

# Tietokoneen käytön analysointi- työkalun suunnittelu ja toteutus

Atte Tarvainen

Opinnäytetyö

Joulukuu 2016

Tekniikan ja liikenteen ala

Insinööri (AMK), Ohjelmistotekniikan koulutusohjelma

Tekijä(t) Tarvainen, Atte	Julkaisun laji Opinnäytetyö, AMK	Päivämäärä Joulukuu 2016
	Sivumäärä 93	Julkaisun kieli Suomi
		Verkojulkaisulupa myönnetty: x
Työn nimi <b>Tietokoneen käytön analysointityökalun suunnittelu ja toteutus</b>		
Tutkinto-ohjelma Ohjelmistotekniikan koulutusohjelma		
Työn ohjaaja(t) Ari Rantala		
Toimeksiantaja(t)		
<p>Tiivistelmä</p> <p>Opinnäytetyön tavoitteena oli toteuttaa järjestelmä, jonka avulla käyttäjä kykenee vertaamaan järjestelmään tuomansa dataa rinnakkain ja löytämään tätä kautta mahdollisia korrelaatioita täysin toisistaan irrallisten tunnuslukujen väliltä. Järjestelmään oli tavoitteena toteuttaa erikseen sovellus datan analysointiin sekä käyttäjän päätelaitteen käyttöä analysoiva työkalu, jonka keräämä data tallennettaisiin varsinaiseen dataa analysoivaan sovellukseen.</p> <p>Työssä toteutettiin web-ohjelmointiteknologioita käyttäen web-sovellus, sekä käyttäjän päätelaitteelle sijoitettava tietokoneen käyttöä analysoiva sovellus. Web-sovellus ohjelmoitiin kahdessa eri osassa: web-selainsovelluksena sekä palvelinsovelluksena. Palvelinsovellus toteutettiin käyttäen LAMP-sovelluskokoelmaa hyödyntäen useita valmiita sovelluskomponentteja. Palvelinsovellus ohjelmoitiin avaamaan REST-rajapinta, jonka kautta web-selainsovellus kykenee viestimään. Web-selainsovellus ohjelmoitiin käyttämällä AngularJS-sovelluskehystä ja käyttöliittymäkomponenttina käytettiin Angular Material -käyttöliittymäkirjastoa. Työpöytäsovelluksen ohjelmoinnin alustana käytettiin useita eri käyttöjärjestelmiä tukevaa NWJS-sovelluskehystä, jonka ohjelmoinnissa hyödynnettiin web-selainsovellukseen ohjelmoituja komponentteja.</p> <p>Sovellukseen toteutettiin kaikki sovelluksen vaatimuksiin määritellyt toiminnot. Sovellus otettiin käyttöön opinnäytetyön laatijan kotiympäristöön, jossa se asetettiin tallentamaan lämpötilaa sekä muita elämään liittyviä tunnuslukuja.</p>		
Avainsanat ( <a href="#">asiasanat</a> ) Angular, JavaScript, PHP, Symfony, REST, Aikadimensio		
Muut tiedot		

Author(s) Tarvainen, Atte	Type of publication Bachelor's thesis	Date December 2016
		Language of publication: Finnish
	Number of pages 93	Permission for web publication: x
Title of publication <b>Design and development of a software for computer usage analyzation</b>		
Degree programme Software Engineering		
Supervisor(s) Ari Rantala		
Assigned by		
<p>Abstract</p> <p>The main goal of the thesis was to implement a web service where users may save and analyze their data. Users should also be able to compare separated data series with each other and find hidden correlations between them. The system aimed at two different applications. The first was the web application and the second the one to be installed to the user's computer. The computer application was intended to log user's computer activity statistics and send them over to the web application.</p> <p>Two applications were designed and programmed using the web application development techniques. The LAMP stack with several frameworks and libraries was used as a base for the web application development. The web application was built to consist of two separated parts, the server application and the client application. The communication interface between the client application and the server application was implemented as REST. The desktop application was programmed on NWJS application framework and some parts of the Angular application built for the web application were used in the desktop application development as well.</p> <p>The thesis results in two totally separated applications which both implemented all the requirements listed in the original system specification. The whole system was taken into use in the writer's home environment as a service for saving the home temperature data side by side with all other data on indicators gathered from ordinary living.</p>		
Keywords/tags ( <a href="#">subjects</a> ) Angular, JavaScript, PHP, Symfony, REST, DateDimension		
Miscellaneous		

## Sisältö

<b>Sanasto .....</b>	<b>7</b>
<b>1 Työn lähtökohdat .....</b>	<b>10</b>
1.1 Taustat.....	10
1.2 Toimeksiantaja ja yhteistyökumppanit.....	10
1.3 Opinnäytetyön tavoitteet .....	11
1.3.1 Konkreettinen tavoite .....	11
1.3.2 Muut tavoitteet .....	11
<b>2 Saatavilla olevat toteutukset .....</b>	<b>12</b>
2.1 Yleistä .....	12
2.2 Tietokoneen käytön mittaaminen .....	12
2.3 Datan analysoinnin työkalut.....	14
2.4 Johtopäätökset .....	14
<b>3 Vaatimusmäärittely .....</b>	<b>15</b>
3.1 Yleistä .....	15
3.2 Vaatimukset koko ohjelmiston kontekstissa .....	15
3.3 Web-sovellus .....	15
3.3.1 Arkkitehtuurilliset vaatimukset.....	15
3.3.2 Toiminnalliset vaatimukset.....	16
3.4 Työpöytäsovellus .....	16
<b>4 Suunnittelu ja teknologiavalinnat .....</b>	<b>17</b>
4.1 Yleistä .....	17
4.2 Arkkitehtuurilliset päälinjat .....	18
4.2.1 Yleistä.....	18
4.2.2 Oma palvelin vai pilvitietokanta palveluna? .....	18
4.2.3 Järjestelmän kokonaisrakenne ja toimintaperiaate.....	19
4.3 Datan varastointi .....	20

	2
4.3.1 Yleistä.....	20
4.3.2 Tietokantaratkaisut.....	22
4.4 Web-palvelinohjelmoinnin teknologiat .....	23
4.4.1 Yleistä.....	23
4.4.2 Palvelinkäyttöjärjestelmät .....	23
4.4.3 HTTP-palvelinohjelmat.....	24
4.4.4 Palvelinohjelmointikielet .....	24
4.4.5 Sovelluskehukset.....	26
4.4.6 Yhteenveto.....	27
4.5 Web-selainsovellus.....	28
4.5.1 Yleistä.....	28
4.5.2 Selainsovelluskehysä.....	28
4.5.3 Käyttöliittymän ehostaminen .....	30
4.6 Työpöytäsovellus.....	30
4.6.1 Yleistä.....	30
4.6.2 Teknologiavaihtoehdot.....	31
4.6.3 Yhteenveto.....	32
4.7 Rajapintaratkaisut .....	32
4.8 Teknologiavalinnat yhteenvetona .....	33
<b>5 Toteutus .....</b>	<b>34</b>
5.1 Yleistä .....	34
5.2 Suunnittelu .....	35
5.2.1 Yleistä.....	35
5.2.2 Dynaaminen tiedon tallentaminen .....	35
5.2.3 Rajapintasuunnittelu.....	36
5.3 Kehitystyökalut ja käytetyt palvelut .....	37
5.3.1 Sovelluskehitysympäristö .....	37

5.3.2	Versionhallinta .....	38
5.3.3	Testausympäristö.....	38
5.4	Palvelinsovellus .....	39
5.4.1	Symfony-sovelluskehiksen käyttöönotto .....	39
5.4.2	Symfony ja tietokanta .....	42
5.4.3	Tietokannan versionhallinta .....	44
5.4.4	REST-rajapinnan toteutus .....	45
5.4.5	Käyttäjän autentikointi ja istunnonhallinta .....	46
5.4.6	Annotaatioiden käyttö PHP-ohjelmoinnissa .....	48
5.4.7	Sisäänluetun datan aukilasku .....	49
5.4.8	Datan haku.....	51
5.4.9	Tietokannan kokonaisrakenne.....	53
5.5	Selainsovellus .....	54
5.5.1	SPA – yhden sivun sovellus .....	54
5.5.2	AngularJS:n käyttöönotto ja konfigurointi .....	56
5.5.3	Arkkitehtuuri AngularJS-sovelluksessa .....	57
5.5.4	Käyttöliittymän toteuttaminen Angular Material -kirjastolla .....	58
5.5.5	Käyttäjän istunnon ja käyttöoikeuksien hallinta .....	60
5.5.6	Kieliversiot .....	61
5.5.7	Datan haku ja haetun datan visualisointi .....	62
5.6	Työpöytäsovellus .....	66
5.6.1	NWJS-sovelluskehiksen käyttöönotto.....	66
5.6.2	Globaalit syöttölaitteiden tapahtumankuuntelijat .....	67
5.6.3	Aktiivisen sovelluksen selvittäminen alustariippumattomasti.....	68
5.6.4	Datan kerääminen ja lähettäminen .....	69
5.6.5	Työpöytäsovelluksen käyttöliittymä .....	70
5.7	Järjestelmän käyttöönotto Ubuntu-käyttöjärjestelmälle .....	72

