joten se haluttiin uudistaa responsiiviseksi ja mobiiliystävälliseksi. Verkkosivustoja selaillaan yhä useammin mobiililaitteella, joten insinöörityössä oli tärkeää, että verkkosivusto tarjoaa yhtä miellyttävän käyttökokemuksen niin mobiililaitteen kuin tietokoneen näytöllä. Uudessa mobiiliystävällisessä sivustossa on otettu responsiivisuuden lisäksi huomioon muun muassa linkkien ja nappien käyttömukavuus pienellä mobiililaitteella.

Insinöörityön tuloksena saatiin koottua modernin verkkokehityksen työkalukokoelma, jota voi käyttää tulevaisuudessa uusien projektien pohjalla. Työkalujen pohjana on lokaali kehitysympäristö, joka mahdollistaa verkkokehityksen ilman erillistä aktiivista palvelinta. Kokoelman verkkokehitystä helpottavia työkaluja ovat esimerkiksi paketinhallintajärjestelmät, jotka helpottavat muiden kehittäjien luomien koodipakettien hyödyntämistä projekteissa. Paketinhallintajärjestelmien kautta voidaan ottaa käyttöön eri käyttötarkoituksiin suunniteltuja verkkokehitystyökaluja, kuten responsiivisen verkkosivuston kehitystä helpottava Foundation-ohjelmistokehys tai manuaalista työtä vähentävä Grunt, joka toimii automaattisena JavaScript-tehtävien suorittajana.

Insinöörityössä tutkittiin myös, millaisia työkaluja ja rajapintoja sosiaalisen median palvelut tarjoavat. Koska sosiaalinen media on jatkuvasti vahvistanut asemaansa yritysten modernina viestintävälineenä, useat verkkosivut sisältävät sosiaalisen median palveluiden kanssa keskustelevia elementtejä. Kesäkoulun sivuille tuotiin sisältöä Facebookin, Instagramin ja Twitterin rajapintoja apuna käyttäen.

Avainsanat	verkkokehitys, responsiivinen verkkosuunnittelu, sosiaalinen media

Author	Pauliina Väisänen
Title	Modern web development tools and techniques
Number of Pages	39 pages
Date	5 April 2016
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Elina Forsberg, Project Manager Ilkka Kylmäniemi, Senior Lecturer

The objective of this final year project was to research the use of modern web development tools and techniques created for developing a website. The purpose was to create a combination of tools that makes the workflow of development easier and reduces the number of manual tasks. The end result of this project is a new and modern website for a summer school operating under a university in Helsinki. The maintenance and updating of the new website is easy for both the customer and the developer.

The customers website needed to be updated because the graphic identity of the university changed, and so the look of the summers school needed to be updated too. Since the old website was not responsive or mobile friendly, the graphic look was not the only part in need of an update. Websites are used on mobile devices more frequently all the time, so while developing the new website, it was important to provide the same pleasant user experience on both mobile devices and desktops. The new mobile friendly website is responsive and the content was designed with mobile user experience in mind.

The outcome of this project is a collection of modern web development tools that can easily be used as a base for a new project. The base of the collection is a local development environment that enables web development without a separate active server. Useful web development tools include the package management systems that make it easy to use code packages created by other developers around the world. Packages studied in this thesis include Foundation, a front-end framework that simplifies the development of a responsive website, and Grunt, an automatic JavaScript task runner that automatizes many of the repetitive tasks of web development.

The project also included research on the tools and application programming interfaces that different social media platforms offer. Because social media continuously strengthens its value as a modern communication platform for companies, many websites contain elements that communicate with social media services. The customers new website fetches content from the application programming interfaces of Facebook, Instagram and Twitter.

Keywords	web development, responsive web design, social media

Sisällys

1 Johdanto

Insinöörityön tavoitteena on kehittää uusi helposti ylläpidettävä, moderni ja mobiiliystävällinen verkkosivusto Helsinki Summer Schoolille. Uuden verkkosivuston tavoitteena on toimia responsiivisesti ja moitteettomasti niin erinäisillä mobiililaitteilla kuin tietokoneella riippumatta näyttökoosta. Verkkosivuston tulee myös toimia kaikilla moderneilla ja käytetyimmillä vanhemmilla selaimilla.

Helsinki Summer School toimii Helsingin yliopiston alaisuudessa ja tarjoaa kesäkursseja niin suomalaisille kuin kansainvälisille yliopisto-opiskelijoille. Helsingin yliopiston graafinen ilme ja verkkosivusto päivittyivät alkuvuodesta 2015, joten myös Helsinki Summer School -sivuston ilme halutan päivittää yhdenmukaiseksi. Vanha verkkosivusto ei myöskään vastaa enää vaatimuksiin, joita nykyään verkkosivuille asetetaan, sillä sivusto on tehty vuonna 2008. Vanha sivusto ei ole responsiivinen, eikä sivuston päivittäminen ole asiakkaalle kovin helppoa.

Verkkosivusto toteutetaan moderneja verkkokehitystyökaluja käyttäen, ja toteutuksessa pyritään vaivattomaan ylläpitoon. Sivusto rakennetaan WordPress-sisällönhallintajärjestelmän päälle, koska se tarjoaa kehitykselle valmiin alustan, jonka ominaisuuksia on helppo hyödyntää. WordPress mahdollistaa myös sen, että asiakas voi helposti ja itsenäisesti ylläpitää sivustoa ja tehdä pieniä muutoksia sivuston ulkoasuun.

Insinöörityön tavoitteena on koota yhteen työkaluja, jotka helpottavat kehitystyötä riippumatta siitä, työskenteleekö projektin parissa yksi vai useampia kehittäjiä samanaikaisesti. Tavoitteena on saada aikaa toimiva ja monipuolinen työkalukokoelma, jota voidaan käyttää erilaisissa verkkosovellusprojekteissa. Valittujen työkalujen kriteerinä on myös se, että kehittäjän, joka ei tunne projektia, on helppo jatkaa sivuston kehitystyötä ilman alkuperäisen kehittäjän avustusta ja ohjausta.

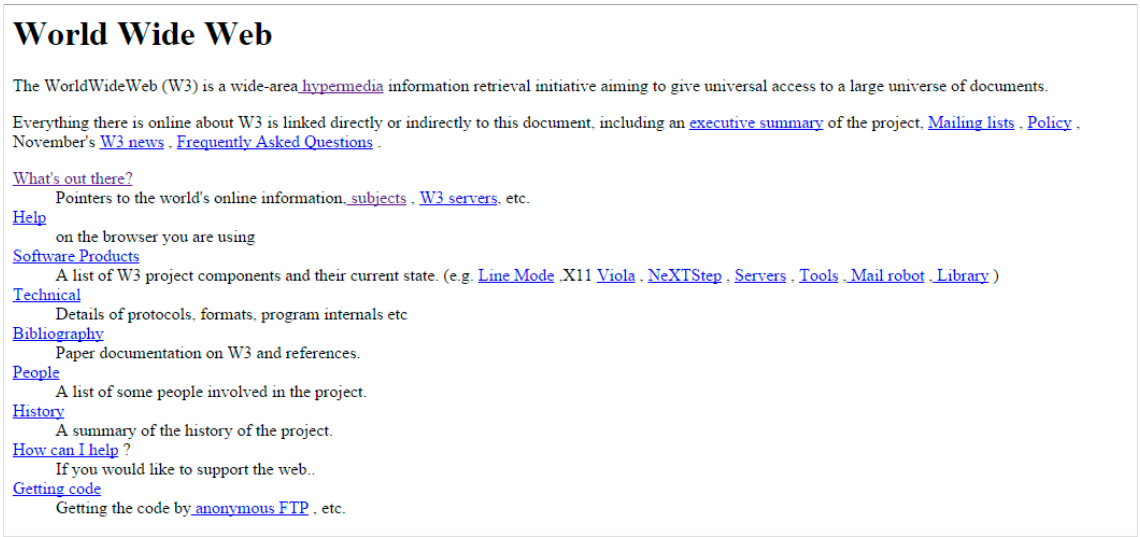
Kehitysympäristön ja -työkalujen tutkimisen lisäksi tavoitteena on paneutua niihin seikkoihin, jotka ovat nykyaikaisessa verkkosivustossa tärkeimpiä käyttäjille. Sivuston käyttäjä kaipaa virheetöntä ja sujuvaa käyttökokemusta huolimatta siitä, millä laitteella tai selaimella sivustoa käytetään. Käyttökokemuksen tulee olla myös yhtenevä erikokoisilla

näytöillä ja eri laitteilla. Käyttäjät kaipaavat luotettavaa, joustavaa ja ajankohtaista käyttökokemusta. Ajankohtaista sisältöä Helsinki Summer Schoolin sivuille halutaan tuoda sosiaalisen median sisältöseinällä, jonka sisältö päivittyy jatkuvasti.

2 Verkkokehityksen historiaa

2.1 Ensimmäisistä verkkosivuista nykypäivään

Maailman ensimmäinen verkkosivusto julkaistiin elokuussa vuonna 1991. Sivusto sisälsi selvityksen siitä, mitä World Wide Web eli maailmanlaajuinen verkko tarkoittaa, jatkuvasti päivittyvän listan kaikista maailman Internet-sivuista ja ohjeet siihen, miten kuka tahansa voi laajentaa ja parantaa maailmanlaajuista verkkoa. [1.] Tästä pelkkää tekstiä ja hyperlinkkejä sisältäneestä verkkosivustosta (kuva 1) on päästy pitkälle 24 vuoden aikana. Tämän lauseen kirjoitushetkellä aktiivisia verkkosivuja on tuhannen verkkosivuston tarkkuudella 929 797 000 [2].



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Kuva 1. Maailman ensimmäinen verkkosivusto [1].

Ensimmäiset verkkosivustot koostuivat pelkästä HTML:stä (Hypertext Markup Language eli hypertekstin merkintäkieli). Pian kuitenkin verkkosivustojen kehittäjät tahtoivat vaikuttaa sivujensa ulkoasuun, joten vuonna 1994 alettiin kehittää CSS-tyylikieltä (Cascading Style Sheets eli porrastetut tyyliarkit), joka mahdollistaa erilaisten tyylimääritteiden määrittämisen HTML-elementeille. Ensimmäinen julkinen versio CSS:stä, CSS1, julkaistiin

vuonna 1996. Tällä hetkellä yleisesti käytössä on kolmas versio, CSS3, joka otettiin laajalti käyttöön 2010-luvun alkupuolella. [3.]

HTML:n ja CSS:n lisäksi käytetyimpiä nykyaikaisten verkkosivustojen kehittämiseen luotuja ohjelmointikieliä ovat JavaScript ja PHP. JavaScript on ohjelmointikieli, jolla voidaan esimerkiksi muokata ja manipuloida verkkosivun sisältöä sekä sallia interaktio käyttäjän ja verkkosivuston välillä. Se julkaistiin vuonna 1996 nimellä Mocha, ja uusin versio on JavaScript 1.8.2, joka julkaistiin vuonna 2009. [4.] PHP sen sijaan on ohjelmointikieli, joka toimii palvelinpuolella ja suorittaa koodin, ennen kuin verkkosivu latautuu selaimessa. PHP julkaistiin vuonna 1995 nimellä PHP/FI (Personal Home Page/Forms Interpreter). PHP:n uusin versio PHP 7 julkaistiin joulukuussa 2015, mutta useimmilla sivustoilla on vielä käytössä PHP 5 tai vanhempi versio. PHP 6 hylättiin eikä sitä julkaistu. [5.]

Tekniikoiden kehittyessä verkkokehittäjät halusivat kokeilla jokaista uutta teknologiaa ja ideaa, joten kun kuvien ja animaatioiden näyttäminen tuli uusien tekniikoiden myötä mahdolliseksi, verkkosivustot saattoivat olla täynnä välkkyviä animaatioita ja tekstitaidetta. Kehittäjät pyrkivät tekemään verkkosivustoja uusimpien trendien mukaan ja unohtivat, että sivustojen tulisi toimia myös vanhemmilla selaimilla. HTML ja CSS ovat vikasietoisia, eli selaimen kohdatessa jonkin tuntemattoman tyylimääritteen tai HTML-elementin se jätetään huomiotta ja siirrytään seuraavaan kohtaan. Näin vanhemmillakin selaimilla voidaan selata toimivaa verkkosivustoa, vaikka kaikki elementit eivät näkyisikään niin kuin niiden olisi tarkoitus. [6, s. 5–6.]

Toisin kuin HTML ja CSS, JavaScript ei ole vikasietoinen. Jos selain kohtaa jonkin tuntemattoman funktion koodia käsiteltäessä, koodin käsittely keskeytyy ja selain ilmoittaa virheen tapahtuneen. Tämän vuoksi JavaScript-koodi tulisi kirjoittaa niin, että virheistä voidaan palautua tai ne voidaan ennakoida, jolloin tuntematon funktio ei keskeytä koodin suorittamista ja heikennä tai estä sivuston toimintaa. Verkkokehitysteknologioiden kehittyessä vikasietoisuus kuitenkin usein unohdettiin, koska kaikkia uusia kiinnostavia ominaisuuksia haluttiin käyttää heti. Verkkosivut kehitettiin uusimpien trendien mukaan, jolloin myös yhteensopivuus vanhempien selainten kanssa jäi helposti taka-alalle. [6, s. 5–6.]

2.2 Helsinki Summer Schoolin vanhat verkkosivut

Helsinki Summer Schoolin vanha sivusto (kuva 2) kehitettiin vuonna 2008, joten se alkoi olla jo hyvin vanhentunut ja kaipasi kipeästi uudistusta niin ulkoasuun kuin hallintaankin. Sivusto oli rakennettu kiinteään 750 pikselin leveyteen, eikä se ollut responsiivinen, eli sivuston sisältö ja leveys eivät mukautuneet selainikkunan koon mukaan. Responsiivisia verkkosivuja on alettu tehdä runsaammin 2010-luvun alussa, kun mobiililaitteiden käyttö on yleistynyt ja käyttäjät ovat alkaneet vaatia parempaa käyttökokemusta myös pienellä näytöllä. Vuonna 2008 rakennettu sivusto on siis tehty aikana, jolloin responsiivisuutta ei osattu ottaa kehitystyössä huomioon. Ennen responsiivisuuden nousua verkkosivujen sisällölle annettiin tietty kiinteä leveys, joka valittiin pienimpien tietokonenäyttöjen mukaan. Kiinteän leveyden tuli olla niin pieni, että koko sivusto mahtuisi pienelle näytölle ilman tarvetta vierittää sivua sivusuunnassa. [7.]

Kuva 2. Helsinki Summer Schoolin vanhan sivuston etusivu [8].