5.7.1	Yleistä.....	72
5.7.2	Tarvittavat sovellukset ja mahdolliset lisäosat .....	72
5.7.3	Web-palvelinsovelluksen asentaminen .....	74
5.7.4	Web-selainsovelluksen asentaminen .....	76
5.7.5	Ensimmäinen käyttökerta .....	77
5.8	Lisämoduulit ja ohi vaatimusmäärittelyn toteutetut ominaisuudet .....	77
5.8.1	Lämpötilaa mittaava moduuli .....	77
5.8.2	Tietokannan varmuuskopiointi .....	78
5.8.3	Ilmoitusjärjestelmä .....	79
<b>6</b>	<b>Lopputulokset ja johtopäätökset .....</b>	<b>80</b>
6.1	Lopullinen tuote .....	80
6.2	Testauttaminen .....	81
6.3	Havaitut viat ja puutteet .....	82
6.4	Kehitysideat ja –haaveet .....	83
6.4.1	Sovelluksen käyttö ryhmien tietojen seuraamiseen .....	83
6.4.2	Ylenpalttinen modulaarisuus .....	84
6.4.3	Rajapintojen käsittelyn edistäminen .....	84
6.4.4	Muut lisätoiminnot .....	85
6.5	Johtopäätökset .....	85
	<b>Lähteet.....</b>	<b>87</b>
	<b>Liitteet .....</b>	<b>90</b>
	Liite 1. Kuvakaappauksia web-selainsovelluksen liittymistä.....	90

## Kuviot

Kuvio 1. IOGraphican tuottamaa abstraktia taidetta .....	13
Kuvio 2. Sovelluksen kokonaisrakenne ja yhteydet.....	20
Kuvio 3. Esimerkki relaatiotietokannan tauluista.....	21
Kuvio 4. Esimerkkidata esitettynä epärelaatiotyyppisenä .....	21
Kuvio 5. Sovelluksen kokonaisrakenne avattuna yksittäisten teknologioiden tasolle	34
Kuvio 6. Symfony-projektin luonti.....	40
Kuvio 7. Sovelluksen parametritiedoston, parameters.yml, oletussisältö .....	40
Kuvio 8. Composer-paketinhallintajärjestelmän asennustoiminnon tuloste .....	41
Kuvio 9. Interaktiivinen entiteetin generoiva näkymä .....	43
Kuvio 10. Symfonyn komennon automaattisesti generoima entiteettiluokka.....	43
Kuvio 11. Tietokannan skeeman päivityskomennon ajaminen .....	44
Kuvio 12. Tietokannan user-taulun rakenne komennon suorittamisen jälkeen .....	44
Kuvio 13. Muuttuneesta entiteettiluokasta generoitu migraatiotiedosto .....	45
Kuvio 14. Tietokantamigraatioiden ajaminen tietokantaan.....	45
Kuvio 15. Omien tietojen hakupolku määriteltynä annotaatioilla .....	46
Kuvio 16. @Permission-annotaation käyttö metodin käyttöoikeustarkastelussa .....	49
Kuvio 17. Tietokannan eventin määrittely SQL-kielellä.....	50
Kuvio 18. Dataa auki laskevan proseduurin alkusanat .....	51
Kuvio 19. Aikadimensiotaulun rakenne ja viisi ensimmäistä riviä .....	52
Kuvio 20. Esimerkki sovelluksen tekemästä datanhaun toteuttavasta SQL-kyselystä	53
Kuvio 21. Tietokannan kokonaisrakenne .....	54
Kuvio 22. Moduulien injektointi AngularJS-sovellukseen.....	56
Kuvio 23. NgRouten polkumäärittelyjä .....	57
Kuvio 24. AngularJS-sovelluksen arkkitehtuuri .....	58
Kuvio 25. Angular Materialin avulla web-selainsovellukseen toteutettu lomakenäkymä .....	59
Kuvio 26. Me-kyselyn suorittava määrittely AngularJS-sovelluksessa .....	61
Kuvio 27. Angular Translate -moduulin käyttö käännösten hallintaan .....	62
Kuvio 28. Päällekkäin verrattuja datasarjoja esitettynä viivakaaviossa .....	63
Kuvio 29. Hiiren liikettä tietyllä aikavälillä kuvaava raportti .....	64
Kuvio 30. Näppäimistön kuormituskuvaaja web-selainsovelluksessa.....	65

Kuvio 31. Näppäimistökomponentin toteuttava ohjelmakoodi.....	65
Kuvio 32. Angular Materialin latausindikaattorista räätälöityjä komponentteja .....	66
Kuvio 33. Globaali hiiren kytkimen painalluksen nappaava kuuntelija .....	68
Kuvio 34. Työpöytäsovelluksen käyttöliittymään toteutetut reaaliaikakomponentit.	71
Kuvio 35. Web-rajapinnan asetusnäkymä työpöytäsovelluksessa .....	71
Kuvio 36. mysql_secure_installation-asennusohjelma .....	73
Kuvio 37. Uuden käyttäjän luonti tietokantaan .....	73
Kuvio 38. Apache-palvelinohjelman oletussivu.....	74
Kuvio 39. Symfonyn interaktiivinen parametrienkyselytoiminto .....	75
Kuvio 40. Apache-palvelinohjelman uudelleenohjauksen konfigurointi.....	76
Kuvio 41. Parametrien määrittelytiedoston sisältö.....	77
Kuvio 42. Web-selainsovelluksen lähdekoodin määrä .....	81
Kuvio 43. Web-palvelinsovelluksen lähdekoodin määrä.....	81
Kuvio 44. Työpöytäsovelluksen lähdekoodin määrä .....	81

## **Sanasto**

### **AngularJS**

Googlen kehittämä JavaScript-pohjainen (ks. JavaScript) web-selainsovelluskehys, joka toteuttaa MVC-arkkitehtuuria (ks. MVC).

### **Annotaatio**

Tiettyyn sovelluksen komponenttiin, esimerkiksi funktioon, liitettävää metatietoa.

### **Apache**

Web-palvelinohjelma, jonka tehtävä on mahdollistaa staattisten web-resurssien (ks. web-resurssi) lähettäminen asiakassovellukselle pyydettäessä.

### **API**

Application Programming Interface. Ohjelmointirajapinta, joka muodostuu joukosta toteutustapoja, protokollia ja työkaluja määrittäen sovelluskehityksen tai sovellusten välisen viestinnän keinot.

### **Cronjob**

Unix-käyttöjärjestelmäympäristössä ajastettujen toimintojen suorittamisen käyttöjärjestelmätasolla mahdollistava työkalu.

### **Entiteetti**

Tosielämän esinettä tai asiaa kuvaava mallinnos ohjelmakoodissa.

### **Git**

Hajautettu versionhallintajärjestelmä (ks. VCS) pienille ja suurille epälineaarisille ohjelmistoprojekteille.

### **HTML**

Hypertext Markup Language. Webin merkintäkieli, jota käyttämällä ilmaistaan web-selaimelle (ks. web-selain), mitä web-sivulla tulee näyttää.

### **HTTP**

Hypertext Transfer Protocol. Web-selainten ja web-palvelimien käyttämä tiedonsiirtoprotokolla.

### **InnoDB**

MySQL-tietokannanhallintajärjestelmän oletustietokantamoottori. Taso, joka lopulta suorittaa kaikki tietokannan lisäys-, muokkaus-, luku- ja poisto-operaatiot.

### **Java**

Yleiskäyttöinen, monialustainen, tiukasti tyyпитetty sekä käännettävä ja tulkattava ohjelmointikieli, jota suoritetaan Java-virtuaalikoneella.

**JavaScript**

Tyypittämätön tulkettava webin ohjelmointikieli, jota web-selaimet käyttävät mahdollistaakseen dynaamisten web-sivujen toteuttamisen.

**MariaDB**

Tietokannanhallintajärjestelmä, joka toimii välittäjänä käyttäjän ja tietokantamoottorin välissä.

**MVC**

Ohjelmistosuunnittelumalli, joka perustuu sovelluslogiikan jakamisen kolmeen osaan: malliin, näkymään ja ohjaimeen.

**Node.js**

JavaScript-kielellä ohjelmoitava monialustainen palvelinohjelmointiympäristö.

**NWJS**

Chromium-web-selaimeen pohjautuva monialustainen työpöytäsovelluskehys, jonka ohjelmoinnissa käytetään perinteisiä web-selainohjelmointiteknologioita (ks. web-selain) sekä Node.js-palvelinohjelmointiteknologiaa (ks. Node.js).

**ORM**

Object-Relational Mapping. Ohjelmointiteknologia, jossa dataa muutetaan tietokannasta kohdeympäristöön sopivaan objektimuotoon ja vastaavasti objekteista takaisin tietokantaan.

**PHP**

Personal Home Page (PHP: Hypertext Preprocessor). Perinteikäs web-palvelinohjelmointikieli (ks. web-palvelin), joka on ensisijaisesti suunnattu web-sovelluskehitykseen.

**REST**

Representational State Transfer. Tilaton tiedonsiirtoprotokolla, joka lähes kaikissa tapauksissa käyttää hyödykseen HTTP-protokollaa (ks. HTTP).

**Sovelluskehys**

Kokoelma valmiita sovelluskomponentteja, jotka abstrahoivat sovelluskehityksen osioita yleisemmälle tasolle.