Ennen responsiivisen web-suunnittelun yleistymistä verkkosivujen käyttäminen mobiililaitteilla oli hankalaa ja käyttäjän täytyi suurentaa sivua manuaalisesti voidakseen lukea

tekstisisältöä ja navigoida sivuilla. Tämä ongelma pyrittiin ratkaisemaan siten, että sivuista tehtiin erillinen mobiiliversio, jonne käyttäjä ohjattiin sivuston tunnistaessa mobiililaitteen. Erillisen mobiilisivuston tekeminen kuitenkin aiheutti verkkokehittäjälle ja suunnittelijalle kaksinkertaisen työmäärän, koska mobiilisivut tehtiin alusta asti uusiksi. Responsiivisen suunnittelun yleistyttyä verkkokehittäjän työ on helpottunut, sillä kehitteillä olevasta sivustosta saa mobiililaitteissa toimivan version verrattain pienellä lisätyöllä, esimerkiksi vain kiinnittämällä huomiota elementtien joustavaan leveyteen ja lisäämällä mediakyselyitä. [7.] Mediakysely on CSS:ssä käytettävä tekniikka, joka mahdollistaa erilaisten tyylien määrittämisen selainikkunan leveyden tai esimerkiksi mobiililaitteen orientaation mukaan.

Helsinki Summer Schoolin vanhat sivut olivat myös vanhentuneet hallinnan ja ylläpidon kannalta. Vanha sivusto oli rakennettu Smarty-sivupohjamoottorin päälle, ja vaikka suurta osaa sivujen sisällöstä oli mahdollista päivittää, asiakas toivoi, että lähes kaikki sivuston sisällöstä olisi muokattavissa mahdollisimman yksinkertaisesti ja helposti. Tästä syystä uudet sivut rakennettiin WordPress-sisällönhallintajärjestelmän päälle niin, että kaikkea tekstisisältöä ja lähes kaikkia elementtejä voidaan muokata WordPress-hallinnan kautta. WordPress päivittyy jatkuvasti, joten sivuston hallinta ei vanhene ja WordPress korjaa ilmenneet virheet ja mahdolliset haavoittuvaisuudet päivityksien yhteydessä, joten ongelmat saadaan korjattua vaivattomasti.

2.3 Verkkosivuston yhteensopivuus ja mobiiliystävällisyys

Nykyään yksi verkkokehityksen tärkeimmistä osista on selainyhteensopivuus. Verkkosivustoja ja -sovelluksia testataan useilla eri selaimilla, niiden eri versioilla sekä eri mobiililaitteilla, jotta voidaan tarjota mahdollisimman hyvä käyttökokemus mahdollisimman laajalle yleisölle. Selaimista ja mobiililaitteista tulee uusia päivitettyjä versioita kaiken aikaa, esimerkiksi vuonna 2015 Google Chrome -selaimesta ilmestyi yhteensä kahdeksan merkittävää päivitystä [9]. Näin ollen myös testaustyön määrä kasvaa jatkuvasti. Kaikista vanhimpia selaimia ei kuitenkaan enää voida tukea, eikä niitä myöskään kannata tukea, sillä niitä käyttää häviävän pieni prosentti kaikista verkkosivun potentiaalisista vierailijoista. Usein sivuston muokkaaminen yhteensopivaksi vanhan selaimen kanssa voi viedä huomattavasti aikaa, jopa enemmän kuin potentiaalisten käyttäjien yhteenlaskettu sivunkatseluaika kyseisellä selaimella. Verkkokehittäjien avuksi on onneksi kehitetty

useita apuvälineitä, jotka mahdollistavat esimerkiksi mediakyselyiden käyttämisen vanhoissa selaimissa, jotka eivät tue niitä automaattisesti. Tyylimääriteapuvälineiden lisäksi on kehitetty myös erilaisia JavaScript-kirjastoja, jotka lisäävät vanhoihin selaimiin tuen sellaisille JavaScript-metodeille, joita selain ei muutoin tue. Nämä kirjastot tavallaan tekevät JavaScriptistä vikasietoisen, koska kohdatessaan tuntemattoman funktion vanha selain ei lopeta koodin toteutusta, vaan hakee kirjastosta niin sanotun selityksen funktiolle.

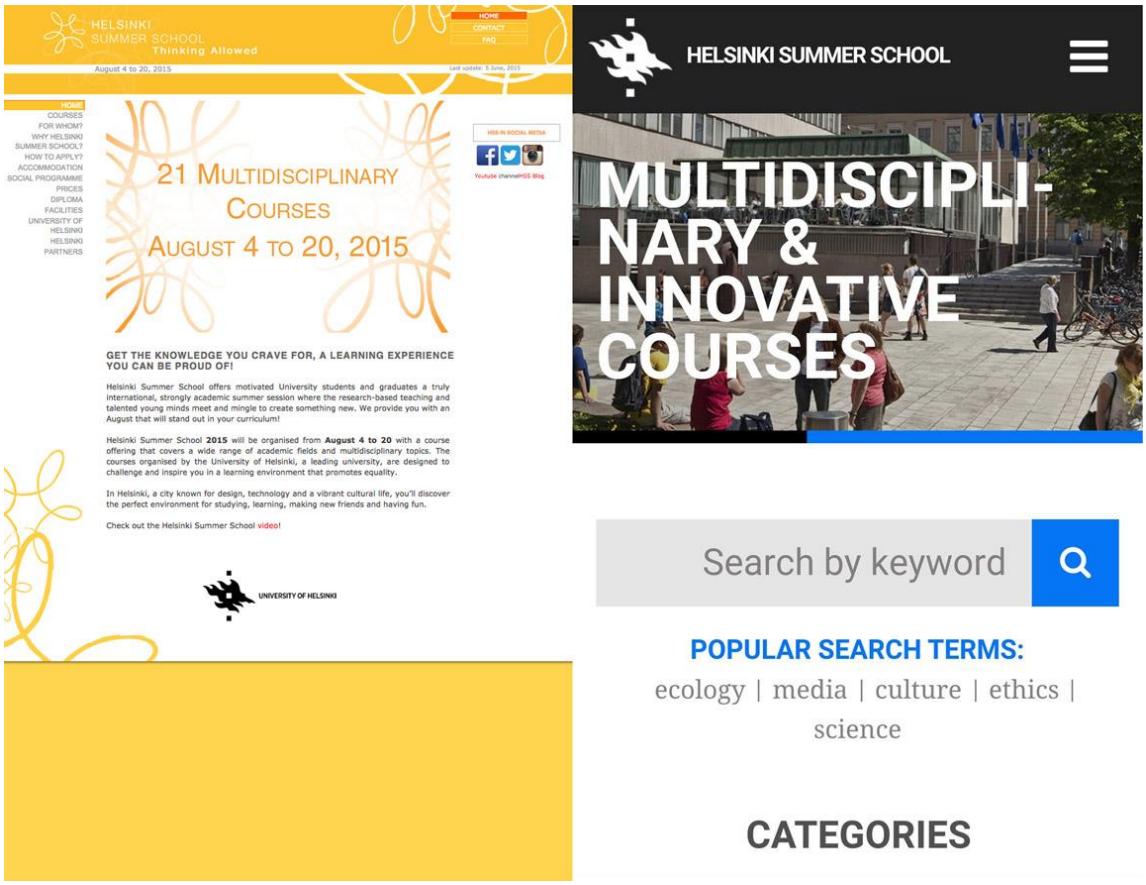
Yksi responsiivisuutta suuresti edistänyt tekijä on Googlen hakukoneen mobiiliystävällinen päivitys. Huhtikuussa 2015 Google julkaisi päivityksen, joka suosii mobiiliystävällisiä verkkosivuja hakujen tuloksissa. Tämä tarkoittaa, että mobiililaitteella hakua tehtäessä responsiiviset tai muilla keinoin mobiiliystävälliseksi tehdyt sivut saavat helpommin hyvän sijoituksen hakutuloksessa, eli sijoittuvat lähemmäs listan alkua. Google tarkkailee sivuissa esimerkiksi seuraavia asioita:

- Sivun tekstisisältö on luettavissa ilman sivun suurentamista manuaalisesti sormin.
- Klikattavat kohteet, eli esimerkiksi linkit ja napit, ovat niin suuria ja erillään toisistaan, että niitä on helppo klikata sormella osumatta väärään kohteeseen.
- Sivusto mukautuu laitteen tai selainikkunan leveyteen, jolloin sivustoa ei tarvitse vierittää horisontaalisesti.
- Sivusto ei sisällä mobiililaitteella toimimatonta sisältöä, esimerkiksi videoita, joita ei voi katsoa.

Jos edellä mainitut ominaisuudet tai osa niistä pätee sivuun, Google näkee sivun mobiiliystävällisenä ja sivu nousee hakutulosten listalla ylemmäs kuin muutoin samalla tasolla oleva sivu, joka ei ole mobiiliystävällinen. Google tarkkailee hakukoneessaan vain yksittäistä sivua, eli koko sivuston ei tarvitse välttämättä olla mobiiliystävällinen, jotta yksittäinen sivu saa niin sanotun mobiilitehostuksen. [10.]

Kuvassa 3 on rinnakkain kuvakaappaukset Helsinki Summer Schoolin vanhan sivuston etusivusta ja uuden sivuston Kurssit-sivusta. Uusi sivusto on responsiivinen, eli se mukautuu ikkunan kokoon, ja napit, eli valikko-nappi ja haku-nappi, ovat tarpeeksi suuria sormin painettavaksi. Vanha sivusto sen sijaan ei ole responsiivinen, vaan sisältö näkyy todella pienenä ilman sivun suurentamista. Sivua täytyy suurentaa manuaalisesti, jotta teksti on luettavissa, mutta silloin sivua täytyy vierittää myös sivusuuntaan. Vaikka sivu

olisi suurennettu luettavaan kokoon, olisivat vasemman reunan sivunavigaation linkit vaikeasti klikattavissa sormella, sillä linkit ovat pieniä ja todella lähellä toisiaan. Näin ollen vanha sivusto ei saisi Googlen mobiilitehostusta, mutta uusi sivusto saisi, koska se täyttää kaikki mobiiliystävällisen sivun vaatimukset.



Kuva 3. Helsinki Summer Schoolin uusi ja vanha sivusto mobiilikoossa [8].

3 Modernin verkkokehityksen työkalut

3.1 Lokaali kehitysympäristö

Lokaali kehitysympäristö mahdollistaa verkkokehityksen ja projektien testaamisen ilman erillistä palvelinta, jolle tiedostot tulee siirtää ja aina muutosten myötä päivittää nähdäkseen todellisen version verkkosivustosta. Verkkokehityksen historian alussa verkkosivustot koostuivat lähinnä HTML- ja CSS-tiedostoista, jolloin verkkosivustoja pystyi testaamaan avaamalla tiedostot suoraan tietokoneelta selaimeen, mutta verkkosivustojen

kehittyessä dynaamisemmiksi työskentelystä tuli monimutkaisempaa. Koska verkkosivustot vaativat palvelinympäristöä toimiakseen tavoitellulla tavalla, työnkulku vaikeutuu ilman lokaalia kehitysympäristöä. [11.]

Pitääkseen lokaalit tietokoneelle tallennetut tiedostot ajan tasalla ja voidakseen testata kaikkia ominaisuuksia ja toimintoja, kehittäjän täytyy siirtää muokatut tiedostot esimerkiksi FTP- tai SFTP-yhteyttä (File Transfer Protocol eli tiedostojen siirtoprotokolla ja Secure FTP eli suojattu tiedostojen siirtoprotokolla) käyttäen aktiiviselle verkkopalvelimelle. Kun tietokoneella on pystyssä lokaali kehitysympäristö, voi sivustoa testata ilman erillistä palvelinta ja tiedostot voi siirtää verkkoon vasta testauksen jälkeen. Tämä mahdollistaa sen, että sivuston tai projektin voi julkaista tai siirtää julkiselle palvelimelle vasta, kun se on täysin valmis. Tällöin kehittäjän ei myöskään tarvitse huolehtia siitä, että aktiivisella palvelimella oleva projekti on kehitysvaiheessa piilotettu niin, että ulkopuolinen käyttäjä ei pysty sitä näkemään. [11.]

Vaikkakin lokaali kehitysympäristö helpottaa kehitystyötä huomattavasti, sillä on myös omat huonot puolensa. Projektin tiedostoista täytyy olla kopiot niin kehittäjän tietokoneella kuin palvelimella, jolloin tiedostojen synkronisointi saattaa aiheutua ongelmalliseksi. On tärkeää, että lokaalit ja palvelimella sijaitsevat tiedostot pysyvät synkronisoituina, eli kaikki lokaaliin versioon tehdyt muutokset siirretään jossain vaiheessa palvelimelle. Tiedostojen synkronisointi on melko yksinkertaista, jos muutetut tiedostot päivitetään palvelimelle heti, kun muutokset on testattu ja hyväksytty. Jos muutoksia sen sijaan tehdään esimerkiksi useisiin eri osiin projektia ja ne halutaan siirtää palvelimelle eri aikoihin, synkronisointi vaikeutuu huomattavasti. [11.]

Projektin kehityksessä saattaa tulla tilanteita, joissa projektiin halutaan tehdä jokin nopea ja väliaikainen muutos, jolloin kehittäjä saattaa kokea helpommaksi ja nopeammaksi vaihtoehdoksi tehdä sen suoraan palvelimelle ja jättää muutokset pois lokaalista versiosta. Tällaisissa tilanteissa ongelmia syntyy, jos muutoksesta tuleekin pysyvä tai projektin lokaaliin versioon tehdään muita muutoksia niin, että tuotantopalvelimelle tehdyt muutokset unohdetaan siirtää myös lokaaliin versioon. Vielä hankalammaksi synkronointi muodostuu, jos lokaalin version ja tuotantopalvelimen lisäksi on käytössä testipalvelin, jota käytetään esimerkiksi kehitysvaiheessa olevien verkkosivujen testaamiseen ja asiakkaalle esittämiseen. Esimerkiksi sisällönhallintajärjestelmien käyttämät tietokannat tuovat myös omat ongelmansa projektin eri kopioiden ajan tasalla pitämiseen. [11.]

Projektien kopioiden synkronisointiin on kuitenkin kehitetty erinäisiä apuvälineitä. Esimerkiksi PhpStorm-ohjelmointiympäristössä on mahdollisuus ottaa käyttöön työkalu, jonka avulla voidaan konfiguroida yhteys yhdelle tai useammalle palvelimelle. Työkalun asetuksiin määritetään palvelimen osoite ja palvelintunnukset, jolloin ohjelmisto voi ottaa yhteyden palvelimeen. Kun yhteys palvelimeen on konfiguroitu, voidaan projektissa olevat tiedostot synkronisoida palvelimella olevien tiedostojen kanssa. Työkalu näyttää, mitä tiedostoja on muokattu, ja käyttäjä voi valita haluamansa tiedostot ja siirtää ne palvelimelle tai vaihtoehtoisesti ladata tiedostoja palvelimelta ja korvata niillä lokaalit tiedostot. Tällainen työkalu helpottaa muutosten siirtämistä ja synkronointia palvelimien ja lokaalin version välillä, koska kaikki tehdyt muutokset on selkeästi merkitty ja käyttäjä näkee heti, mitkä tiedostot poikkeavat toisistaan. Manuaalisesti tiedostoja siirtäessä ja päivittäessä jokin tiedosto saattaa jäädä huomioimatta, mikä pahimmassa tilanteessa aiheuttaa sen, että tuotantopalvelimella oleva projekti lakkaa toimimasta tai menee selkeästi rikki, kun projekti on koko ajan julkisesti nähtävissä.