**SQL**

Structured Query Language. Kyselykieli, jota relaatiotietokannat käyttävät tietokantaoperaatioiden toteuttamiseen.

**Transaktio**

Transaktio koostuu tietystä joukosta tiedonsiirtoa, jota käsitellään yhtenä erillisenä toimenpiteenä, jolloin voidaan datan palauttatismahdollisuuden ansiosta varmistua tietokantaoperaatioiden tekemien datamuutosten eheydestä.

**VCS**

Version Control System. Versionhallintajärjestelmä mahdollistaa siirtymisen sovelluskehityksen eri vaiheiden välillä pitäen kirjaa kaikista sovellukseen tehtävistä muutoksista.

**Web-palvelin**

Tietokonejärjestelmä, joka mahdollistaa tiedon jakamisen internetiin käyttäen yleisiä tiedonsiirtoprotokollia.

**Web-resurssi**

Mikä tahansa webistä yksilöitävä ladattavissa oleva asia. Yleistetään yleensä koskemaan esimerkiksi tiedostoja, kuvia tai muuta multimediaa, joka voidaan esittää web-sivulla.

**Web-selain**

Web-sivun sisällön lataava ja esittävä asiakkaan käyttämä sovellus. Yleisesti käytettyjä web-selaimia ovat esimerkiksi Google Chrome ja Mozilla Firefox.

# 1 Työn lähtökohdat

## 1.1 Taustat

Kuluttaessaan päivät tietokonetyötä tekevänä tulee monesti pysähtyttyä miettimään, mitä oikeasti päiväsi aikana teet? Miten tapasi elää muuta elämääsi vaikuttaa siihen, miten toimit työssäsi ja käytät sinulle osoitettua ainoaa konkreettista, käsin kosketeltavaa työkaluasi, tietokonetta? Voidaanko tiettyjen tunnuslukujen välillä tietokoneen ja elävän elämän välissä löytää jokin yhteys? Onko esimerkiksi yönunen pituuden ja kirjoitusnopeuden välillä havaittavissa yhteys? Entä miten tähän yhteyteen vaikuttaa juomasi kofeiinipitoisen juoman, esimerkiksi kahvin määrä?

Kun tietokonetyöläisen urasi jatkuu ja jatkuu, nousee varmasti mieleesi myös kysymys siitä, miten olet kehittynyt? Kirjoitatko yhä yhtä hitaasti, teetkö yhä samat virheet kuin aina ennenkin ja käytätkö aina vaan samoja sovelluksia päivästä toiseen samaan kellonaikaan? Entä millaista oma tekemisesi on verrattuna muiden, käytännössä täsmälleen samaa työtä tekevien suhteen?

Onko edellä mainituilla asioilla mitään yhteyttä keskenään? Voiko kahdella täysin toisistaan irrallisella asialla olla keskenään jokin yhteys? Onko millään mitään väliä? Kysymykset ovat vaikeita eikä niihin ehkä koskaan saada vastausta. Opinnäytetyössä toututetaan yksi tapa etsiä vastauksia näihin kysymyksiin.

## 1.2 Toimeksiantaja ja yhteistyökumppanit

Työllä ei ollut varsinaista toimeksiantajaa idean ja ajatuksen kummutessa opinnäytetyön tekijältä itseltään. Työn suorituksen ja kehityksen aikaan yhteistyökumppanina toimi viitasaarelainen yritys Tmi Tarvaisen Vaappu ja Vehje, joka tarvittaessa kykeni avustamaan työskentelyssä tarjoamalla käyttöön työtilojaan ja -välineitään. Opinnäytetyön ollessa tuotantoon siirrettävässä vaiheessa suoritettiin myös järjestelmän pienimuotoinen paikallinen käyttöönotto yrityksen tiloihin, joissa sovelluksella oli määrä aluksi mitata työtilojen ilmastollisia olosuhteita ja myöhemmässä vaiheessa syvemmin itse työtilojen ja työvälineiden käyttöastetta ja työtiloissa tapahtuvan työskentelyn määrää.

### 1.3 Opinnäytetyön tavoitteet

#### 1.3.1 Konkreettinen tavoite

Opinnäytetyössä konkreettisena tavoitteena oli suunnitella ja toteuttaa järjestelmä, jolla käyttäjä voi seurata omaa tietokoneen käyttöä vähintäänkin näppäimistön, hiiren ja käytettyjen sovellusten osalta. Käyttäjän tulee myös kyetä tuomaan järjestelmään muuta dataa täysin järjestelmän ulkopuolelta ja vertaamaan tätä tietoa keskenään tietokoneen käytöstä saatuun dataan. Käyttäjän tulee lisäksi pystyä muokkaamaan ja tarvittaessa poistamaan sovellukseen tuomaansa dataa.

Opinnäytetyössä toteutettiin alustaa suuremmalle kokonaisuudelle, jonka kehittäminen jatkuu opinnäytetyöprojektin päättymisen jälkeen. Jatkokehitykseen suunnitellut ominaisuudet asettivat toisaalta osaltaan vaatimuksia jo opinnäytetyön aikaan tehtäviin ohjelmoinnillisiin ja arkkitehtuurillisiin ratkaisuihin.

Jotta alkuperäinen opinnäytetyön ajatus on todennettavissa myös käytännössä, opinnäytetyön lopputulos oli tavoitteena ottaa käyttöön opinnäytetyön tekijän kotiympäristössä siten, että sovellukseen kirjataan tietokoneen mittausdatan ohella myös muuta tekijän elämään liittyvää dataa. Tällöin tätä muuta dataa voidaan peilata tietokoneelta kerättyyn dataan. Datan keräykseen liittyen toteutettiin myös lämpötilaa mittaava ja sovellukseen lähettävä moduuli.

Tarkempi sovelluksen vaatimusmäärittely esitellään luvun 3 alussa.

#### 1.3.2 Muut tavoitteet

Henkilökohtaisena tavoitteena opinnäytetyössä oli kyetä hallitsemaan vähäinen käytettävissä oleva aika siten, että työkuorma jakautuu tasaisesti. Opinnäytetyön toteutusvaiheessa pyrittiin hyödyntämään olemassa olevaa toteutusta ja tutustumaan uusiin ohjelmointikomponentteihin ja -välineisiin sen sijaan, että ohjelmoidaan kaikki alusta saakka itse. Tekemisen ideana olikin tutkia, oppia ja tehdä virheitä, joiden kautta uutta oppia voi ammentaa.

Opinnäytetyö vietiin myös läpi ohjelmistotuotannollisin menetelmin käyttäen valittua ja sopivaksi sovellettua menetelmää. Sovelluksen toteutusvaiheessa otettiin huomioon myös ohjelmistotestaukselliset erityishuomiot, jotka kuitenkin varsinaisessa opinnäytetyökontekstissa jätettiin pienemmälle huomiolle.

## **2 Saatavilla olevat toteutukset**

### **2.1 Yleistä**

Mittaamiseen ja mittausdatan analysointiin toteutettuja sovelluksia on saatavilla runsaasti. Markkinoilta löytyvät sovellukset ovat yleensä kuitenkin hyvin fokusoituneita mittaamaan juuri sitä, mitä mittaavat, ja niistä saatava data on käytännössä aina sidottu hyvin tiukasti sovellukseen itseensä: sovelluksen mittaamaa dataa voidaan tarkastella vain sovelluksen itsensä sisällä, ja huonoimmassa tapauksessa data on vielä täysin sessiokohtaista eikä edes tallennu mihinkään.

Sovelluksia, joihin mittaavista sovelluksista mahdollisesti ulos saatua dataa voidaan viedä ja analysoida yhdessä jonkin toisen datan kanssa, on tarjolla jonkin verran ja monen lähtöön. Nykyisin suuri osa sovelluksista on selainpohjaisia, jolloin ei yleensä tarvita kuin rekisteröityminen ja kirjautuminen palveluun, minkä jälkeen kaikki toiminnot ovat käytössä riippumatta käyttäjän maantieteellisestä sijainnista. Monet näistä sovelluksista tukevat todella mutkikkaidenkin raporttien ja kaavioiden toteuttamista käyttäjän datasta.

### **2.2 Tietokoneen käytön mittaaminen**

Tietokoneen käytön mittaamiseen on markkinoilla vaihtoehtoja valittavaksi asti. Useimmat näistä sovelluksista ovat alustariippuvaisia toimien pelkästään esimerkiksi Windows-käyttöjärjestelmään asennettuna. Sovellukset myös mittaavat hyvin toisistaan poikkeavaa dataa, eikä yksittäistä sovellusta, joka kattaisi kokonaisvaltaisesti kaikkea tietokoneen ohjauslaitteista käytettyihin sovelluksiin ja käyttöaikoihin, löydä laisinkaan.