Esimerkkejä lokaalin kehitysympäristön toteutuksesta

On olemassa monenlaisia eri tapoja toteuttaa lokaali kehitysympäristö: internetistä etsimällä löytyy muun muassa valmiita ladattavia ohjelmistoja ja monia ohjeistuksia eri tarkoituksiin sopivien kehitysympäristöjen käyttöönottoon. Usein helpointa on ladata jokin valmiiksi konfiguroitu paketti, joka vaatii käyttäjältä vain muutamien ohjelmistojen asentamista ja mahdollisimman vähän manuaalista asetusten muuttamista. Tällaisia paketteja on esimerkiksi Scotch Box, joka sisältää kaiken olennaisen perinteisten verkkosivustojen kehittämiseen lokaalisti. Scotch Box on LAMP-kokoelman sisältävä valmiiksi konfiguroitu Vagrant-ohjelmistoa käyttävä paketti. [12.] LAMP on ohjelmistokokoelma, jonka nimi tulee sen sisältämistä komponenteista:

- Linux-käyttöjärjestelmä
- Apache-palvelinohjelma
- MySQL-tietokantarajapinta
- PHP, Perl tai Python-ohjelmointikieli.

Näistä komponenteista muodostuu LAMP-palvelin, joka mahdollistaa lokaalin verkkokehityksen. [12, s. 149.] Scotch Boxin käyttöönotto vaatii vain kahden ohjelmiston asentamisen sekä Scotch Box -tiedostojen lataamisen ja asentamisen versionhallinnasta [12].

Muita valmiita kehitysympäristöpaketteja on esimerkiksi XAMPP, josta on versio kaikille käytetyimmille käyttöjärjestelmille [11].

Helsinki Summer School -sivustoa kehitettäessä käytössä oli OS X -käyttöjärjestelmälle suunniteltu verkkokehitysympäristö. Ympäristö rakennettiin ja konfiguroitiin Chris Mallinsonin kokoaman ohjeistuksen pohjalta. Kun ohjeistuksen mukainen kehitysympäristö on saatu otettua käyttöön, se vaatii käyttäjältä todella vähän työtä uutta projektia luodessa. Jotkin valmiit kehitysympäristöpaketit vaativat aina erinäisten käytettyjen ohjelmistojen käynnistämistä manuaalisesti tietokoneen käynnistyttyä ja ohjelmistojen sulkemista manuaalisesti ennen tietokoneen sammuttamista. Käytetty ympäristö vaatii ainoastaan MySQL-tietokantapalvelimen käynnistämisen komentorivillä, jos tarkoituksena on työskennellä tietokantaa käyttävän projektin parissa, ja on muun muassa tästä syystä hyvin helppokäyttöinen. [14.]

Sivustoa kehitettäessä käytetty lokaali kehitysympäristö vaatii vain muutaman ohjelmiston ja työkalun asentamista, koska OS X -käyttöjärjestelmä sisältää valmiiksi monia kehitykseen tarvittavia ohjelmistoja, kuten Apache-palvelinohjelmiston. Yksi tärkeimmistä ohjelmistoista on Homebrew-paketinhallintajärjestelmä, joka auttaa asentamaan uusia työkaluja ja pitämään ne ajan tasalla ja helposti löydettävissä. Yksi esimerkki Homebrew’n avulla asennettavista lokaalin kehitysympäristön työkaluista on dnsmasq, joka on kevyt kehitystyötä helpottava työkalu. dnsmasq mahdollistaa sen, että jokaisella projektilla on oma uniikki URL-osoite. dnsmasq toimii siten, että kun selaimessa kirjoitetaan osoiteriville tiettyyn päätteeseen (esimerkiksi ”.dev”, lyhenne sanasta development eli kehitys) loppuva URL-osoite, dnsmasq keskeyttää pyynnön ja ohjaa käyttäjän konfiguroinnissa määritettyyn IP-osoitteeseen. Projektit on kansiorakenteen ja dnsmasq:n konfiguroinnin vuoksi hyvä pitää yhden kansion sisällä. Jotta dnsmasq osaa ohjata oikeaan kansioon, dnsmasq:n konfiguroinnin lisäksi Apachen asetustiedostoihin täytyy muuttaa halutun kansion polku ja URL-osoitteen pääte, kuten esimerkkikoodissa 1 näkyy. Kun konfigurointi on valmis, voidaan esimerkiksi Sites-kansioon luoda uusi kansio nimellä helsinkisummerschool ja tämän kansion sisään public-kansio, joka sisältää projektin tiedostot. Kun kansiorakenne on oikein, voidaan selaimessa mennä osoitteeseen helsinkisummerschool.dev, ja sivusto toimii kuin oikealla verkkopalvelimella. [14.]

```
<Virtualhost *:80>
    VirtualDocumentRoot "/Users/pauliinav/Sites/%1/public"
    ServerName sites.dev
    ServerAlias *.dev
    UseCanonicalName Off
  </Virtualhost>
```

Esimerkkikoodi 1. Ote Apachen httpd-vhosts-konfigurointitiedostosta.

3.2 Versionhallinta

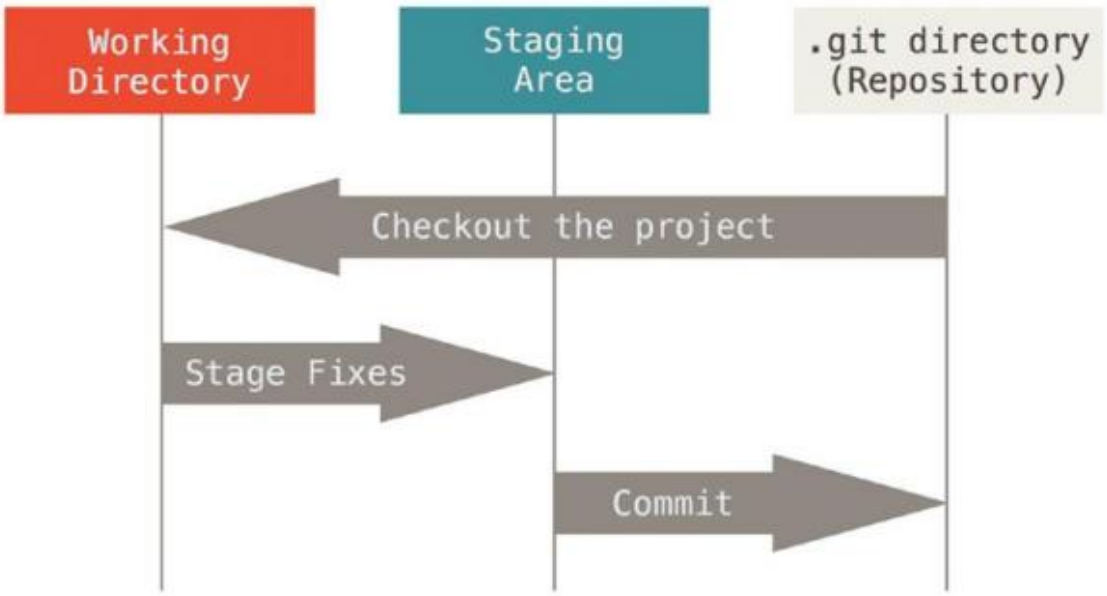
Versionhallinta on järjestelmä, joka tarkkailee tarkkailtaviksi määritettyjä tiedostoja ja niihin tehtyjä muutoksia. Versionhallinnan avulla projektin tiedostoista tallennetaan versioita, ja tämän ansiosta tiedostot voidaan palauttaa vanhempaan versioon. Versionhallintaa käytetään yleisimmin ohjelmoinnissa, mutta sitä voi käyttää lähes jokaisen tiedostotyypin kanssa. Esimerkiksi graafinen suunnittelija voisi käyttää versionhallintajärjestelmää suunnitellessaan verkkosivuston ulkoasua. Tällöin olisi mahdollista palata PhotoShop-tiedoston vanhaan versioon ilman jopa kymmenien eri versioiden tallentamista lokaalisti tietokoneelle. Versionhallinta pitää siis tallessa kaikki sinne tallennetut tiedostoversiot, jolloin käyttäjän tietokoneella tarvitsee olla vain yksi versio tiedostoista. [15, s. 1.]

Verkkokehityksessä versionhallinta on erityisen tärkeää silloin, kun projektin parissa työskentelee useampi kehittäjä samanaikaisesti. Kun yksi kehittäjä tekee muutoksia tiedostoon ja tallentaa sen versionhallintaa, toinen kehittäjä voi ladata muutokset heti tallennuksen jälkeen, jolloin projektin eri kopiot pysyvät mahdollisimman hyvin synkronoituina. Versionhallinta auttaa myös tilanteissa, joissa useampi kehittäjä on tehnyt muutoksia samaan tiedostoon ja muutokset ovat konfliktissa. Tällöin järjestelmä huomauttaa konfliktista muutoksia tallennettaessa ja kehittäjä voi korjata konfliktit ennen tallentamista. Ilman versionhallintaa konflikti voisi jäädä huomaamatta ja tiedosto ylikirjotettaisiin niin, että toisen kehittäjän muutokset katoaisivat kokonaan. Versionhallinta mahdollistaa sen, että konfliktitilanteessa eri kehittäjien muutokset voidaan ottaa huomioon ja yhdistää ne siten, ettei mitään muutoksia epähuomiossa ylikirjoiteta. [15, s. 1.]

Git

Helsinki Summer School -projektissa on käytössä Git-versionhallinta. Git on Linus Torvaldsin ja muiden Linux-käyttöjärjestelmäytimen kehittäjien luoma laajalti käytetty versionhallintajärjestelmä [15, s. 5]. Vaikka projektin parissa työskentelisi vain yksi kehittäjä, on tärkeää, että hieman suuremmassa projektissa käytetään versionhallintaa. Versionhallinta auttaa yksin työskentelevää kehittäjää palaamaan tiedostojen vanhoihin versioihin ongelmitta, ja lähes poikkeuksetta sivustoja muokataan myös sen jälkeen, kun ne on jo julkaistu. Kun projektissa on valmiiksi konfiguroitu versionhallinta, kuka tahansa voi tehdä koodiin muutoksia niin, että kaikki projektin kopiot voidaan pitää helposti ajan tasalla.

Kuva 4 on yksinkertaistettu graafinen esitys Gitin eri osioista. Working directory eli työskentelyhakemisto on lokaali kopio Git-projektista, eli se versio projektista, jota käyttäjä muokkaa omalla tietokoneellaan. Staging area on eräänlainen välivaihe, joka sisältää tiedon siitä, mitä kehittäjä aikoo lisätä Git repositoryyn eli Git-säiliöön. Git-hakemisto tai -säiliö sisältää kaiken tarpeellisen Git-projektista, ja se kloonataan käyttäjän tietokoneelle, kun uusi kopio projektista luodaan. Git-hakemistot voidaan ladata esimerkiksi niille tarkoitetulle verkkosivulle, joista useiden ihmisten on helppo kloonata ne omalle tietokoneelleen. Tällaisia palveluita ovat esimerkiksi suositut GitHub ja Bitbucket. Kun Git-hakemiston lataa omalle tietokoneelleen, tiedostoista muodostuu uusi työskentelyhakemisto, johon kehittäjä tekee muutokset. Halutussa vaiheessa muutokset lisätään välivaiheeseen (kuva 4, nuoli ”Stage fixes”), ja kun kaikki halutut muutokset on siirretty välivaiheeseen, ne lisätään Git-hakemistoon (kuva 4, nuoli ”Commit”). Kun Git-hakemistossa on jonkun toisen lisäämiä muutoksia, joita omasta lokaalista versiosta ei löydy, ne voidaan ladata, jolloin muutokset siirtyvät omaan työskentelyhakemistoon (kuva 4, nuoli ”Checkout the project”). [15, s. 8.]



Kuva 4. Gitin toimintaperiaate [15, s. 8].

Kaikkia tiedostoja ei tarvitse eikä joissain tilanteissa edes kannata lisätä Git-hakemistoon. Esimerkiksi WordPress-sivustojen kansiot sisältävät useita WordPress-tiedostoja, joita kehittäjä ei tule missään vaiheessa muokkaamaan ja jotka muuttuvat jokaisen WordPress-päivityksen yhteydessä. Tällöin on helpompaa, että nämä tiedostot eivät ole versionhallinnassa, vaan kehittäjä hakee WordPress-asennuksen suoraan esimerkiksi WordPressin verkkosivuilta kloonattuaan projektin.

Versionhallinnasta poissa pidettävät tiedostot merkitään gitignore-tiedostoon, jottei tiedostoja voi vahingossa lisätä Git-hakemistoon. Gitignore-tiedostoon merkitään kaikki halutut tiedostot, kansiot ja esimerkiksi tiedostotyypit, jotka halutaan jättää huomioimatta, kun Git tarkastelee tiedostoihin tulleita muutoksia. [15, s. 20.] Helsinki Summer School -projektissa versionhallinnasta on jätetty pois muun muassa kaikki WordPress-asennuksen mukana tulevat tiedostot sekä erinäisiä ohjelmointiympäristöjen luomia projektin konfigurointitiedostoja ja niiden kansioita. Lisäämällä tiedostoja tai kansioita huomiotta jätettävien listalle voidaan Git-hakemiston kokoa pienentää huomattavasti.

3.3 Node.js-ohjelmistokehys

Node.js on palvelinpuolen ohjelmistokehys, joka on rakennettu saman JavaScript-moottorin, v8:n, päälle, jolla myös Google Chrome -selain toimii. Sen voi asentaa esimerkiksi aiemmin mainitun Homebrew’n avulla tai lataamalla asennustiedoston internetistä.