## Visual TimeAnalyzer

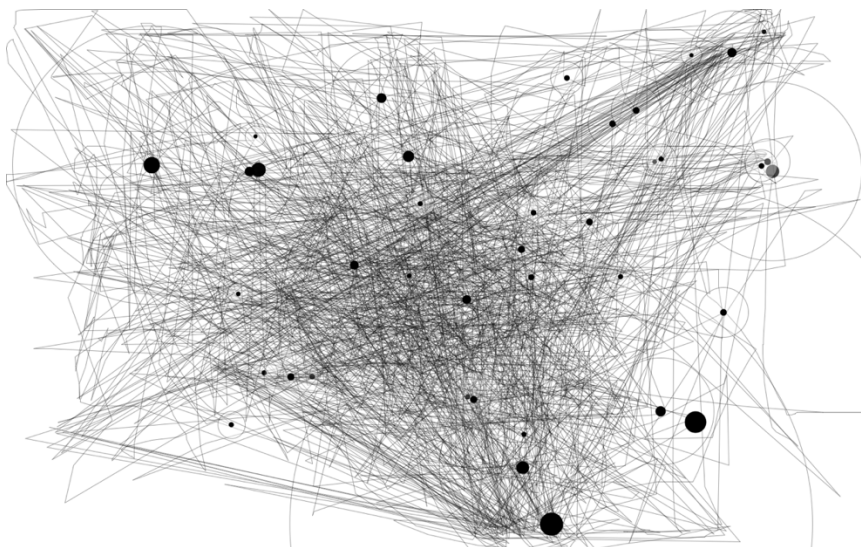
Visual TimeAnalyzer on Neuber Softwaren kehittämä sovellus tietokoneen käytön seuraamiseen esimerkiksi perheenjäsenten kesken. Sovellus seuraa henkilön käyttämiä sovelluksia, prosesseja ja internetin käyttöä muodostaen keräämistään tiedoista informatiivisia raportteja. Sovellus on asennettavissa vain Windows-käyttöjärjestelmään. (Time Tracking Software 2016.)

## Mousotron

Mousotron on Blacksun Softwaren toteuttama sovellus tietokoneeseen kytketyn hiiren analysointiin. Sovellus tallentaa tietoa kursorin kulkemasta matkasta, hiiren näppäimen painalluksista, tuplaklikkauksista, hiiren rullan pyöryksestä, kursorin nopeudesta sekä kursorin koordinaateista tietokoneen näyttöpäätteellä. Mousotron tarjoaa myös tavan viedä mittaustieto ulos sovelluksesta HTTP-pyyntöjä käyttäen. Näin käyttäjä voi esimerkiksi lähettää mittaustuloksensa omalle kotisivulle esitettäväksi. Mousotron on saatavilla vain Windows-käyttöjärjestelmään. (Mousotron 2016.)

## IOGraphica

IOGraphica on sovellus, jolla voidaan nauhoittaa tietokoneeseen kytketyn hiiren liikkeitä ja painalluksia ja luoda tämän nauhoitetun tiedon pohjalta eräänlaista taidetta (ks. kuvio 1). Sovellus piirtää hiiren kulkeman polun viivoina ja hiiren klikkaukset palloina, jonka johdosta syntyy eräänlaista modernia abstraktia taidetta. Sovellus on saatavilla niin Windows-, OSX- kuin Linux-käyttöjärjestelmiin. (IOGraphica 2016.)



Kuvio 1. IOGraphican tuottamaa abstraktia taidetta

## 2.3 Datan analysoinnin työkalut

### **plot.ly**

Plot.ly on omien sanojensa mukaan moderni alusta ketterälle liiketoimintatiedon hallinnalle ja datan tutkimukselle. Sovelluksena plot.ly tarjoaa yleiskäyttöisen alustan erilaisen datan esittämiseen ja analysointiin antaen käyttäjälleen mahdollisuuden luoda esimerkiksi kuvaajia, taulukoita ynnä muita graafisia dataa ja trendejä visualisoivia elementtejä. Selainpohjainen sovellus on tiettyyn rajaan saakka myös täysin käyttäjälleen ilmainen, ja sovellusta on saatavilla myös itse ylläpidettävänä versiona. (Plot.ly 2016.)

### **Microsoft Excel**

Ehkä perinteisin vaihtoehto tiedon kaikkien käsittelyyn sen elinkaaren aikana on Microsoft Office -tuoteperheen Excel-tilukkolaskenta-ohjelmisto. Excelin avulla käyttäjä voi laatia laajan skaalan erilaisia dynaamisia kuvaajia ja yhteenvetoja omasta datasta. Excel tukee myös omien laajennosten kirjoittamista Visual Basic –ohjelmointikielellä, mikä mahdollistaa sovelluksen entistäkin tehokkaamman käytön. Microsoft Office -tuoteperheen tuotteet ovat nykyisin saatavilla myös selainohjelmoina, minkä ansiosta laskentataulukoita voidaan jakaa ja muokata riippumatta omasta fyysisestä sijainnista. Excel on saatavana myös työpöytäsovelluksena Windows- ja OSX-käyttöjärjestelmille. (Excel 2016.)

## 2.4 Johtopäätökset

Erilaisen tiedon keräämiseen on olemassa paljon valmiita ratkaisuita, joiden joukosta löytyy varmasti jokaiseen tarpeeseen sopiva ratkaisu. Kuitenkaan yhtään sovellusta, jota voisi käyttää hyvin yleiskäyttöisesti eri mittauksista saadun tiedon tallennukseen ja analysointiin, ei löydy. Tavallisen kuluttajan kannalta tämä on todennäköisesti jopa toivottu ratkaisu – näin ohjelmistot voidaan pitää yksinkertaisina ja helposti lähestyttävänä.

Käytännössä yhtään avoimen lähdekoodin ohjelmistoa, johon voisi joustavasti lisätä eri moduuleja käyttötärpeen mukaan itse ohjelmoiden, ei selvästikään ole vielä ole-

massa. Verkossa saatavilla olevien ohjelmistojen lähdekoodit eivät useinkaan ole saatavilla, mikäli sovelluksesta tarjotaan maksullista versiota. Näin esimerkiksi ohjelman ajaminen omalla web-palvelimella ei ole mahdollista.

### **3 Vaatimusmäärittely**

#### **3.1 Yleistä**

Ohjelmiston vaatimusmäärittely antaa pohjan ohjelmiston suunnittelulle. Vaatimusmäärittelyssä sovellukselle määritellään toiminnallisia ja ei-toiminnallisia vaatimuksia ja jotkin näistä vaatimuksista ovat osaltaan myös järjestelmää rajoittavia tekijöitä (Paakki 2016). Luvut 3.2-3.4 avaavat sovellukselle asetetut vaatimusmäärittelyt. Kuva-  
tut määrittelyt koskevat opinnäytetyön aikaan toteutettavaa sovellusta, eivätkä sisällä jatkokehityksen alaisia kehityskohteita.

#### **3.2 Vaatimukset koko ohjelmiston kontekstissa**

Sovellus toteutetaan avoimen lähdekoodin periaatteella käyttäen MIT-lisenssiä, jolloin kuka tahansa voi niin halutessaan ottaa sovelluksen käyttöönsä käyttämässä vapaasti jaossa olevaa lähdekoodia kuitenkin sisällyttäen alkuperäiset kopiointioikeustiedot ja lisenssihuomiot sovelluksen jokaiseen kopioon (MIT License n.d). Sovelluksen käyttöönottoon tuotetaan tarvittavan selkeät ohjeet ja oppaat, joiden avulla käyttöönotto voidaan suorittaa määrittelyn alaisille alustoille. Lisäksi työpöytäsovellukselle toteutetaan yksinkertainen asennustapa, jolla sovellus voidaan ottaa käyttöön helposti ilman suurempia määrittelyitä vain lataamalla se verkkosivustolta.

#### **3.3 Web-sovellus**

##### **3.3.1 Arkkitehtuurilliset vaatimukset**

Web-sovelluksen täytyy olla irrallinen kokonaisuus, joka asennetaan aina erikseen käyttäjän itse hankkimalle alustalle. Sovelluksen käyttöönotto tulee voida suorittaa Linux-palvelintietokoneelle kuitenkin siten, että sovellus on mahdollista asentaa Raspberry Pi 3 –minitietokoneelle ja käyttää sitä täten paikallisena asemana. Sovellus on

voitava hajauttaa useammalle palvelintietokoneelle siten, että käyttöliittymä, palvelinsovellus ja tietokanta voivat sijaita täysin irrallaan toisistaan. Sovellus tulee voida edelleen niin haluttaessa asentaa siten, että sitä voidaan käyttää omasta fyysisestä sijainnista riippumatta Internetyhteyden tämän salliessa.

### 3.3.2 Toiminnalliset vaatimukset

Web-sovellukseen täytyy voida kirjautua sisään käyttämällä käyttäjän henkilökoh- taista käyttäjätunnusta ja salasanaa. Sovelluksesta täytyy myös voida yksiselitteisesti kirjautua ulos, minkä jälkeen käyttäjän on kirjauduttava sisään uudestaan käyttääk- seen sovellusta. Käyttäjän tulee voida muokata omia käyttäjätietojaan ja vaihtaa omaa salasanaansa. Käyttäjän tulee voida lisätä datatauluja, joihin voi käydä lisäämässä da- taa käsin tai tuoda sitä sovellukseen sovelluksen ulkopuolelta. Käyttäjän tulee voida myös tarvittaessa muokata ja poistaa tallentamaansa dataa.

Sovellukseen täytyy voida tuoda dataa ulkopuolelta joko siten, että ulkopuolinen so- vellus kirjoittaa dataa toteutettavan sovelluksen avaamaan rajapintaan, tai vaihtoehtoisesti siten, että toteutettavaan sovellukseen määritellään rajapinta, jota sovellus käy automaattisesti määrätyin väliajoin lukemassa.

## 3.4 Työpöytäsovellus

Työpöytäsovelluksen tulee olla alustariippumaton siinä määrin, että se voidaan asen- taa Windows-, Linux- ja OSX-käyttöjärjestelmiin. Sovelluksen tulee voida nauhoittaa käyttäjän syöttölaitteiden tapahtumia sisältäen hiiren liikkeen, hiiren näppäinten pai- nallukset ja näppäimistön painallukset. Sovelluksen tulee voida myös tarkkailla käyttä- jän aktiivista sovellusta ja näytön resoluutiota. Sovelluksen tulee voida lähettää hank- kimansa tieto web-sovellukselle jatkokäsiteltäväksi.

Sovellus tulee voida asettaa käynnistymään käyttöjärjestelmän käynnistymisen yhtey- dessä, jolloin sen nauhoittaman datan voidaan olettaa sisältävän kaiken tietokoneen käytön aikana tehdyn aktiivisen työskentelyn aiheuttamat tapahtumat. Sovelluksen tu- lee myös kyetä hallitsemaan yhteydetön tila siten, että esimerkiksi internetyhteyden katketessa tallennettu tieto ei katoa, vaan tallentuu paikallisesti, kunnes internetyh- teys saadaan jälleen muodostettua.

## 4 Suunnittelu ja teknologiavalinnat

### 4.1 Yleistä

Ohjelmistotuotannossa erittäin tärkeässä roolissa on alkuvaiheen ohjelmistosuunnittelu. Suunnittelun myötä voidaan rakentaa pohja, joka vastaa suurimpaan osaan ohjelmoijien myöhemmän vaiheen kysymyksistä. On myös selvää, ettei sovellusta voida suunnitella kerralla alusta loppuun – suunnitelma kehittyykin jatkuvasti ohjelmistokehityksen ohella. Hyvä ohjelmistosuunnittelu toimii myös pelkkää ideana parempana pohjana sovelluksen kehitykseen kuluvan ajan ennustamiseen, mikä edelleen helpottaa projektin myymistä asiakkaalle. (Importance of Design In Software Development 2013.)

Suunnittelutyön järkevä määrä riippuu merkittävästi tekijöiden ammattitaidosta, kokemuksesta ja etenkin siitä, mitä ollaan tekemässä – jos tekijöillä ei etukäteen ole aavistustakaan, millaisia vaatimuksia toteutuksella on teknologioiden osalta, ei sovellusta kannata yrittää toteuttaa suunnittelemalla, vaan pikainen siirtyminen ”yritä ja erehdy” –tyyppiseen toteutukseen on suositeltavaa. Kokenut ohjelmistosuunnittelija osaa kuitenkin etukäteen aavistaa mahdolliset sudenkuopat ja on siten askeleen edellä.

Opinnäytetyö sisälsi kokonaisuuksia, joista aiempaa kokemusta ei ohjelmointitiimin sisällä ollut. Tällöin teknologiavalintoja ei yksinkertaisesti pystytty suunnittelemaan, vaan ne täytyi toteuttaa täysin intuitiivisesti kokeilemalla eri vaihtoehtoja ja valitsemalla näistä edelleen tarkoituksenmukaisin. Tällainen teknologinen mysteeri oli esimerkiksi tietokoneelle asennettava sovellus, jonka tuli pystyä globaalissa kontekstissa kuuntelemaan käyttäjän tietokoneeseen kytkemiä syöttölaitteita sekä käyttäjän käyttämiä sovelluksia.

## 4.2 Arkkitehtuurilliset päälinjat

### 4.2.1 Yleistä

Sovellusarkkitehtuurin valinnassa on ensimmäisenä seikkana otettava huomioon vaatimusmäärittely, joka antaa pohjan ja suurpiirteisen rajauksen toteutettavan sovelluksen kokonaisratkaisuille. Muita rajaavia osatekijöitä arkkitehtuuriratkaisuissa ovat yleensä myös raha, kohdekäyttäjätyyppi ja esimerkiksi halutun siirrettävyyden ja modulaarisuuden taso – mikäli sovelluksesta halutaan helposti alustalta ja koneelta toiselle siirrettävä, täytyy myös pohjalla olevien ratkaisuiden tukea tätä tavoitetta.

Opinnäytetyössä sovelluksen luonne määriteltiin web-sovellukseksi ja siitä eriytetyksi työpöytäsovellukseksi jo vaatimusmäärittelyssä. Lisäksi vaatimusmäärittely totesi edelleen sen, että web-sovellus tulee voida asentaa Linux-palvelinympäristöön. Tämä rajasi edelleen käytössä olevien mahdollisten ratkaisuiden määrää.

### 4.2.2 Oma palvelin vai pilvitietokanta palveluna?

Perinteisesti web-sovelluksen arkkitehtuuriin kuuluvat selainsovellus, palvelinsovellus, sekä tietokanta, johon sovelluksen käyttämä data tallennetaan. Nykyään kuitenkin markkinoita ovat vallanneet erilaiset datantallennuspalvelut, jotka abstrahoivat perinteisesti itse ohjelmoidun palvelinsovelluslogiikan siten, että tietynlaisissa projekteissa ei välttämättä tarvita enää lainkaan omaa palvelininfrastruktuuria.

Esimerkkinä modernista pilvipalvelusta voidaan pitää Firebasea, joka tarjoaa monipuolisesti palveluita datan tallennukseen sekä sovelluksen käyttäjien analysointiin. Firebase tarjoaa myös joukon erilaisia autentikointivaihtoehtoja sekä mahdollisuuden lisätä datakohtaisia käyttöoikeusrajoituksia, joilla ohjelmoija voi rajata sovelluksessa näkyvää dataa käyttäjän oikeuksien perusteella. (Documentation 2016.)

Verrattaessa perinteistä tapaa toteuttaa palvelinohjelmointi itse ja uutta, palvelumuotoista datantallennusta voidaan huomata, että uusi malli vähentää huomattavasti erilaista konfiguroinnin tarvetta ja ylläpidon määrää. Edelleen verraten esimerkiksi näiden kahden tavan hintaa suhteellisen pienien sovellusten osalta voidaan huomata,

että käytännössä kumpikin on yhtä edullinen. Useimmiten Firebasen kaltaisissa palveluissa voi riittävän pienen sovelluksen datan varastoinnin hoitaa jopa täysin ilmaiseksi (Pricing 2016).

### **Vanhassa vara parempi?**

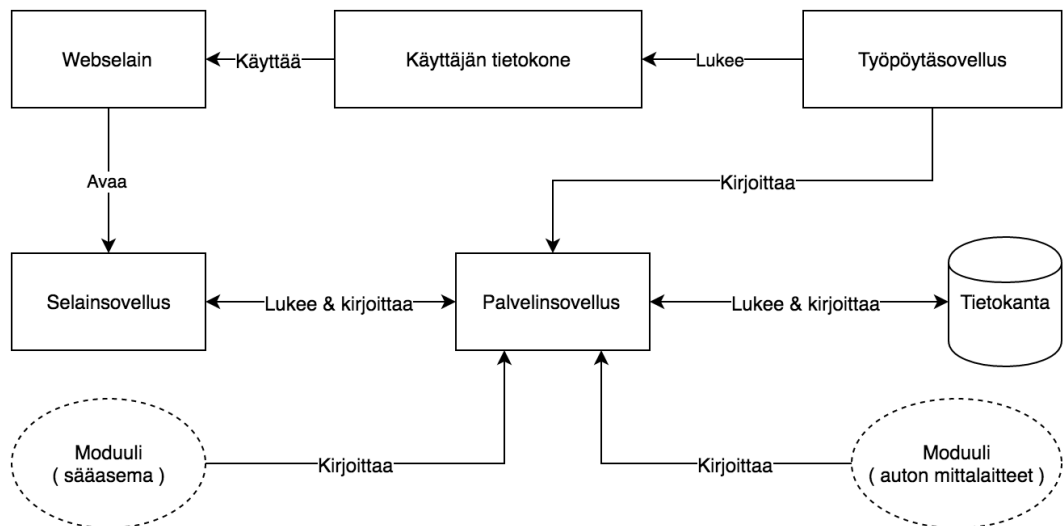
Miksi kukaan enää edes haluaisi hankkia muutamalla dollarilla kuussa oman palvelimen, asentaa sinne kaiken tarvittavan ja konfiguroida ensin päivän verran, jotta saadaan mitään ylipäänsä toimimaan, kun kaiken tämän voi ohittaa vain lisäämällä muutamaman rivin omaa asiakassovelluksen lähdekoodiin?

Perinteisen mallin tukemiseen on löydettävissä useitakin eri syitä. Usein halutaan, että kaikki langat ovat omissa käsissä. Kun langat luovuttaa jonkin kolmannen osapuolen käsiin käytännössä täysin, on niistä itse vaikea lähteä enää nykimään. Entä jos käytetyn palvelun rajat tulevatkin vastaan jossain, jota ei etukäteen osattu arvatakaan? Tällöin joudutaan kuitenkin ottamaan omaa palvelinympäristöä käyttöön pelkästään tätä kustomoitua toimintoa varten. Myös esimerkiksi internetyhteyden tarve voi olla rajaava tekijä joissain projekteissa, vaikkakin esimerkiksi Firebase sisältää hyvän tuen yhteydettömässä tilassa operoimiseen.