Node.js on kehitetty käytettäväksi raskaiden siirräntäsovellusten ohjelmoinnissa. [16, s. 1, 9.] Siirräntä tarkoittaa keskustelua eli informaation siirtoa tietokoneen ja jonkin ulkopuolisen asian, kuten vaikka tietokoneen näppäimistön välillä [16]. Ohjelmointi tehdään JavaScriptillä. [16, s. 1, 9.]

npm

Helsinki Summer Schoolissa Node.js:ää käytetään npm:n eli Node Package Managerin muodossa. Node Package Manager on paketinhallintajärjestelmä, jonka kautta projektiin voidaan asentaa erilaisia JavaScript-paketteja ja -moduuleja. Node.js-paketit ovat usein myös moduuleja, ja moduulit ovat usein paketteja, joten selkeyden vuoksi viittaan tässä insinöörityössä niin paketteihin kuin moduuleihin käyttämällä ainoastaan sanaa moduuli. npm:ää käytettiin aluksi vain palvelinpuolella käytettävien Node.js-moduulien jakamiseen, mutta nykyään sen kautta voi löytää myös käyttäjäpuolella käytettäviä JavaScript-moduuleja. npm:n pohjimmaisena ajatuksena on mahdollisuus hyödyntää muiden kehittäjien kehittämiä pienehköjä koodipaketteja. npm-moduulien tarkoituksena on ratkaista jokin pieni ongelma mahdollisimman hyvin. Näin ollen useita pieniä moduuleja apuna käyttäen voidaan koota suurempia kokonaisuuksia helposti. [16, s. 9.]

npm tallentaa asennetut moduulit automaattisesti kansioon nimeltä node_modules eli Node-moduulit, mutta npm:n voi myös konfiguroida käyttämään jotain toista kansiota. Asennettujen moduulien tiedot tallentuvat listattuina niin sanottuun manifestitiedostoon, eli JSON-tiedostoon (JavaScript Object Notation eli JavaScript-olion merkintätapa), joka on nimetty package.json. Manifestitiedostosta moduulit voidaan asentaa aina uudelleen suorittamalla komentorivillä käsky ”npm install”. [16, s. 16–17.]

Kun tietyn projektin parissa työskentelee useampia henkilöitä, npm:n avulla kaikki kehittäjät käyttävät samoja työkaluja ja apuvälineitä, jolloin kehitys on suoraviivaisempaa eikä projektin eri versioiden välille tule eroja, jotka aiheuttavat ongelmia tiedostojen synkronoinnissa. Node-moduulit sisältävä kansio kasvaa helposti todella suureksi, vaikka moduuleista käytettävä koodin määrä ei lopullisessa projektissa olisikaan kovin suuri. Kannattaa siis lisätä vain manifestitiedosto versionhallintaan ja lisätä Gitiä käytettäessä moduulikansiot gitignore-tiedostoon. Kun moduulit eivät ole versionhallinnassa, täytyy komentorivillä suorittaa käsky ”npm install” sen jälkeen, kun projekti on kloonattu tietokoneelle. Tällä käskyllä kaikki manifestitiedoston Node.js-moduulit asentuvat tietokoneelle. Kun kehittäjä haluaa lisätä uuden moduulin projektiin, sen tiedot kirjataan manifestiin, ja

kun tiedosto on päivitetty versionhallintaan, voivat muut projektin parissa työskentelevät hakea päivitetyn manifestin ja päivittää oman projektinsa tiedoston mukaisesti. ”npm install” -käsky toimii myös silloin, kun moduuleja on jo asennettu. Käsky vertaa manifestitiedostoa ja moduulit sisältävää kansiota keskenään ja asentaa vain ne moduulit, joita ei löydy kansiosta. [15, s. 20; 16, s. 16–18.]

Node-moduulien määrittely

Helsinki Summer School -projektissa käytetyt Node-moduulit ovat kehityksessä käytettäviä apuvälineitä, joiden tarkoitus on nopeuttaa ja helpottaa kehitystyötä eri tavoin. Esimerkkikoodi 2 on lyhennetty ja karsittu ote projektin manifestitiedostosta. Tiedostossa täytyy määrittää vähintään projektin nimi ja versio. Näiden lisäksi määritetään asennetut paketit ja moduulit kohdassa ”dependencies” eli riippuvuudet. Esimerkkikoodissa riippuvuudet ovat listattuna ”devDependencies”-nimellä, koska ne ovat moduuleja, joita tarvitaan vain kehitysvaiheessa, mutta ei enää valmiissa sovelluksessa. [16, s. 20–21.]

```
{
  "name": "helsinki-summer-school",
  "version": "1.0.0",
  "devDependencies": {
    "bower": "^1.3.3",
    "grunt": "~0.4.1",
    "grunt-contrib-watch": "~0.6.1",
    "grunt-sass": "*",
    "node-sass": "*"
  }
}
```

Esimerkkikoodi 2. Ote package.json-tiedostosta.

Riippuvuudet määritetään moduulin nimellä (kuten ”bower”) ja versionumerolla (kuten ”^1.3.3”). Versionumeron tarkkuuden voi määritellä esimerkiksi pienempi kuin- ja suurempi kuin -merkeillä, jolloin asennettavan version täytyy olla pienempi tai suurempi kuin määritetty versionumero. Aaltoviiva versionumeron edessä merkitsee sitä, että asennettavan version täytyy olla vähintään yhtä suuri kuin määritetty versio, mutta se ei saa olla suurempi kuin seuraava merkittävä versio, eli versionumeron ensimmäinen numero ei

saa muuttua. ^-merkki tarkoittaa sitä, että asennettava versio ei saa muuttaa versionumeron ensimmäistä merkkiä, joka ei ole nolla. Tämä tarkoittaa, että esimerkiksi versionumerossa ”^1.3.3” numero 1 ei saa muuttua, mutta kaksi viimeistä numeroa saavat. Jos määritetty versionumero olisi ”^0.0.5”, mikään version numeroista ei saisi muuttua, eli ainoa vaihtoehto olisi asentaa versionumero ”0.0.5”. Pelkkä asteriski versionumeron kohdalla tarkoittaa sitä, että voidaan asentaa mikä tahansa merkittävä päivitys tai versio. [18.]

3.4 Bower-paketinhallintajärjestelmä

Bower on paketinhallintatyökalu, jonka voi asentaa esimerkiksi npm:n kautta. Bower ja npm ovat molemmat paketinhallintajärjestelmiä, jotka toimivat suurimmaksi osaksi samoin, mutta niiden kautta asennetaan eri tarkoituksiin käytettäviä moduuleja. npm:n kautta asennetaan lähinnä palvelinpuolen kehityksessä käytettäviä Node.js-moduuleja, ja Bowerin avulla voidaan asentaa ohjelmointiprojektiin erilaisia käyttäjäpuolen ohjelmoinnissa käytettäviä moduuleja, kuten JavaScript-lisäosia tai tyylikirjastoja. Nämä moduulit ovat sellaisia, jotka jollain tavalla näkyvät valmiissa ohjelmointiprojektissa sen käyttäjälle, esimerkiksi JavaScript-animaatioiden tai fonttikirjaston kautta. [19, s. 1.]

Kun verkkosivustot ovat ajan kuluessa muuttuneet monimutkaisemmiksi ja dynaamisemmiksi, erilaisten lisäosien tarve on kasvanut. Vielä muutama vuosi sitten ainut sivustolla käytetty JavaScript-kirjasto oli usein jQuery, nykyään käytettyjä pienempiä lisäosia ja kirjastoja saattaa olla jopa kymmeniä, jolloin niiden hallintaan on hyvä olla jokin apuväline. Bower toimii samalla periaatteella kuin npm, eli haluttujen Bower-moduulien nimet ja versionumerot tallennetaan manifestitiedostoon (tässä tapauksessa bower.json-tiedostoon), josta ne voidaan asentaa käskyllä ”bower install”. [19, s. 1–3.] Bowerin ja npm:n käyttämät manifestitiedostot ovat rakenteeltaan samanlaiset.

Koska Bower ja npm ovat molemmat paketinhallintajärjestelmiä, ne toimivat hyvin samalla tavalla. Monet komentorivikäskyistä käyvät molempiin ja suorittavat saman toiminnon, käskyn alkuun tulee vain lisätä joko ”bower” tai ”npm”. Kummankin järjestelmän tapauksessa manifestitiedoston voi luoda käskyllä ”init”, lisättäviä moduuleja voi etsiä käskyllä ”search [hakusana]” ja niitä voi asentaa käskyllä ”install [moduulin nimi]”. Asennus-käskyllä asennettu moduuli ei automaattisesti tallennu manifestitiedostoon, vaan

käskyn perään täytyy vielä lisätä ”--save”-kytkin. [16, s. 16–18; 19, s. 2–5.] Bower-moduulit tallentuvat automaattisesti bower_components-kansioon, mutta asennuskansion voi vaihtaa muokkaamalla Bowerin konfigurointitiedostoa. Kuten npm:n moduulien tapauksessa, myös Bowerin tapauksessa kannattaa asennetut moduulit jättää lisäämättä versionhallintaan.

3.5 Grunt-tehtävänsuorittaja

Grunt on automaattinen JavaScript-tehtävien suorittaja, joka helpottaa verkkokehittäjän työtä huomattavasti. Gruntin voi konfiguroida haluamallaan tavalla muokkaamalla sen asetustiedostoa, ja sille on kehitetty useita erilaisia toimintatapoja. Yleisimmin käytettyjä toimintoja ovat esimerkiksi JavaScript-koodin minimointi eli rivinvaihtojen ja ylimääräisten välien poistaminen ja muuttujanimien lyhentäminen sekä Sass-tyylitiedostojen muuttaminen CSS-tiedostoiksi ja näiden tiedostojen minimointi. [20.] Helsinki Summer Schoolin sivuston kehityksessä Grunt auttoi automatisoimaan monet eri toimenpiteet, jolloin manuaalisen työn määrä väheni huomattavasti.

Gruntin konfigurointi

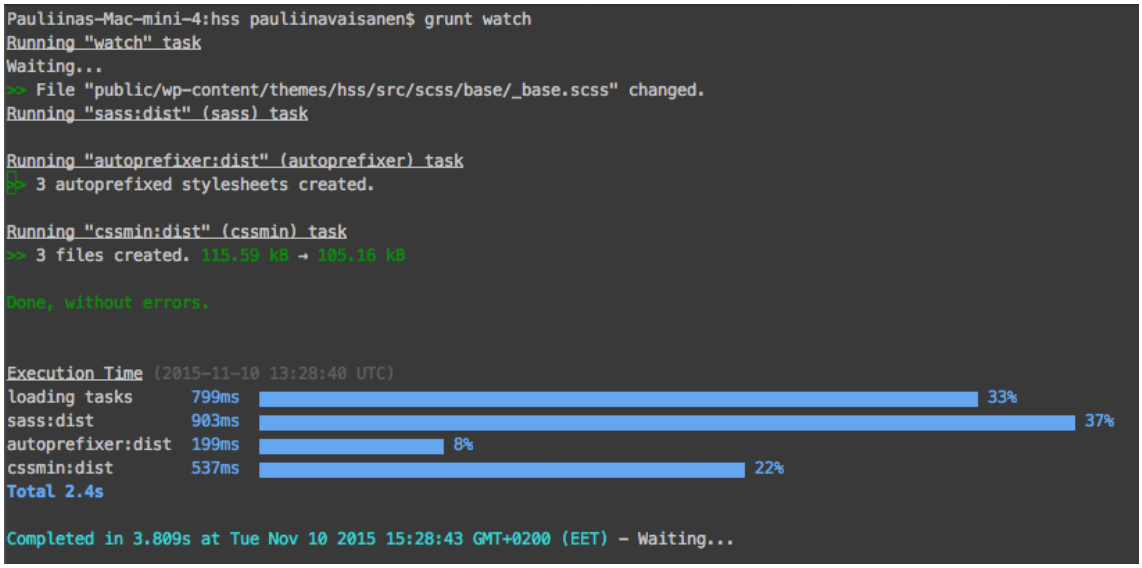
Grunt tarvitsee toimiakseen kaksi tiedostoa, npm:n manifestitiedoston ja Gruntfile.js-tiedoston. npm:n manifestitiedostossa määritetään ne Grunt-moduulit, joita projektissa halutaan käyttää. Gruntfile.js on asetustiedosto, jossa Grunt konfiguroidaan. [20.] Esimerkkikoodi 3 on lyhennetty ja karsittu ote Helsinki Summer School -projektin Gruntfile.js-tiedostosta. Esimerkkikoodissa määritetään npm:n manifestitiedoston sijainti, tarvittavia tiedostopolkuja, eli tässä tapauksessa vain polku WordPress-asennuksen teemakansioon, sekä watch- eli niin sanottu tarkkailukomento. Watch-komento kutsuu kahta Grunt-tehtävää, sass-tehtävää ja js-tehtävää. Sass koostuu kolmesta muusta tehtävästä, jotka määritetään ”tasks”- eli tehtävät-objektissa. Sass-tehtävä muuttaa kaikki ”files”- eli tiedostot-objektissa määritetyt SASS-tiedostot CSS:ksi, autoprefixer-tehtävä lisää tarvittavat etuliitteet selainkohtaisiin tyylimääritteisiin ja lopuksi cssmin-tehtävä minimoi CSS:n eli poistaa kaiken tarpeettoman, kuten kommentit ja ylimääräiset välit. js-tehtävä hakee tehtävät-objektissa määritetyn tiedoston ja suorittaa uglify-tehtävän, joka minimoi JS-koodin muun muassa poistamalla ylimääräiset välilyönnit ja rivinvaihdot sekä vaihtamalla muuttujien nimet mahdollisimman lyhyiksi. Kaikki näiden kahden tehtävän kutsumat tehtävät on myös konfiguroitu erikseen.

```
grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    path: {
        wp: 'public/wp-content/themes/hss',
    },
    watch: {
        sass: {
            files: '<%= path.wp %>/src/scss/**/*.scss',
            tasks: ['sass', 'autoprefixer', 'cssmin']
        },
        js: {
            files: ['<%= path.wp %>/src/js/app.js'],
            tasks: ['uglify:dev']
        }
    }
});
```

Esimerkkikoodi 3. Ote Grunt-konfiguroinnista.

Gruntin käyttö projektissa

Grunt-toimintoja suoritetaan komentoriviltä, esimerkiksi komennolla ”grunt watch” (kuva 5). Watch-toimintoa käytetään yleisesti kehitysvaiheessa havaitsemaan muutoksia tiedostoissa, jolloin Grunt esimerkiksi havaitsee automaattisesti Sass-tiedostoon tehdyt muutokset ja päivittää muutokset käännettyyn CSS-tiedostoon. [20.] Kun watch-toiminto tarkkailee tiedostoihin tehtyjä muutoksia, kehittäjän ei tarvitse miettiä tiedostojen minimointia tai Sassin kääntämistä CSS:ksi, vaan kaikki tämä tapahtuu jatkuvasti taustalla ilman erillisiä toimenpiteitä. Kuvassa 5 näkyy, kuinka komentorivillä on ajettu grunt watch -käsky ja tämän jälkeen tehty muutoksia _base.scss-tiedostoon, minkä jälkeen Grunt kääntää muutokset CSS:ksi ja lisää ne minimoituun tyylitiedostoon, johon kaikki sivuston tyylit on yhdistetty. Minimoidut tyylitiedostot ja JavaScript-tiedostot on hyvä tallentaa eri kansioon kuin alkuperäiset tiedostot, koska silloin kansio- ja tiedostorakenne pysyvät siistimpänä ja minimoitujen tiedostojen kansion voi helposti pitää poissa versionhallinnasta. Yleisesti ottaen minimoitujen tiedostojen kansiota on turha lisätä versionhallintaan, sillä kaikki minimoitavat tiedostot ovat versionhallinnassa alkuperäisessä muodossa.



Kuva 5. Grunt watch -komento suoritettuna komentorivillä.

Grunt auttaa verkkosivuston optimoinnissa, sillä mitä pienempiä ja mitä vähemmän tiedostoja sivusto hakee, sitä nopeammin se toimii. Jokainen rivinvaihto ja ylimääräinen välilyönti suurentavat tiedostokokoa, joten suurissa tyyli- tai JavaScript-tiedostoissa niiden osa koko tiedoston koosta voi olla huomattava. Myös tiedostojen määrä voi hidastaa sivustoa merkittävästi, joten vaikka kehitysvaiheessa useaan osaan jaetut tyylitiedostot ovat helpompia käsitellä, ne on lopuksi hyvä yhdistää vain yhdeksi tiedostoksi, jolloin sivusto lähettää vain yhden HTTP-pyynnön hakiessaan tiedoston. Helsinki Summer Schoolin Grunt-konfiguroinnissa on mukana myös sivuston optimoinnissa auttava imagemin-moduuli, joka optimoi kuvat ja tekee tiedostoista mahdollisimman pienet. Jos sivustolla käytetään paljon pieniä kuvakkeita, kannattaa ne optimoinnin lisäksi siirtää yhteen ainoaan kuvatiedostoon, ja hakea oikea osa kuvaa tyylimäärittеillä. Näin ollen sivuston ei tarvitse tehdä useaa HTTP-pyyntöä kuvia hakiessa, vaan kaikki kuvat haetaan yhdellä pyynnöllä. [21.]

Tiedostojen minimoinnin ja yhdistämisen lisäksi on olemassa useita hyödyllisiä Grunt-moduuleja, jotka eivät suoraan muokkaa lopullisia tiedostoja. Tässä projektissa käytössä on muun muassa JSHint-moduuli, joka on JavaScript-koodia tarkistava työkalu. JSHint huomaa mahdolliset virheet ja ongelmat jo kehitysvaiheessa muutoksia tehtäessä ja ilmoittaa niistä. Usein JavaScript-virheet huomataan vasta selaimessa projektia testatessa. Selaimen konsolin antamat virheilmoitukset eivät ole kovin tarkkoja, joten virheen etsiminen ja korjaaminen saattaa viedä huomattavasti aikaa. JSHint kuvailee virheet tar-

kemmin ja kertoo niiden tarkat sijainnit. Virheiden lisäksi JSHint huomauttaa myös mahdollisista ongelmista, jotka on hyvä korjata jo ennen kuin niistä muodostuu varsinaisia virheitä. [22.] Kuvassa 6 JSHint varoittaa virheestä, jossa if else -lauseessa else-sana on toistettu kaksi kertaa. Virheviesti kuvailee virheet tarkasti ja kertoo rivin ja sarakkeen tarkkuudella, mistä virhe löytyy, jolloin sen korjaaminen on helpompaa kuin selaimen konsolissa näkyvän virheviestin perusteella.

```
pauliina-vaisanen:hss pauliinav$ grunt watch
Running "watch" task
Waiting...
>> File "public/wp-content/themes/hss/src/js/app.js" changed.
Running "jshint:files" (jshint) task

public/wp-content/themes/hss/src/js/app.js
  line 293  col 20  Expected an identifier and instead saw 'else'.
  line 293  col 20  Expected an assignment or function call and instead saw an expression.
  line 293  col 24  Missing semicolon.

✖ 3 problems

Warning: Task "jshint:files" failed. Use --force to continue.

Aborted due to warnings.
```

Kuva 6. JSHintin virheviesti osoittaa JavaScript-tiedoston virheeseen.

Toinen esimerkki hyödyllisestä Grunt-moduulista on Grunt Copy -moduuli, joka kaikessa yksinkertaisuudessaan kopioi tiedostoja. Kopioint-moduuli on erityisen hyödyllinen silloin, kun projektissa on esimerkiksi käytössä jokin Bower-moduuli, jonka sisältämiä tiedostoja tai koko moduuli halutaan kopioida sellaisenaan toiseen kansioon. Bower-moduulien kansio on usein projektin juuressa, eikä moduulikansiota välttämättä siirretä tuotantopalvelimelle, jolloin tiedostot täytyy kopioida sellaiseen sijaintiin, mistä valmis verkkosivusto tai -sovellus voi ne hakea. Esimerkiksi WordPress-teemassa tyylitiedostot, fontit ja muut vastaavat on helpompi hakea teema-kansiosta kuin palvelimen juuresta. Tiedostojen kopiointi Gruntin kautta sen sijaan, että ne kopioitaisiin manuaalisesti haluttuun kansioon, mahdollistaa sen, että kopioidut tiedostot voidaan pitää ajan tasalla silloinkin, kun Bower-moduuli päivittyy ja kopioitavat tiedostot muuttuvat. Jos tiedostot kopioitaisiin manuaalisesti, täytyisi kopiointi tehdä manuaalisesti aina, kun moduuli päivittyy. [23.] Helsinki Summer School -projektissa Grunt Copya käytettiin Bowerin kautta asennetun fonttikirjaston kopioimiseen teema-kansioon.

3.6 Sass-esikääntäjä

Sass (Syntactically Awesome StyleSheets eli syntaktisesti mahtavat tyylitiedostot) on esikääntäjä, joka helpottaa ja nopeuttaa tyylitiedostojen kirjoittamista. Sass mahdollistaa esimerkiksi muuttujien käytön ja tyylimääritteiden sisentämisen, jolloin tyylimääritteitä kirjoittaessa voidaan välttyä turhalta määritteiden toistolta. Sassissa on kaksi eri syntaksia, vanhempi, jota kutsutaan vain Sassiksi, sekä uudempi Scss (Sassy CSS), jota Helsinki Summer School -projektissa käytetään. Syntaksit eroavat muun muassa tyylimääritteiden sisentämisessä, sillä Scss:ssä käytetään aaltosulkeita, kun taas Sassissa ei käytetä sulkeita vaan pelkkää määritteiden sisentämistä välilyönneillä tai sarkaimella. Sassissa ei myöskään käytetä puolipistettä määritteen lopussa, vaan rivinvaihto erottaa määritteet toisistaan. Sass-syntaksia käyttävät tiedostot loppuvat .sass-päätteeseen ja Scss-syntaksia käyttävät loppuvat .scss-päätteeseen. Kun käytetään Scss-syntaksia, tyylitiedostot kirjoitetaan ensin Scss-tiedostoiksi, minkä jälkeen ne muunnetaan asennettua Sass-moduulia käyttäen CSS-tiedostoiksi lopulliseen muotoon, jotta tyylitiedostot ovat luettavissa selaimella. [24.]

Sass-muuttujat toimivat samoin kuin muuttujat muissakin ohjelmointikielissä, eli niille määritetään nimi ja arvo. Arvon tulee olla tyypiltään sellainen, jota Sass tukee, esimerkiksi värikoodi tai numero. Jos muuttujan arvo on tyypiltään vääränlainen, Sassin muuntamiseen käytettävä moduuli varoittaa virheestä ja keskeyttää tyylitiedoston muuntamisen. Muuttujien käyttäminen vähentää toistoa ja helpottaa esimerkiksi värikoodien käyttöä. Kun jokin värikoodi (esimerkiksi #0575f3) määritetään muuttujan (esimerkiksi $sininen) arvoksi, voidaan tyylimääritteitä kirjoittaessa käyttää vain muuttujaa $sininen eikä varsinaista värikoodia tarvitse muistaa ulkoa tai kopioida sitä toisesta kohtaa. Tällöin myös värikoodin vaihtaminen on helpompaa, sillä se täytyy muuttaa vain muuttujan arvoon, eikä jokaisen tiedoston jokaiseen kohtaan, jossa väriä on käytetty. [24.]

Muita Sassin hyödyllisiä ominaisuuksia ovat esimerkiksi funktiot ja mixinit. Funktioiden täytyy aina palauttaa jokin määrite tai määritteen arvo, jotta ne toimivat tyylimääritteinä. Sass-funktioissa voi käyttää myös muista ohjelmointikielistä tuttuja silmukoita. Niitä voi käyttää esimerkiksi tekemään laskutoimituksen, joka määrittää elementin leveyden. [24.]

Mixinit voivat sisältää useita määritteitä, ja ne kirjoitetaan samoin kuin tavallisen valitsimen tyylimääritteet, mutta mixineissä valitsimen nimen tilalla lukee ”@mixin [mixinin

nimi]”. Mixin lisätään valitsimen määritteeksi käyttämällä ”@include [mixinin nimi]” -ominaisuutta. Mixinin voi kirjoittaa myös vastaanottamaan argumentin, joka voi olla esimerkiksi värikoodi. Mixinit ovat hyödyllisiä muun muassa, kun useassa kohtaa käytetään samoja tyylimääritettä, jolloin on helpompaa kirjoittaa ne kerran mixiniksi ja käyttää luotua mixiniä eri valitsimien määritteissä. Esimerkiksi laatikkovarjon kirjoittaminen eri selaimilla toimivaksi vaatii monta riviä määritteitä, jolloin kannattaa tehdä niistä mixin. Varjo-mixinin voi kirjoittaa vastaanottamaan argumenttina esimerkiksi värikoodin, jota käytetään varjon värinä. [24.]

Esimerkkikoodissa 4 on aluksi lyhennetty ja karsittu ote _hero.scss-tiedostosta ja sen alapuolella samat tyylimääritteet CSS:ksi käännettynä ja Gruntilla minimoituna. Scss-tiedoston esimerkissä näkyy Scss-syntaksin mukainen sisennys, &-merkin käyttö valitsimessa sekä muuttujan ja mixinin käyttö. CSS:ää käyttäessä eri valitsimia eli esimerkiksi luokkaa (esimerkkikoodissa ”.hero”) tai elementin tyyppiä (esimerkkikoodissa ”h3”) ei voi sisentää toistensa sisään, jolloin valitsimien nimiä täytyy toistaa useaan otteeseen (esimerkkikoodissa app.css-tiedostossa ”.hero .hero-content h3.black”). Scss:ssä sisentäminen mahdollistaa sen, että valitsimia ei tarvitse toistaa määritteitä kirjoittaessa, vaan eri valitsimien kirjoittaminen toistensa sisään tuo saman lopputuloksen. Sisentämisen lisäksi Scss:ssä voi käyttää muutamia erikoismerkkejä valitsimen kirjoittamisessa, esimerkiksi sisennetyssä valitsimessa käytetty &-merkki aiheuttaa sen, että alkuun liitetään sen valitsimen nimi, jonka sisällä &-merkillä alkava valitsin on. Esimerkkikoodissa 4 ”&.black”-valitsin muuntuu muotoon ”h3.black”, eli viitataan h3-elementtiin, jolla on ”black”-luokka. Jos &-merkki puuttuisi valitsimesta, se muuntuisi muotoon ”h3 .black”, jolloin viitattaisiin h3-elementin sisällä olevaan elementtiin, jolle on määritetty ”black”-luokka. [24.]

_hero.scss

```
.hero {
  background-size: cover;
  .hero-content {
    min-height: 433px;
    h3 {
      @include font-size(76px);
      &.black {
        color: $black;
      }
    }
  }
}
```

app.css

```
.hero{background-size:cover}.hero .hero-content{min-
height:433px}.hero .hero-content h3{font-size:76px;font-
size:4.75rem }.hero .hero-content h3.black{color:#000}
```

Esimerkkikoodi 4. Ote _hero.scss-tiedostosta ja sen tuottamista minimoiduista CSS-määritteistä.

Esimerkkikoodissa 4 käytetty ”font-size”-mixin ottaa argumenttina vastaan fonttikoon pikseleinä ja tulostaa ensiksi fonttikoko-ominaisuuden pikseliarvolla ja seuraavalle riville fonttikoon, jonka arvona on laskutoimituksen tuloksena saatu arvo rem-yksikköinä. Rem-yksikkö on verrannollinen juuritason, eli html-elementin, fonttikokoon. Rem-yksikkö kertoo verkkosivun juuritason eli html-elementin fonttikoon arvollaan, esimerkiksi 1 rem on 100 % ja 1,5 rem 150 % juuritason fonttikoosta. Fonttikoon määrittäminen rem-yksikköinä mahdollistaa muun muassa sen, että jos laitteessa on käytössä helppokäyttötoimintona fontin suurennus, myös sivuston fonttikoot mukautuvat laitteen määritysten mukaan. Mixiniä käytetään fonttikoon määrittämiseen siksi, että jotkin selaimet eivät tue CSS3:n mukana tullutta rem-yksikköä, joten varmistuksena fonttikoko määritetään myös pikseleinä. Kun tyylitiedostossa on määritetty saman valitsimen samalle tyyliominaisuudelle useampia arvoja, viimeiseksi määritetty ylikirjoittaa muut määritteet ja se otetaan käyttöön. Tällöin ne selaimet, jotka tunnistavat rem-yksikön, ottavat käyttöön viimeiseksi

määritetyn rem-arvon, ja ne selaimet, jotka eivät tue rem-yksikköä, ottavat käyttöön pikseliarvon. [25.] Tämä on esimerkki CSS:n vikasietoisuudesta.