Opinnäytetyössä toteutettiin datan varastointi ja ohjelmoitiin palvelinsovellus käyttämällä omaa palvelinta, jonne pystytettiin tietokanta tiedon tallennusta varten. Tällöin voitiin olla lähestulkoon varmoja siitä, että järjestelmä itsessään ei tule asettamaan rajoituksia toteutukselle. Myös vaatimusmäärittelystä ilmennyt vaade sovelluksen asennuksen mahdollistamiseksi internetyhteydettömään tilaan saatiin tällä tavoin toteutumaan.

#### **4.2.3 Järjestelmän kokonaisrakenne ja toimintaperiaate**

Opinnäytetyössä toteutettu järjestelmä koostuu web-sovelluksesta ja työpöytäsovelluksesta, sekä muista joukkoon toteutetuista demomoduuleista. Web-sovellus muodostuu taas edelleen selainsovelluksesta, palvelinsovelluksesta ja tietokannasta. Järjestelmän kokonaisrakenne ja yhteydet ovat hyvin korkealla tasolla graafisesti avatuna kuviossa 2.



Kuvio 2. Sovelluksen kokonaisrakenne ja yhteydet

## 4.3 Datavaroitointi

### 4.3.1 Yleistä

Web-sovelluksen toimintaan liittyvät yleisesti data ja datan käsittely. Datavaroitukseen tarvitaan yleensä jokin ulkoinen tietovarasto, joka on täysin irrallaan käyttäjän laitteista. Näin dataa voidaan käsitellä sovelluksen kontekstissa ilman, että data on sidoksissa käytettävään päätelaitteeseen tai käyttäjän laitteen fyysiseen sijaintiin. Datavaroituksesta web-sovelluskontekstissa käytetään yleisesti nimitystä *tietokanta*.

Tietokannat voidaan jakaa karkeasti kahteen luokkaan: relaatio- ja epärelaatiotietokantoihin. Valittu tietokantatyyppi määräytyykin tarvittavien ominaisuuksien perusteella.

#### Relaatiotietokannat

Relaatiotietokannat sisältävät nimensä mukaisesti tauluja, joiden välille muodostetaan liitoksia eli suhteita. Relaatiotietokannoissa tieto on yleensä pakotettuna tiettyyn muotoon, jolloin datan laatua tarkkaillaan jo itse tietokannan tasolla. Tietty tietokantamootorit valvovat jopa taulujen välisiä liitoksia ja pitävät näin huolta tietokannan eheydestä. Kuviossa 3 suhteen USER- ja POST-taulujen välillä määrittää POST-taulun

USER\_ID-kenttä. Tyypillistä relaatiotietokannalle on se, ettei tietokannassa toisteta samaa tietoa turhaan vaan taulut ovat *normalisoituja* siten, että tieto esiintyy tietokannassa vain yhden kerran. (Gentz 2016.)

USER			
ID	USERNAME	FIRSTNAME	LASTNAME
1	jdoe	John	Doe
2	mameik	Matti	Meikäläinen

POST			
ID	USER_ID	CONTENT	CREATED_AT
1	1	Hi!	2016-12-01 12:23:22
2	2	Oh hi	2016-12-01 12:23:32
3	1	How are you?	2016-12-01 12:23:49
4	2	Very, very nice.	2016-12-01 12:23:58

Kuvio 3. Esimerkki relaatiotietokannan tauluista

### Epärelaatiotietokannat

Epärelaatiotietokannoissa dataa ei pakoteta tiettyyn muotoon, vaan tallennettu tieto voi olla sisäisesti keskenään hyvin poikkeavaa. Yleinen epärelaatiotietokannan muoto on niin kutsuttu dokumenttitietokanta, jossa tieto tallennetaan yleensä JSON-tyyppisenä dokumentteihin. Epärelaatiotietokannassa ei määritellä datan välisiä suhteita avaimilla, vaan suhteet muodostetaan asettamalla emäobjektin sisään aliobjekteja. Kuviossa 3 kuvattu data-asetelma relaatiotietokannassa esitetään JSON-tyyppisessä epärelaatiotietokannassa kuvion 4 esittämällä tavalla. (Gentz 2016.)

```
[
  {
    "username": "jdoe",
    "firstname": "John",
    "lastname": "Doe",
    "posts": [
      {
        "content": "Hi!",
        "createdAt": "2016-12-01 12:23:22"
      },
      {
        "content": "How are you?",
        "createdAt": "2016-12-01 12:23:49"
      }
    ]
  },
  {
  }
```

Kuvio 4. Esimerkkidata esitettynä epärelaatiotyyppisenä

Kuviosta 4 huomataan, ettei objekteilla ole mitään tunnistettavaa arvoa. Tämän puutteen korvaa yleensä epärelaatiotietokantajärjestelmä, joka automaattisesti generoi

kaikille tietokantaan tallennetuille objekteille yksilöllisen avaimen, jonka avulla ne voidaan tunnistaa.

Datavarastoille asetettavat vaatimukset riippuvat suurimmaksi osaksi sovelluksen luonteesta ja karkea rajausta voidaan vetää sen mukaan, millaista dataa sovellukseen halutaan tallentaa. Epärelaatiotietokannat ovat suhteessa helpommin skaalattavia ja joustavampia, jolloin ne ovat myös ohjelmistokehityksen suhteen ketterämpiä kuin perinteiset relaatiotietokannat. Epärelaatiotietokantoja käytetäänkin usein nopeasti kehittyvissä ketterissä sovelluksissa ja sellaisessa ympäristössä, jota on voitava joustavasti skaalata pienelle tai suurelle käyttäjäryhmälle. (Gentz 2016.)

Epärelaatiotietokantoihin mutkikkaampien kyselyiden tekeminen on kuitenkin vaikeaa ja kömpelöä, minkä vuoksi relaatiotietokantojen suosio kohdistuu voimakkaasti rakenteellisen datan käsittelyyn. Relaatiotietokantojen pitkän historian aikaan niihin on myös ehditty toteuttaa suuri määrä erilaisia sisäänrakennettuja toimintoja, jotka epärelaatiotietokannoista niiden tuoreuden vuoksi edelleen puuttuvat.

#### 4.3.2 Tietokantaratkaisut

Opinnäytetyössä toteutetun sovelluksen datan varastointi toteutettiin käyttämällä perinteistä relaatiotietokantaa. Relaatiotietokannan valintaan johti epävarmuus siitä, miten vaikeaa laskentaa jossain vaiheessa sovelluksen kehitystä voidaan joutua tekemään. Vaikkakin epärelaatiotietokanta saattaisi itsessään olla nopeampi ja ketterämpi suhteessa relaatiotietokantaan, on erilaisten laskentojen pakkaaminen tietokannan sisään ja tietokantakyselyihin nopeampaa ja helpompaa, kuin tehdä laskentaa sovelluslogiikassa.

Tietokannanhallintajärjestelmäksi valittiin MariaDB, joka on avoimen lähdekoodin jatke MySQL-tietokannanhallintajärjestelmälle. MariaDB käyttää oletuksena InnoDB-tietokantamoottoria, joskin mahdollista on käyttää myös koko joukkoa muita tietokantamoottoreita (XtraDB and InnoDB 2016). InnoDB on hyvä yleiskäyttöinen moottori moniin kohteisiin sisältäessään muun muassa transaktiokäsittelyt ja tietokannan sisäisiä keinoja ylläpitää datan viite-eheyttä rajoitteilla (constraint) ja viite-avaimilla (foreign key) sekä vyörytyssäännöillä (cascade).

## 4.4 Web-palvelinohjelmoinnin teknologiat

### 4.4.1 Yleistä

Web-palvelinohjelmoinnilla tarkoitetaan sitä web-ohjelmoinnin osaa, jossa ohjelmoidaan sovellusta web-palvelimella ajettavaksi. Web-palvelinsovellus on sen sijaan sovellus, jonka ohjelmakoodia ajetaan web-palvelimella käyttäen hyväksi palvelintietokoneen resursseja. Tämä on selkein ero *asiakassovellukseen*, joka ajetaan täysin käyttäen käyttäjän päätelaitteen resursseja. (Client Side vs. Server Side n.d.)

### 4.4.2 Palvelinkäyttäjärjestelmät

Palvelinkäyttäjärjestelmä on käyttöjärjestelmä, joka on toteutettu erityisesti toimimaan asiakas-palvelin-tyyppisessä arkkitehtuurissa. Palvelinkäyttäjärjestelmiä on monenlaisia, ja niiden ominaisuudet poikkeavat hieman toisistaan. Ei liene olemassa yhtä syytä valita jokin tietty käyttöjärjestelmä, ja palvelinkäyttäjärjestelmien suhteen pätee myös sama merkkiuskollisuus, mikä tavallisten työpöytäkäyttöjärjestelmienkin suhteen: mikäli jokin käyttöjärjestelmä on koettu hyväksi ja toimivaksi, ei siitä liene mitään syytä siirtyä pois.