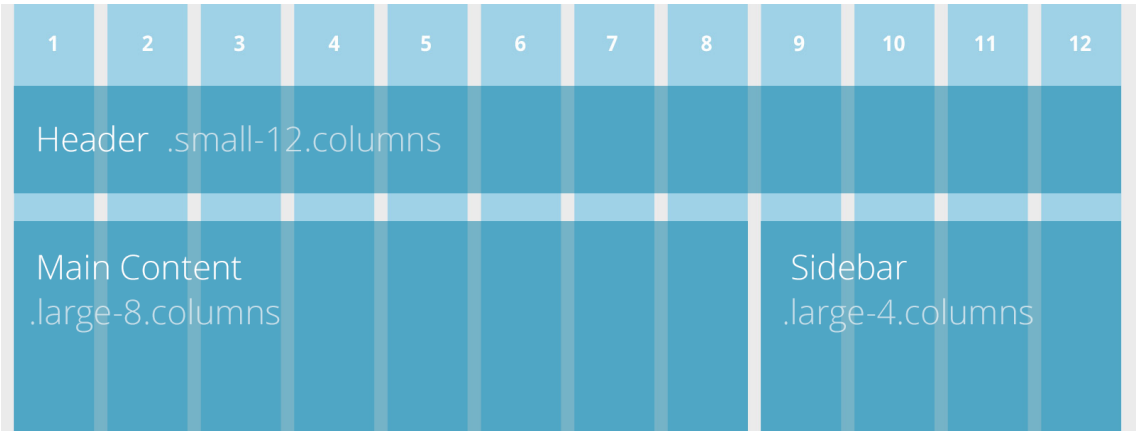
Helsinki Summer School toimii Helsingin yliopiston alaisuudessa, joten uutta sivustoa suunniteltaessa haluttiin tehdä sivustosta yhtenäinen Helsingin yliopiston verkkosivuston kanssa. Verkkokehityksessä yhtenäistämisen avuksi otettiin käyttöön Helsingin yliopiston graafinen ohjeistus. Graafinen ohjeistus on mahdollista asentaa Bower-moduulina projektiin, jolloin sitä voi käyttää kuin mitä tahansa moduulia. Kun graafinen ohjeistus on asennettu moduulina, siitä voi ottaa käyttöön vain ne tiedostot, joita projektissa tarvitaan, jolloin ohjeistuksen viemä tila voidaan minimoida. Kun moduulista valitut tiedostot on lisätty projektin konfigurointiin, voidaan haluttuja komponentteja käyttää projektissa sellaisenaan tai niitä voidaan tarvitessa myös muokata projektin tyylitiedostoissa koskematta varsinaiseen moduuliin. Helsinki Summer School -projektissa graafisesta ohjeistuksesta on käytetty muun muassa värimuuttujia ja nappien tyylejä, joita on hieman modifioitu värien ja tekstityylien osalta. Kun graafinen ohjeistus kehitetään niin, että sitä voi käyttää ulkopuolisessa projektissa esimerkiksi Bower-moduulina, on mahdollista pitää eri projektien ulkoasu yhtenäisenä ja samalla vähentää vaadittavaa työtä.

3.7 Foundation-ohjelmistokehys

Verkkokehityksen helpottamiseksi on kehitetty monenlaisia eri tarkoituksiin käytettäviä ohjelmistokehyksiä. Helsinki Summer Schoolin sivustoa kehitettäessä käyttöön otettiin front-end-ohjelmistokehys, eli ohjelmistokehys, joka auttaa käyttöliittymän kehittämisessä. Suosituimpia käyttöliittymäohjelmistokehyksiä tällä hetkellä ovat Bootstrap ja Foundation, joista käyttöön valittiin Foundation. Bootstrap ja Foundation tarjoavat hyvin pitkälti samat ominaisuudet eri paketeissa, eikä niitä voi asettaa paremmuusjärjestykseen. Valinta näiden kahden välillä on siis puhtaasti mielipide- ja tottumuskysymys. Foundation valittiin tähän projektiin muun muassa siksi, että se sisältää kaikki tämän projektin vaatimat ominaisuudet. Foundation tarjoaa valmiit komponentit, joita verkkokehittäjä voi käyttää vaivattomasti ja yksinkertaisesti responsiivisten verkkosivustojen rakentamiseen.

Responsiivisen verkkokehityksen yleistyttyä yhdeksi verkkosivuston tärkeimmistä tyylikomponenteista on muodostunut elementtien ruudukkoasettelu. Foundationissa on ole-

tuksena kahdentoista sarakkeen ruudukko (kuva 7), mutta määrän voi itse vaihtaa projektiin sopivammaksi. Ruudukkoasettelu tarkoittaa, että verkkosivun sisältöalue jaetaan samankokoisiin sarakkeisiin, joita yhdistämällä saadaan erilevyisiä elementtejä. Sisältöalueen enimmäisleveys on Foundationissa oletuksena 1 200 pikseliä, mutta määritteen voi ylikirjoittaa omassa tyylitiedostossa. Elementtien leveys määritetään sarakkeina, eli elementti voi olla esimerkiksi kuusi saraketta leveä. Leveydet on määritetty tyylitiedostoihin prosentteina, jolloin ne mukautuvat aina selainikkunan leveyteen. Sarake-elementin voi myös asettaa toisen sarake-elementin sisään, jolloin sisemmän elementin enimmäisleveys määrittyy ulomman sarakkeen mukaan. Jos ulompi elementti on kuuden sarakkeen levyinen ja sen sisään lisätään elementti, joka on myös kuuden sarakkeen levyinen, on sisempi elementti puolet ulomman sarakkeen leveydestä eli neljäsosan uloimman elementin leveydestä. [26.]



Kuva 7. Foundationin kahdentoista sarakkeen ruudukko [27].

Kuvassa 7 näkyy Header- eli ylätunniste-elementti, jolle on annettu leveydeksi kaksitoista saraketta, eli tässä tapauksessa koko sisältöalueen leveys. Main Content- eli pääsisältöelementin leveydeksi on annettu kahdeksan saraketta ja Sidebar- eli sivupalkkielementin leveydeksi neljä saraketta, eli nämä elementit vievät yhteensä kaikki kaksitoista saraketta, jotka mahtuvat vierekkäin. Elementin leveys määritetään luokka-attribuutilla HTML-tiedostossa, esimerkiksi kuvassa 7 pääsisältöelementille on asetettu luokat ”large-8” ja ”columns”. Foundationin valmiissa tyylitiedostoissa on määritetty eri luokkien leveydet ja muut ruudukkoasettelun vaatimat tyylimääritteet. Sarakkeiden määrän sisältävälle luokalle, esimerkiksi ”large-8” ja ”small-12”, on tyylitiedostoissa määritetty vain leveys, ja ”columns”-luokka sisältää kaikki muut tyylimääritteet, jotka elementti tarvitsee asettuakseen ruudukkoon. [26.]

Foundation jakaa ikkunakoot mediakyselyillä viiteen kokoon, pieneen, keskikokoiseen, suureen, extra-suureen ja kaikista suurimpaan. Kokoihin jakaminen tehdään breakpointien eli pysäytyspisteiden avulla. Pysäytyspiste on mediakyselyllä asetettu selainikkunan leveys, jonka kohdalla tyylimäärite muuttuu. Esimerkiksi Foundationissa pienin pysäytyspiste on 640 pikseliä, eli sivusto noudattaa samoja tyylimääritteitä aina kun selainikkunan leveys on alle 640 pikseliä. Jos elementille on määritetty luokat ”small-12”, ”medium-6” ja ”large-3”, on elementin leveys pienimmässä ikkunakoossa kaksitoista saraketta, eli elementti on kokoleveä, keskikokoisessa ikkunakoossa kuusi saraketta, eli elementti on leveydeltään puolet sisältöalueesta, ja suuressa ikkunakoossa kolme saraketta, eli elementin leveys on neljäsosan sisältöalueen leveydestä. Foundation on suunniteltu ”mobiili edellä”, eli rakenne on suunniteltu aluksi pienille mobiililaitteiden näytöille ja tätä rakennetta muokataan aina isompaan näyttöön sopivaksi. Pysäytyspisteitä käytetään siis muuttamaan pienelle näytölle määritettyjä tyylejä isompaan näyttökokoon siirryttäessä. [26.]

Tyylimääritteiden lisäksi Foundation tarjoaa erilaisia JavaScript-apuvälineitä, jotka auttavat kehittäjää pienissä ja suurissa asioissa. Helsinki Summer Schoolin verkkosivuilla on käytetty näistä muun muassa Orbit Slider -karusellia, jota voi käyttää esimerkiksi kuva- tai tekstikarusellien luomiseen. Helsinki Summer Schoolin sivuilla kuvakarusellia käytetään esimerkiksi kurssisivuilla kuvasisällön näyttämiseen. Erilaisia kuvakarusellilisäosia on tarjolla runsain määrin, mutta Orbit Slider on varta vasten Foundationille suunniteltu, ja se on helppo ottaa käyttöön, jos projektissa on jo valmiiksi käytössä Foundation. [28.]

Kaikki Foundationin tarjoamat apuvälineet eivät ole suuria kokonaisuuksia tai ominaisuuksia, vaan osa niistä on pieniä ja melko yksinkertaisia apuvälineitä. Ne helpottavat kehittäjän työtä, koska aikaa ei tarvitse käyttää jonkin pienen yksityiskohdan kehittämiseen, vaan projektiin voidaan vain lisätä JavaScript-apuväline, joka tekee saman asian. Tällaisia pieniä apuvälineitä on esimerkiksi Equalizer. Se tasoittaa elementit eli mahdollistaa sen, että vierekkäiset elementit ovat samankorkuisia riippumatta niiden sisällöstä. Se tarkkailee elementtejä ja määrittää korkeimman elementin korkeuden kaikille tarkkailluille elementeille. Näin elementit pysyvät tasakorkuisina, vaikka niiden sisällön määrä ja korkeus vaihtelisi suurestikin elementtien välillä, ja elementtien asettelu pysyy siistimpänä. Tällaisen ominaisuuden kehittäminen itse olisi melko yksinkertaista, mutta aikaa vievää, joten koska apuväline on valmiina tarjolla, voidaan aika käyttää jonkin uuden kehittämiseen, jota ei ole valmiina tarjolla. [29.]

4 Sosiaalisen median hyödyntäminen verkkosovelluksessa

4.1 Sosiaalinen media markkinoinnin työkaluna

Sosiaalinen media on melko tuore nykyajan ilmiö, ja moni yritys vielä harjoittelee sen käyttöä ja hyödyntämistä. Analytiikka- ja markkinointityökaluja tulee kaiken aikaa lisää, ja valmiit työkalut muuttuvat, koska myös sosiaalisen median kanavat muuttuvat ja kehittyvät jatkuvasti. Sosiaalinen media on kuitenkin nykyään enemmänkin pakollinen kuin valinnainen työkalu yrityksille, jotka haluavat luoda itsestään luotettavan ja avoimen mielikuvan. Sisällön julkaiseminen esimerkiksi Facebookissa tulee yritykselle hyvin halvaksi ja saattaa ilman mainostustakin saavuttaa yllättävän laajan yleisön. Kun yrityksen Facebook-sivusta tykänneet käyttäjät tykkäävät, kommentoivat ja jakavat sisältöä, julkaisu saa näkyvyyttä, sillä ilmoitus interaktiosta ilmestyy käyttäjän ystävien aikajanoille, vaikka he itse eivät olisi tykänneet sivusta. Myös Facebook-julkaisun mainostaminen tulee halvemmaksi ja mainoksen luominen vie luultavasti myös vähemmän aikaa ja vaivaa kuin esimerkiksi mainoskampanjan luominen lehteen. [30, s. 1–2.]

Palveluja tarjoavalle yritykselle yksi tärkeimmistä sosiaalisen median käyttötarkoituksista on rehellinen kanssakäyminen asiakkaiden kanssa. Kun yritys on sosiaalisessa mediassa ihmisläheinen, persoonallinen ja vastaa niin positiivisiin kuin negatiivisiin palautteisiin, tulee potentiaaliselle asiakkaalle mielikuva luotettavasta yrityksestä. [30, s. 19.] Helsinki Summer School on ollut aktiivisesti esillä Facebookissa, Twitterissä ja Instagramissa. Kaikilla näillä kanavilla on omat erityisominaisuutensa. Facebookissa sisältö on lähinnä Helsinki Summer School -sivun julkaisemia ajankohtaisia uutisia, kun taas Twitterissä ja Instagramissa Helsinki Summer Schoolin omien tilien lisäksi opiskelijat ja muut käyttäjät julkaisevat omaa sisältöään ja mielipiteitään käyttäen aihetunnistetta ”#helsinkisummerschool”. Muiden käyttäjien aihetunnistetta käyttävät julkaisut ovat käytännössä ilmaista sisältöä yritykselle, ja uutta verkkosivustoa kehitettäessä tämä sisältö haluttiin hyödyntää.

Yksityishenkilöiden sosiaalisessa mediassa jakama yrityksiin liittyvä sisältö voi tuntua potentiaalisesta asiakkaasta luotettavammalta ja puolueettomammalta kuin itse yrityksen julkaisut, koska yksityishenkilöllä ei yleensä ole syytä antaa valheellisesti positiivista kuvaa yrityksestä, vaan mielipiteet ovat todellisia ja rehellisiä. Helsinki Summer Schooliin yleisö ja potentiaaliset asiakkaat koostuvat suurimmaksi osaksi muualla kuin Suomessa

asuvista opiskelijoista, joten sosiaalisen median kanavat ovat helppo ja kätevä tapa saavuttaa yleisö myös toisella puolella maapalloa. Sosiaalinen media mahdollistaa ihmisläheisen, persoonallisen ja nopean kanssakäymisen mahdollisten asiakkaiden kanssa, ja moni yritys hyötyy sen käyttämisestä.

Koska Helsinki Summer School itse ja asiakkaat jakavat sisältöä sosiaalisessa mediassa melko säännöllisesti, se haluttiin hyödyntää ja tuoda sisältövirta myös verkkosivuille. Sosiaalisen median sisältöseinä sijoitettiin etusivulle, jossa se ei vie huomiota asiasisällöltä, mutta kiinnittää helposti käyttäjän huomion, koska sivulla ei ole suurta määrää muuta tekstiä tai kuvia. Sisältöseinään haetaan uutta sisältöä säännöllisesti puolen tunnin välein Facebookista, Twitteristä ja Instagramista. Kaikki projektissa käytetyt sosiaaliset mediat tarjoavat kehittäjälle rajapinnan, jonka avulla voidaan hakea julkaisuja esimerkiksi aihesanan tai käyttäjän perusteella. Rajapintojen avulla haetut julkaisut tallennetaan WordPressiin artikkeleiksi, jolloin esimerkiksi asiattomien julkaisujen piilottaminen etusivulta onnistuu vaivattomasti myös asiakkaan toimesta poistamalla artikkelit.

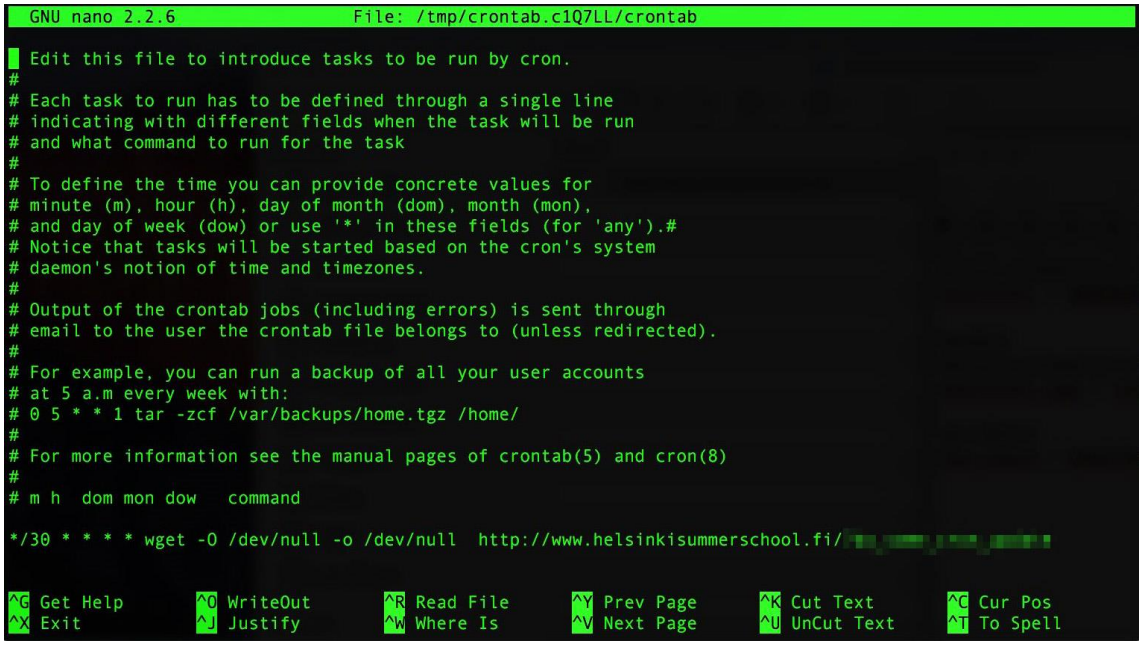
4.2 Julkaisujen hakeminen Cron-prosessin avulla

Cron on järjestelmän taustaprosessi, eli niin kutsuttu daemon, jota käytetään Unix-pohjaisissa käyttöjärjestelmissä. Cron mahdollistaa automatisoidun tehtävien ajon tiettynä ajankohtana. Cronin ajamia toimintoja kutsutaan cronjobeiksi eli Cron-tehtäviksi. Tehtävät kirjataan crontab-tiedostoon suorittaen komentorivillä ”crontab -e” komento, joka avaa crontabin muokattavaksi tai luo sen, jos tiedostoa ei ole valmiiksi olemassa. Jokaisella järjestelmän käyttäjällä on oma crontab-tiedosto, joka on tyypiltään yksinkertainen tekstitiedosto. Järjestelmä voi olla esimerkiksi Linux-pohjainen palvelin. Crontab-tiedostoon kirjataan suoritettavan tehtävän lisäksi tarkka ajankohta, jolloin tehtävä tulee suorittaa. Tehtävät suoritetaan, kun järjestelmän kellonaika ja Cron-tehtävän ajankohta täsmäävät. Käyttäjän ei tarvitse olla kirjautuneena järjestelmään, vaan tehtävät suoritetaan joka tapauksessa. Cron-tehtäviä käytetään muun muassa säännöllisten päivitysten tekemiseen ja varmuuskopioiden ottamiseen. [31.]

Ajankohdan määrityksessä *-merkki tarkoittaa sitä, että tehtävä suoritetaan jokaisessa kyseisen aikayksikön instanssissa, esimerkiksi joka tunti tai joka päivä. Aikayksiköt erotetaan välilyönnillä, ja tietyn aikayksikön sisällä voidaan käyttää pilkkua, yhdysmerkkiä tai vinoviivaa. Pilkulla aikayksikön instanssien erottaminen tarkoittaa sitä, että toiminto

suoritetaan molempien instanssien kohdalla, yhdysmerkillä instanssien erottaminen tarkoittaa, että toiminto suoritetaan jokaisen annettujen instanssien väliin jäävän instanssin kohdalla, ja vinoviivaa käytetään muodossa */10, joka tarkoittaa joka kymmenettä instanssia. [31.]

Kuva 8 on kuvakaappaus komentorivillä muokattavaksi avatusta crontab-tiedostosta, joka on käytössä Helsinki Summer Schoolin palvelimella. Kuvassa tiedosto on avattu GNU Nano -tekstieditorilla. Crontabin alussa on ohjeet, jotka auttavat tehtävän kirjoittamisessa. Ohjerivin alussa oleva #-merkki tekee rivistä kommentin, jolloin tiedostoa lukevat ohjelmat eivät yritä suorittaa rivin sisältöä. Kuvan 8 crontabiin on kirjattu yksi tehtävä, joka suoritetaan puolen tunnin välein. Cron-tehtävän suoritusajankohtaan voi määrittää aikayksiköt minuuteista kuukauteen, eli tehtävän voi suorittaa useimmillaan joka minuutti. Harvimmillaan tehtävän suoritus tapahtuu, kun ajankohdaksi on määritetty tietyn päivämäärän lisäksi viikonpäivä, jolloin pelkkä päivämäärän täsmääminen ei riitä, vaan myös viikonpäivän täytyy täsmätä. Kuvassa 8 tehtävän ajankohdan määritys ”*/30 * * * *” tarkoittaa, että tehtävä suoritetaan kolmenkymmenen minuutin välein jokaisena tuntina, päivänä, kuukautena ja viikonpäivänä. Ajankohdan jälkeen määritetään suoritettava toiminto. Tässä toiminnossa avataan määritetty URL, suoritetaan sen sisältämät tehtävät ja tallennetaan mahdollinen tuloste null-kansioon, joka on niin sanottu ”musta aukko”, eli todellisuudessa tuloste hävitetään. [31.]



Kuva 8. Helsinki Summer School -palvelimen crontab-tiedosto.

Cron-tehtävässä avattava URL suorittaa PHP-funktiot, jotka hakevat sisältöä Helsinki Summer Schoolin Facebook-sivulta, sekä Instagram- ja Twitter-julkaisuja aihetunnisteella ”#helsinkisummerschool”. Löytyneitä julkaisuja verrataan jo aiemmin WordPressiin tallennettuihin artikkeleihin, jotta samaa julkaisua ei tallenneta kahdesti. Myös poistettuja julkaisuja tarkkaillaan funktiolla, joka huomatessaan julkaisuartikkelin poiston WordPressistä tallentaa julkaisun tunnisteen WordPressin tietokantaan. Kun uutta sisältöä haetaan, tarkistetaan, ettei julkaisun tunniste täsmää tietokannasta löytyvään poistetun julkaisun tunnisteeseen. Jos poistettujen julkaisujen tunnisteita ei tallennettaisi tietokantaan, julkaisut tallentuisivat aina uusiksi artikkeleiksi, jotka pitäisi poistaa kerta toisensa jälkeen.

4.3 Facebook Graph API -rajapinta

Facebook Graph API (Application Programming Interface eli ohjelmointirajapinta) -rajapintaa käytetään datan välitykseen Facebookista ulos ja sisään. Rajapintaa voi käyttää esimerkiksi käyttäjän julkaisujen hakemiseen tai uuden julkaisun tekemiseen Facebookin ulkopuolelta. Suurin osa rajapinnan kautta lähetettävistä pyynnöistä vaatii ”access token”in eli eräänlaisen pääsytunnisteen, josta rajapinta voi tarkistaa, että pyynnön esittäjällä on riittävät oikeudet datan hakemiseen. Facebookin rajapinnassa on käytössä muutama erilainen pääsytunniste, joita käytetään riippuen siitä, minkälaista informaatiota rajapinnan kautta haetaan. Helsinki Summer Schoolin verkkosivuston etusivulle haetaan julkaisuja Helsinki Summer Schoolin Facebook-sivulta, joten rajapintapyyntöihin tarvitaan ”Page Access Token” eli pääsytunniste sivuille. Tämän tunnisteen käyttö mahdollistaa sivujen julkaisujen hakemisen ja sivun muokkaamisen esimerkiksi julkaisemalla kuvia sivun aikajanalle. [32.]

Facebook-rajapinnan käyttöön on tarjolla erilaisia apuvälineitä, muun muassa JavaScript ja PHP SDK:t (Software Development Kit eli ohjelmointipakkaus). Helsinki Summer Schoolin tapauksessa helpointa oli kuitenkin käyttää PHP-funktiota, joka tallentaa URL:n sisällön $json-muuttujaan (esimerkkikoodi 5). URL:iin määritetään halutun sivun ID, pääsytunniste ja valinnaiset kohdat ”fields” ja ”limit”. Fields eli kentät-parametri määrittää rajapinnan vastauksessa olevat kentät. Jos kentät-kohtaa ei määritetä, vastaukseen tulostuu julkaisun tekstisisältö, ID ja julkaisuajankohta. Esimerkkikoodin 5 kentät-parametrissa määritetään haettavaksi julkaisun ID, tekstisisältö, julkaisuaika, julkaisijan nimi sekä ID ja viimeisenä mahdollisen kuvan tai videon ID. Koodi on lyhennetty ja hieman

muokattu Helsinki Summer School -projektista, jossa haettavia kenttiä on yhteensä yksitoista, jotta julkaisuista saadaan kaikki tarvittava tieto. Limit- eli raja-parametrilla määritetään, kuinka monta julkaisua pyynnöllä haetaan. Rajan täytyy olla alle 100, muuten pyyntö palauttaa virheen. Jos raja-parametria ei ole asetettu, pyyntö hakee automaattisesti 25 julkaisua. [32.]

```
HTTP Get -pyynnön lähetys:
$json_url = "https://graph.facebook.com/
125758767438401/feed?access_token=[pääsytunniste]
&fields=id,message,created_time,from,object_id&limit=1”;
$json = file_get_contents($json_url);
$json = json_decode( $json );

Rajapinnan vastaus pyyntöön:
{"data": [
    {
        "id": "125758767438401_1267873703226896",
        "message": "It's almost spring … Helsinki Summer School
team",
        "created_time": "2016-03-04T09:25:39+0000",
        "from": {
              "name": "Helsinki Summer School",
              "id": "125758767438401"
         },
    }
]}
```

Esimerkkikoodi 5. Facebook-rajapintaan lähetettävä pyyntö ja ote rajapinnan vastauksesta.

Facebook-rajapinta vastaa pyyntöön JSON-muotoisella datalla. JSON-dataa voidaan käyttää PHP:ssä esimerkiksi muuttamalla se objektiksi json_decode-funktiolla, kuten esimerkkikoodissa 5. Muunnon jälkeen dataa on helppo käsitellä, ja esimerkiksi WordPress-artikkeleita luodessa voidaan jokainen data-objektin sisällä oleva taulukko käydä läpi silmukassa ja tallentaa halutut tiedot niille tarkoitettuihin WordPress-artikkelin kenttiin.

4.4 Instagram API -rajapinta

Instagram on Facebookin omistama, ja sen rajapinta toimii hyvin pitkälti samoin kuin Facebookin rajapinta. Instagramin rajapinnassa tunnistautumiseen käytetään kahta eri menetelmää, pääsytunnistetta ja asiakkaan ID:tä. Riippuen pyynnöstä ja haettavan informaation laadusta, saattaa rajapinta vaatia molempia tunnistautumistapoja tai pelkästään jompaakumpaa. [34.] Instagram-julkaisuja voi hakea rajapinnan kautta samalla tavalla kuin Facebookista esimerkkikoodissa 5, vain pyynnön URL muuttuu. Molemmista rajapinnoista saatavat vastaukset ovat JSON-muodossa, mutta niiden sisältö eroaa, koska informaatio julkaisuista on erilaista eri kanavissa.

Helsinki Summer Schoolin sivuston etusivulle haetaan Instagram-julkaisuja aihetunnisteen mukaan, ei pelkästään Helsinki Summer Schoolin omalta tililtä. Aihetunnisteen perusteella haettaessa vaaditaan vain pääsytunnus, eli HTTP Get-pyynnön URL voi olla esimerkiksi muodossa https://api.instagram.com/v1/tags/helsinkisummerschool/media/recent?access_token=[pääsytunnus]. Tämän pyynnön vastaus olisi JSON-muotoinen lista uusimmista helsinkisummerschool-aihetunnisteella merkityistä julkaisuista. [35.]

Instagram-julkaisusta dataa haettaessa vastaus sisältää kaiken mahdollisen informaation julkaisusta, eikä kenttiä voi rajata kuten Facebookin rajapintaa käytettäessä. Lisäämällä lukumäärä-parametri (”count”) URL:iin voidaan määrittää kuinka monta julkaisua yhdellä pyynnöllä haetaan. Lukumäärä voi kuitenkin olla enintään 20. Jos julkaisuja halutaan hakea yli 20, täytyy pyyntöjä tehdä useampia ja käyttää apuna maksimi-ID-parametria (”max_tag_id”), jolloin vastaukset voidaan niin sanotusti sivuttaa. Kun maksimi-ID-parametrin arvoksi määritetään edellisen pyynnön pienin ID, vastauksena saadaan 20 julkaisua, joiden ID on pienempi kuin maksimi-ID, eli niiden julkaisuajankohta on aiemmin. Tällöin yksi vastaus on yksi sivu julkaisuja. Pyynnössä voidaan käyttää myös minimi-ID-parametria, jolla määritetään, että haettavien julkaisujen tulee olla uudempia kuin parametrilla määritetty julkaisu. [35.]

Helsinki Summer Schoolin sivusto ohjelmoitiin kesällä 2015, ja Instagramin rajapintaan tuli muutoksia vuoden 2015 lopussa, joten kesällä ohjelmoitu Instagram-sisällön haku ei ole enää ajan tasalla. Rajapintoja uudistetaan jatkuvasti, joten niiden pohjalle kehityt sovellukset saattavat vanhentua yllättävän nopeasti. Yleensä vanhentunut koodi ei kuiten-

kaan lakkaa toimimasta heti uuden rajapintaversion ilmestyessä, vaan kehittäjille annetaan aikaa tehdä tarvittavat muutokset projekteihinsa. Helsinki Summer School -projektissa tunnistautuminen tehdään asiakas-ID:n kanssa, mutta uudessa rajapintaversiossa sen pitäisi tapahtua pääsytunnisteella. Tämän takia projektille täytyy hakea uusi pääsytunniste ja muuttaa koodi käyttämään sitä tunnistautumiseen. [34.]

4.5 Twitter API -rajapinta

Helsinki Summer School -projektissa Twitter-julkaisut haetaan Search API:a eli hakurajapintaa apuna käyttäen. Twitter vaatii monimutkaisemman tunnistautumisen kuin Facebook tai Instagram, joten apuna käytetään ulkopuolista komponenttia, Twitter API Exchange -luokkaa, joka helpottaa pyynnön lähetystä Twitterin rajapintaan. Tunnistautuminen vaatii neljä erilaista tunnistautumisavainta, joista muodostetaan usean välivaiheen myötä lopulliset yksilölliset tunnisteet. Välivaiheet sisältävät muun muassa kuluttaja-avaimen ja kuluttaja-salausavaimen koodaamisen kahden eri merkistön mukaan. [36.]