Yleisesti suosiossa ovat kuitenkin olleet helposti saatavilla olevat ja pienen aloituskynnyksen omaavat Linux-pohjaiset käyttöjärjestelmät, kuten Ubuntu, SUSE, CentOS ja Debian. Linux-pohjaisille käyttöjärjestelmille on myös tarjolla suunnattomasti enemmän erilaisia sovelluskehysjä ja teknologioita sen open source -suosion myötä. Linux-pohjaiset toteutukset ovat yleensä myös hyvin siirrettäviä alustalta toiselle, koska käytännössä kaikki Linux-distributiot pohjautuvat samaan ytimeen sisältäen vain hieman toisistaan poikkeavat kirjastot ja käyttöliittymät.

Opinnäytetyössä sovelluksen vaatimusmäärittelyssä sanotaan, että sovellus on voitava asentaa Linux-serverille ja sen on kyettävä toimimaan myös Raspberry Pi -minitietokoneella. Jo nämä ehdot rajaavat palvelinkäyttäjärjestelmäksi jonkin Linux-distribution. Koska ehdottomasti suurin osa saatavilla olevasta dokumentaatiosta alustavan tietojenetsinnän perusteella on suunnattu Ubuntu-käyttöjärjestelmälle, käytettiin opinnäytetyössä palvelinkäyttäjärjestelmänä Ubuntu Serverin versiota 16.04.

#### 4.4.3 HTTP-palvelinohjelmat

HTTP-palvelinohjelma on palvelinkäyttöjärjestelmään asennettava sovellus, jonka pääasiallinen tarkoitus on vastaanottaa asiakassovelluksilta tulevia pyyntöjä ja vastata niihin toivotulla tavalla (Eftaiha 2012). HTTP-palvelinohjelmia on saatavilla Linux-ympäristöön useita, mutta ehdottomasti kaksi suosituinta HTTP-palvelinohjelmaa ovat Apache Software Foundationin Apache ja NGINX Softwaren NGINX. Näistä kahdesta Apachella on huomattavasti pidempi historia ja näin ollen vakaa ja laaja käyttäjäkunta, mutta NGINX on keveydellään kirinyt välimatkaa kiinni ollen toiseksi suosituin HTTP-palvelinohjelma.

Apachen ja NGINXin erot ovat käytännön tasolla melko pieniä ja opinnäytetyössä ohjelman tehtävänä onkin käytännössä staattisten sivujen tarjoaminen, eikä työhön tulisi kuulua sen suuremmin palvelinohjelman säätämistä. Käytännön erot näiden välillä muodostuvat siitä, miten ohjelmia konfiguroidaan suorittamaan erilaisia operaatiota, kuten esimerkiksi sivujen uudelleenohjaamista, näkyvän osoitteen formatointia ja virtuaalipalvelinten määrittämistä. (Ellingwood 2015.)

Opinnäytetyössä HTTP-palvelinohjelman virkaa valittiin opinnäytetyön suorittamisen ajaksi hoitamaan Apache-palvelinohjelma sen laajan yleisen suosion ja dokumentaation määrän vuoksi.

#### 4.4.4 Palvelinohjelmointikielet

Palvelinohjelmointia suoritetaan palvelinohjelmointikielellä. Nykyään palvelinohjelmointikielenä voidaan käyttää käytännössä mitä tahansa ohjelmointikieltä, mutta yleisimmin käytössä lienevät perinteisimmät PHP, ASP.NET (VB, C#) ja Java. Uutena tulokkaana tähän joukkoon voitaneen nostaa JavaScript, joka Node.JS:n myötä on nopeasti noussut suureen suosioon palvelinohjelmoinnin saralla. Seuraavat kappaleet käsittelevät lyhyesti eri palvelinohjelmointikieliä pohjustaen opinnäytetyössä käytetyn palvelinohjelmointikielen valintaa.

## **PHP**

PHP (PHP: Hypertext Preprocessor, alun perin Personal Home Page) on hyvin yleisesti käytetty avoimen lähdekoodin palvelinohjelmointikieli, jonka ensimmäinen versio julkaistiin vuonna 1995. PHP on heikosti tyypitetty kieli, joskin PHP:n versioon 7 on tuotu mukaan vahva tyyppitys. PHP on yleisimmin dynaamisissa web-sovelluksissa käytetty palvelinohjelmointikieli sen laajan tuen ja hyvän dokumentaation ansiosta. (Pstatz 2010.)

## **ASP.NET**

ASP.NET on Microsoft-tuoteperheen avoimen lähdekoodin web-ohjelmointisovelluskehys. ASP.NET tukee ohjelmointikielinsä Visual Basicia, C#:a ja J#:a. ASP.NET on käytössä yleisesti silloin, kun palvelinympäristöinäkin käytetään Microsoftin tuotteita.

## **Java**

Javaa voidaan pitää jo suorastaan historiallisena tekijänä palvelinohjelmoinnin saralla. Se on ollut palvelinohjelmointikielten suosiotarkastelussa enemmänkin joukon keskivaiheilla todennäköisesti siitä syystä, että Javaa on hyvin hankala käyttää palvelinohjelmointiin (Yank 2001). Javalle on toteutettu suuri määrä eri ohjelmointisovelluskehyksiä, jotka mahdollistavat tiedon sitomisen suoraan palvelinohjelman ja käyttöliittymän välillä ilman sen suurempaa omaa sovelluslogiikkaa.

## **Node.js**

Node.js on palvelinohjelmoinnin saralla uusinta uutta. Node.js on asynkroninen tapahtumapohjainen JavaScript-palvelin, joka on suunnattu nimenomaan skaalattavien web-sovelluksien toteutukseen (Capan 2013). Node.js:n käyttämä paketinhallintajärjestelmä, npm (node package manager), on maailman suurin paketinhallintajärjestelmä lähestulkoon puolella miljoonalla paketillaan.

## **Yhteenvetona**

Kun eri palvelinohjelmointikieliä ympäristöineen vertaillaan keskenään ja peilataan sitä siihen, mitä ollaan tekemässä, voidaan huomata, että kaikki näistäkin ohjelmointikielistä pyrkivät täsmälleen samaan asiaan – tietoa käsitellään, tallennetaan ja luetaan tietokannasta.

Opinnäytetyössä palvelinohjelmointiteknologiana käytettiin PHP:tä. Valintaan johtivat muun muassa sovelluksen siirrettävyyksivaatimus alustalta toiselle ja PHP:n sisältämät valmiit funktiot ja ominaisuudet. Toisaalta PHP:n puoleen veti myös henkilökohtainen mielenkiinto uutta PHP:n versiota ja sen tuomia uusia ominaisuuksia kohtaan.

#### 4.4.5 Sovelluskehukset

Sovelluskehys on kokoelma sovelluskomponentteja, jotka yhdessä ovat sisäisesti toteutettu ratkaisemaan jokin tietty ohjelmoinnillinen kokonaisuus. Luokkakirjastona voidaan taas pitää kokoelmaa luokkia, joita ohjelmoija voi käyttää ohjelmointityössään kutsumalla niitä omasta lähdekoodistaan. Sovelluskehys onkin luokkakirjastoista poiketen parametrisoinnin jälkeen yleensä valmis toimiva kokonaisuus, jonka ohjelmointi tapahtuu laajentamalla ohjelmoijalle avointa sovelluskehysten rajapintaa. (Juslin 1999.)

PHP:lle on tarjolla suunnaton määrä sovelluskehysä, jotka tarjoavat yksilöllisesti ominaisuuksia riippuen täysin siitä, mihin tarkoitukseen sovelluskehys on suunnattu. Sovelluskehystä valittaessa täytyy huomioon ottaa ainakin siirrettävyys, testattavuus ja dokumentaation saatavuus. Kehyksen täytyy myös tukea valittua arkkitehtuuria siinä määrin, että se tukee jonkinlaista rajapintaratkaisua, jonka kautta viestintä selainsovelluksen ja palvelinsovelluksen välillä voidaan hoitaa. Seuraavat kappaleet avaavat muutamia eri vaihtoehtoja sovelluskehysiksi PHP-ohjelmointikielellä käytettäväksi.

#### Yii

Yii on avoimen lähdekoodin web-sovelluskehys, joka on omien sanojensa mukaan suorituskyvyltään äärimmilleen optimoitu täydellinen valinta kaikenlaisiin ja kaikenkokoisiin web-sovellusprojekteihin. Yii toteuttaa MVC-arkkitehtuuria ja tarjoaa ohjelmoijalle nipun erinäisiä apuvälineitä ohjelmoinnin tueksi, sisältäen esimerkiksi DAO:t (Database Access Object), kyselynmuodostajat (Query Builder), tietokantamigraatiot (DB Migrations), syötteen validoinnin ja tuen yksikkötesteille (Yii Framework 2016). Yii:n dokumentaatio on hieman sekava ja siitä löytyvä informaatio vaihtelee merkittävästi lähteiden välillä.