Kun rajapintaan lähetettävän pyynnön rakentamisen apuna käytetään TwitterAPIExhange-luokkaa, kehittäjän täytyy vain hakea Twitteristä tarvittavat tunnistautumisavaimet ja valmis luokka tekee loput. Esimerkkikoodin 6 alussa luodaan TwitterAPIExhange-olio, joka tallennetaan muuttujaan $twitter. Olion luonnin jälkeen pyyntöä varten asetetaan GET-kentät, rakennetaan OAuth-tunnistautuminen ja lopuksi lähetetään pyyntö. GET-kentät ovat pyynnön vaatimia parametreja, eli tässä tapauksessa q-parametrin arvoksi asetetaan hakusana ”helsinkisummerschool”. OAuth-tunnistautumisen rakentamiseen käytetään oliolle aiemmin määritettyjä tunnistaumisavaimia, URL:a, joka määrittyy haettavan datan mukaan, ja pyyntömetodia (GET tai POST). [37.]

```
$twitter = new TwitterAPIExchange(array(
    'oauth_access_token' => '[OAuth pääsytunniste]',
    'oauth_access_token_secret' => '[OAuth pääsytunnisteen sa-
lausavain]',
    'consumer_key' => '[kuluttaja-avain]',
    'consumer_secret' => '[kuluttaja-salausavain]'
));

$json = $twitter->setGetfield( '?q=' . urlencode('helsinkisum-
merschool') )
->buildOauth( 'https://api.twitter.com/1.1/search/tweets.json',
'GET' )
->performRequest();
```

Esimerkkikoodi 6. Twitterin hakurajapintaan pyynnön lähettäminen TwitterAPIExhange-luokkaa käyttäen.

Twitter Search API -rajapintaa käytetään hakemaan julkaisuja hakusanalla ja se toimii lähes samoin kuin Twitterin verkkosivujen hakutoiminto. Hakurajapinta hakee julkaisuja viimeisen viikon ajalta, ja haku pohjautuu ennemmin asiaankuuluvuuteen kuin täydellisyyteen. Hakusana voi olla esimerkiksi yksittäinen sana, #-merkillä alkava aihesana tai se voidaan muotoilla hakemaan vain tietyntyyppisiä julkaisuja, kuten vain mediaa eli kuvia tai videoita sisältäviä tuloksia lisäämällä hakusanan perään ”filter:media”. Hakusanat tulee aina merkistökoodata URL-kelpoisiksi. [38.]

GET-pyyntöön voidaan lisätä muutamia erilaisia parametreja hakusanan lisäksi; julkaisuja voi hakea esimerkiksi sijainnin mukaan. Geocode- eli geokoodi-parametrin arvoksi asetetaan leveys- ja pituuspiirin arvoista muodostuva geokoodi, joiden sijainnin perusteella haetaan julkaisuja. Hakurajapinta etsii aluksi julkaisuja, joiden datassa on pyydetty geokoodi, mutta koska sijainnin tallentumisen julkaisuihin voi estää, tällaisia julkaisuja ei välttämättä löydy. Jos geokoodiin täsmääviä julkaisuja ei löydy, etsitään käyttäjiä, joiden sijainti vastaa geokoodia. Muita mahdollisia parametreja ovat muun muassa ”lang” eli kieli, ”count” eli haettavien julkaisujen lukumäärä ja ”until” eli päivämäärä, johon asti julkaisuja haetaan. Jos lukumääräparametria ei ole määritetty pyyntöön, vastaus sisältää oletuksena korkeintaan 15 julkaisua. Lukumäärä voi olla korkeintaan 100. Twitter-rajapinta vastaa pyyntöön kuten Instagram-rajapinta eli palauttaen kaikki mahdolliset kentät JSON-muodossa. [38.]

5 Yhteenveto

Verkkokehityksen tavoitteena on kehittää tuote tai projekti, joka täyttää käyttäjän, asiakkaan ja kehittäjän tarpeet ja toiveet. Eri käyttötarkoituksilla on erilaiset vaatimukset, ja kehittäjän tehtävä on ottaa kaikki tarpeet ja toiveet huomioon kehitystyössä. Ei riitä, että täytetään asiakkaan ja käyttäjän tarpeet, vaan kehittäjän täytyy myös tehdä omasta työstään mahdollisimman suoraviivaista ja mieluisaa.

Käyttäjälle verkkosivustossa tärkeintä on käyttömukavuus riippumatta käytetystä laitteesta tai selaimesta. Sivuston täytyy olla yhtä hyvin käytettävissä pienellä älypuhelimella ja suurella tietokoneen näytöllä sekä kaikilla mahdollisilla laitteilla näiden välillä. Sivuston täytyy siis olla responsiivinen, ja sisällön täytyy olla joustavaa, mutta on tärkeää kiinnittää huomiota myös käyttömukavuuteen, kuten nappien ja tekstien kokoon. Kehitystyössä täytyy muistaa, että mobiililaitetta käytetään käsin, eli linkkejä tai nappeja painetaan sormin eikä tarkalla tietokonehiiren kohdistimella. Insinöörityössä responsiivisen ja mobiiliystävällisen sivuston kehittäminen onnistui toivotulla tavalla. Sivustoa on mahdotonta testata jokaisella tarjolla olevalla laitteella ja selaimella, mutta testausta tehtiin kattavasti ja sivusto toimii ongelmitta tuetuilla vanhemmillakin selaimilla. Sivusto mukautuu hyvin erikokoisille näytöille, ja käyttökokemus mobiililaitteilla on yhtä hyvä kuin tietokoneella.

Asiakkaalle tärkeää on sivuston moitteeton toimivuus eri laitteilla ja sivuston sisällön päivittämisen helppous. WordPress-sisällönhallintajärjestelmä mahdollistaa sen, että asiakas voi päivittää lähes kaikkea sisältöä. Asiakas on uuden sivuston julkaisemisen jälkeen onnistunut päivittämään ja lisäämään sisältöä itsenäisesti, ja kehittäjän apua on tarvittu lähinnä uusien ominaisuuksien kehittämisessä. Sivustopäivityksen tavoitteena oli myös yhtenäistää Helsingin yliopiston ja Helsinki Summer Schoolin verkkosivujen ilmeet, ja yhtenäistäminen onnistui helposti Helsingin yliopiston graafisen ohjeistuksen Bower-moduulin avulla.

Insinöörityön keskiössä oli verkkokehittäjän sujuva työnkulku. Kehittäjälle tärkeintä uuden verkkosivuston kehittämisessä on toimivan lopputuloksen lisäksi se, että kehitystyö on mieluisaa ja jatkokehitys on helposti siirrettävissä toiselle kehittäjälle. Verkkokehitykseen on tarjolla lukemattomia apuvälineitä, joista on mahdollista koota tapauskohtaisesti kehitysprojektille sopivin yhdistelmä. Kun projektia aloitettaessa konfiguroidaan kaikki

tarpeelliset apuvälineet ja toiminnot, voi kehityksessä keskittyä olennaiseen. Yksi kehitystyökalujen suurimpia etuja on se, että kehittäjä, joka ei tunne projektia, voi vaivattomasti asentaa kaikki tarvittavat apuvälineet muutamalla komennolla ja jatkaa kehitystyötä.

Insinöörityötä kirjoittaessa huomasin pieniä puutteita kehitystyön alussa tehdyissä työkalujen konfiguroinneissa ja opin erilaisia tapoja, joilla työkaluista saisi vielä enemmän hyötyä. Esimerkiksi Grunt konfiguroitiin aluksi niin, että käytetyt JavaScript-kirjastot minimoidaan jokainen omaksi tiedostokseen, vaikka verkkosivuston optimoinnin kannalta olisi järkevintä ketjuttaa kaikki koodi yhteen tiedostoon, koska tällöin palvelimelle lähetettäisiin vain yksi HTTP-pyyntö. Myöskään versionhallinnasta ei otettu kaikkea hyötyä ennen uuden sivuston julkaisua, sillä kehitin sivua yksin eli kukaan muu ei tarvinnut tekemiäni muutoksia omaan lokaaliin versioonsa, joten muutosten siirtäminen versionhallintaan unohtui. Versionhallintaa olisi hyvä käyttää myös silloin, kun kehittäjiä on vain yksi, koska se helpottaa huomattavasti tilanteita, jolloin halutaan palata vanhempaan vaiheeseen esimerkiksi koodivirheen takia. Versionhallintaa on kuitenkin käytetty tarkoituksenmukaisesti sivuston julkaisun jälkeen tehtyjen muutosten tallentamiseen.

Työssä kuvailtu työkaluyhdistelmä on toimiva yhdistelmä työkaluja, ja se on helppo monistaa uutta projektia aloitettaessa. Yhdistelmä sisältää kaiken tarpeellisen uuden verkkosivuston kehitykseen, ja siihen on helppo lisätä uusia ominaisuuksia ja toimintoja. Käytetyt työkalut auttavat kehittämään responsiivisen ja optimoidun verkkosivuston, joka vastaa käyttäjän, asiakkaan ja kehittäjän toiveisiin. Verkkokehitys kehittyy ja uudistuu jatkuvasti, mutta hyvät työkalut päivittyvät sen mukana.

Lähteet

1 Wolchover, Natalie. 2012. What Was the First Website Ever? Verkkodokumentti. Live Science. <http://www.livescience.com/34152-first-website-world-wide-web.html>. 21.8.2012. Luettu 14.9.2015.

2 Total number of Websites. Verkkodokumentti. Internet Live Stats. <http://www.internetlivestats.com/total-number-of-websites/>. Luettu 14.9.2015.

3 The CSS saga. Verkkodokumentti. W3C. <http://www.w3.org/Style/LieBos2e/history/>. Luettu 14.9.2015.

4 JavaScript History. Verkkodokumentti. HowToCreate. <http://www.howtocreate.co.uk/jshistory.html>. Luettu 14.9.2015.

5 PHP - Taking the world by storm. Verkkodokumentti. phpBB. <http://www.phpbbhq.com/developmentofphp.php>. Luettu 14.9.2015.

6 Gustafson, Aaron. 2011. Adaptive web design. Tennessee: Easy Readers.

7 Soojian, Cameron. 2015. A Brief History of Responsive Web Design. Verkkodokumentti. Synecore. <http://engage.synecoretech.com/marketing-technology-for-growth/bid/204297/A-Brief-History-of-Responsive-Web-Design>. 4.2.2015. Luettu 12.10.2015.

8 Helsinki Summer School. 2015. Verkkodokumentti. Helsinki Summer School. Luettu 14.9.2015.

9 Chrome Releases. 2016. Verkkodokumentti. Chrome Releases. <http://googlechromereleases.blogspot.fi/>. Luettu 14.3.2016.

10 Makino, Takaki & Phan, Doantam. 2015. Rolling out the mobile-friendly update. Verkkodokumentti. Google Webmaster Central Blog. <https://webmasters.googleblog.com/2015/04/rolling-out-mobile-friendly-update.html>. 21.4.2015. Luettu 15.3.2016.

11 Ross, Michael. 2013. Website Development: Local vs. Remote. Verkkodokumentti. Michael Ross. <http://www.ross.ws/content/website-development-local-vs-remote>. Luettu 7.3.2016.

12 Scotch Box 2.5. Verkkodokumentti. Scotch Box. <https://box.scotch.io/>. Luettu 7.3.2016.

13 Membrey, Peter & Hows, David. 2013. Learn Raspberry Pi with Linux. California: Apress.

14 Mallinson, Chris. 2016. Verkkodokumentti. Chris Mallinson. <https://mallinson.ca/osx-web-development/>. 16.1.2016. Luettu 7.3.2016.

15 Chacon, Scott & Straub, Ben. 2014. Pro Git. California: Apress.

16 Gackenheimer, Cory. 2013. Understanding Node.js. California: Apress.

17 Rouse, Margaret. 2008. Verkkodokumentti. WhatIs. <http://whatis.techtarget.com/definition/input-output-I-O>. Luettu 8.3.2016.

18 semver. 2016. Verkkodokumentti. npm. <https://docs.npmjs.com/misc/semver>. 8.1.2016. Luettu 8.3.2016.

19 Ambler, Tim & Cloud, Nicholas. 2015. JavaScript Frameworks for Modern Web Dev. California: Apress.

20 Suh, Jonathan. 2014. Get Started with Grunt. Verkkodokumentti. Jonathan Suh. <https://jonsuh.com/blog/get-started-with-grunt/>. 31.3.2014. Luettu 19.10.2015.

21 Lim, Hongkiat. 2009. Ultimate Guide To Web Optimization (Tips & Best Practices). Verkkodokumentti. Hongkiat. <http://www.hongkiat.com/blog/ultimate-guide-to-web-optimization-tips-best-practices/>. 27.8.2009. Luettu 7.3.2016.

22 JSHint, A Static Code Analysis Tool for JavaScript. Verkkodokumentti. JSHint. <http://jshint.com/about/>. Luettu 7.3.2016.

23 grunt-contrib-copy v1.0.0. 2016. Verkkodokumentti. GitHub. <https://github.com/gruntjs/grunt-contrib-copy>. Luettu 7.3.2016.

24 Sass Documentation. Verkkodokumentti. Sass. < http://sass-lang.com/documentation/file.SASS_REFERENCE.html>. Luettu 30.11.2015.

25 Snook, Jonathan. 2011. Font sizing with rem. Verkkodokumentti. <http://snook.ca/archives/html_and_css/font-size-with-rem>. 13.12.2011. Luettu 9.3.2016.

26 Grid. Verkkodokumentti. Foundation Docs. < http://foundation.zurb.com/sites/docs/v/5.5.3/components/grid.html>. Luettu 17.3.2016.

27 The Grid. Verkkodokumentti. Foundation. <http://foundation.zurb.com/grid.html>. Luettu 17.3.2016.

28 Orbit Slider. Verkkodokumentti. Foundation Docs. <http://foundation.zurb.com/sites/docs/v/5.5.3/components/orbit.html>. Luettu 18.3.2016.

29 Equalizer. Verkkodokumentti. Foundation Docs. <http://foundation.zurb.com/sites/docs/v/5.5.3/components/equalizer.html>. Luettu 18.3.106.

30 Funk, Tom. 2013. Advanced Social Media Marketing. California: Apress.

31 CronHowTo. 2015. Verkkodokumentti. Official Ubuntu Documentation. <https://help.ubuntu.com/community/CronHowto>. 4.9.2015. Luettu 9.3.2016.

32 Using the Graph Api. Verkkodokumentti. Facebook for developers. <https://developers.facebook.com/docs/graph-api/using-graph-api>. Luettu 10.3.2016.

33 Access Tokens. Verkkodokumentti. Facebook for developers. <https://developers.facebook.com/docs/facebook-login/access-tokens>. Luettu 10.3.2016.

34 Authentication. 2015. Verkkodokumentti. Instagram. <https://www.instagram.com/developer/authentication/>. Luettu 11.3.2016.

35 Tag Endpoints. 2015. Verkkodokumentti. Instagram. <https://www.instagram.com/developer/endpoints/tags/>. Luettu 11.3.2016.

36 Application-only authentication. 2015. Verkkodokumentti. Twitter. <https://dev.twitter.com/oauth/application-only>. Luettu 12.3.2016.

37 twitter-api-php. 2015. Verkkodokumentti. GitHub. <https://github.com/J7mbo/twitter-api-php>. Luettu 13.3.2016.

38 The Search API. 2015. Verkkodokumentti. Twitter. <https://dev.twitter.com/rest/public/search>. Luettu 13.3.2016.