## **Symfony**

Symfony on helposti laajennettava sovelluskehys, joka on lähtökohtaisesti toteutettu sillä ajatuksella, että jokainen kehyksen osa on erillinen lisättävä palvelu. Symphonyyn on ajan mittaan toteutettu suuri määrä kolmannen osapuolen komponentteja hyvin moninaisten toimenpiteiden suorittamiseen. Symfony antaa mahdollisuuden edelleen kirjoittaa omaa ohjelmakoodia rajoittamatta ja pakottamatta ohjelmoijaa käyttämään Symfonyn itsensä tarjoamia vaihtoehtoja. (Symfony explained to a Developer 2016.)

## **Laravel**

Laravel on pääosin Symfonyn komponentteja käyttäen koottu sovelluskehys, joka mahdollistaa helposti luettavan ohjelmakoodin kirjoittamisen lukuisilla komponenteillaan. Laravel myös pienentää yleisimmin paljon aikaa vieviin kokonaisuuksiin kuluvaan aikaa helpottamalla esimerkiksi autentikointia, sessiökäsittelyä ja välimuistikäsittelyä (cache) (Introduction To Laravel Framework 2016).

## **Yhteenvetona**

Edellä mainitut kolme PHP-sovelluskehystä ovat vain kolme eniten hakutuloksiin nousevaa sovelluskehystä. Samankaltaisia toteutuksia on kuitenkin monin verroin lisää, joista jokaisessa on kiistämättä omat etunsa ja heikkoutensa. Opinnäytetyössä palvelinohjelmoinnin sovelluskehyyksiä valittiin Symfony. Verratessa eri sovelluskehyyksiä ja näiden omia kotisivuja keskenään, ei voi olla huomaamatta, miten ylivertaisen helpolukuinen ja kattava Symfonyn dokumentaatio on. Dokumentaatiota selatessa voidaan todeta, että Symfony sisältää paljon sellaisia komponentteja, joita voidaan etukäteen ajatella tarvitsevan toteutettavan sovelluksen luonnetta mietittäessä. Myös asennusohjeet ovat kattavat niin kehitys- kuin tuotantoympäristöäkin ajatellen.

### **4.4.6 Yhteenveto**

Kun summataan palvelinsovelluksen toteuttamiseen valitut sovellukset yhteen, havaitaan, että sovelluksista muodostuu niin kutsuttu LAMP-kokoelma. LAMP on hyvin yleis-

sessä suosiossa oleva avoimen lähdekoodin sovelluksista muodostuva kokoelma. LAMPin sisältämät sovellukset muodostavat yhdessä web-sovelluspalvelimen, jota voidaan käyttää web-sovellusten ajoalustana. LAMP muodostuu sanoista Linux, Apache, MySQL/MariaDb ja PHP/Perl/Python (Brown 2005).

## 4.5 Web-selainsovellus

### 4.5.1 Yleistä

Web-selainsovelluksella tarkoitetaan sitä sovelluksen osaa, jonka käyttäjä näkee selainohjelmassaan. Selainohjelmointitekologioita on olemassa paljon, mutta käytännön tasolla kaikista korkeamman tason selainohjelmointitekologioista generoituu aina joko HTML:ää (HyperText Markup Language), CSS:ää (Cascading Style Sheet) tai JavaScriptiä. Tällä tavoin menetellään siksi, että käytännössä selaimet tukevat vain näitä teknologioita. Viime aikoina korkeamman tason selainohjelmointiteknologiat ovat vallanneet alaa, mutta myös perinteisillä menetelmillä on yhä oma kannattajakuntansa.

Selainkäyttöliittymän teknologiavalinta ei ole vain yksi valinta. Selainsovellus on itsessään myös yksi sovellus, joka voidaan jakaa vähintäänkin kahteen eri osioon: itse käyttöliittymään ja taustalla pyörivään sovelluslogiikkaan. Käyttöliittymä muodostuu HTML- ja CSS-määrittelyistä, kun taas selainsovelluksen sovelluslogiikka ohjelmoidaan käyttämällä JavaScriptiä, tai jotain muuta vastaavaa ohjelmointikieltä tai -ympäristöä, joka lopputuloksenaan tuottaa selaimen ymmärtämää JavaScriptiä.

### 4.5.2 Selainsovelluskehyksiä

Selainohjelmointi on yksinkertaisimmillaan sitä, että *script*-tagin sisään kirjoitetaan JavaScriptiä, joka sivun latautumisen jälkeen suoritetaan sivustolla. Ohjelmakoodi voi esimerkiksi liikuttaa, muokata, tyhjentää, tai poistaa HTML-elementtejä, tai se voi vaihtoehtoisesti kysyä jotain syötettä käyttäjältä. Yksinkertaisimmillaan selainohjelmointi ei tarvitse mitään kirjastoja tai sovelluskehyksiä, mutta ylläpidettävyyden kannalta sellaisen käyttö on monessa tilanteessa järkevää. Selainsovelluskehukset tarjoavat yleensä laajuudesta riippuen keinoja datan sidontaan komponentin ja ohjelmakoodin välillä, sekä erilaisia käärintäfunktioita muutoin vaikeasti toteutettaville JavaScript-

ohjelmoinnin osioille. Seuraavissa kappaleissa käydään läpi muutamia suosittuja eri selainsovelluskehysjä pohjustaen selainsovelluskehysten valintaa.

### **AngularJS**

AngularJS on Googlen tukema avoimen lähdekoodin JavaScript-sovelluskehys, joka mahdollistaa selainohjelmoinnin käyttämällä MVC-arkkitehtuuria. Angular on erittäin suosittu sen aktiivisen kehityksen ja ylläpidon vuoksi ja näin ollen sille on helppo löytää erilaisia ohjeita ja oppaita. Angulariin on toteutettu myös suuri määrä kolmannen osapuolen kirjastoja, jotka osaltaan tekevät Angularin käytöstä joustavampaa. Angular sisältää lisäladattavina komponentteina esimerkiksi HTTP-pyyntöjen ja animaatioiden käsittelyyn. Angularin ensimmäisen version ohjelmoimiseen käytetään perinteistä JavaScriptiä, joka tekee siitä hyvin upotettavan käytännössä mihin tahansa web-sovellusprojektiin (AngularJS - Overview n.d.).

### **KnockoutJS**

KnockoutJS on kevyt JavaScript-sovelluskehys, jonka pääasiallinen käyttökohde on datansidonta käyttöliittymäkomponenttien ja ohjelmakoodin välillä. KnockoutJS ei tuo varsinaisesti mitään muuta toiminnallisuutta sovelluslogiikkaan ja toimiikin näin ollen enemmänkin datansidontakirjastona kuin varsinaisena sovelluskehysenä. (Introduction To Knockout n.d.)

### **React**

React on Facebookin ylläpitämä JavaScript-sovelluskehys käyttöliittymäohjelmointiin. Reactissa käännetään DOMin (Document Object Model) muokkaus päälaelleen käyttämällä virtuaalista DOMia. Sen sijaan, että käyttöliittymäkomponentit merkittäisiin kaikki HTML-dokumenttiin ja sovelluskehys kytkettäisiin DOMiin käyttämällä HTML-elementtien attribuutteja, Reactissa elementti tehdään sovelluslogiikan puolella, jolloin elementtiin liitetään kaikki mahdolliset tapahtumankäsittelijät ja datansidonta. Käyttöliittymän päivitys tapahtuu edelleen vertaamalla luotua virtuaalista ja alkuperäistä DOMia keskenään ja vaihtamalla muuttuneet elementit (What Is React? n.d.).

### **Yhteenvetona**

Opinnäytetyössä selainohjelmoinnin sovelluskehysenä käytettiin AngularJS:ää sen laajan tuen ja laajennettavuuden vuoksi. AngularJS:n dokumentaatio on hyvin selkeä

ja mahdollisiin ongelmiin löytyy reilusti ennakkotapauksia sen laajan käyttöasteen ansiosta. Angular mahdollistaa myös käyttöliittymäkomponenttien uudelleenkäyttämisen, joka pienentää tarvittavan koodin määrää huomattavasti.

#### 4.5.3 Käyttöliittymän ehostaminen

Käyttöliittymän ehostamiseen on saatavilla monenlaisia erilaisia toteutuksia. Suurin osa näistä kirjastoista toimii sillä periaatteella, että luokitellaan käyttöliittymäkomponentteja eri luokkamääreillä, joiden mukaan ne saavat erilaisia toiminnallisia ja visuaalisia ominaisuuksia. Käytännön tasolla kirjasto voi siis olla yksi tyylitiedosto, joka lisätään web-sivulle mukaan. Tällaisia kirjastoja ovat mm. *Materialize* ja *Bootstrap*, joka lienee tällaisista yleiskäyttöisistä css-pohjaisista käyttöliittymäkirjastoista kaikkein suosituin. Tällaisen kirjaston valinnassa on kyse lähinnä mausta – kaikki toimivat käytännössä samalla idealla, mutta lopputulos on eri näköinen.

Toisaalta saatavilla on myös kokonaisia käyttöliittymäkomponenttikirjastoja. Nämä kirjastot toteuttavat täysin uusia käyttöliittymäkomponentteja sen sijaan, että vain tyylittelisivät web-selaimen tarjoamia oletuselementtejä. Käyttämällä täysin eri komponentteja, jotka ovat toteutettu täysin irrallisina JavaScriptiä käyttäen, voidaan saada aikaan halutun kaltainen toiminta jollekin komponentille, jonka oletustoiminta ei jostain syystä miellytä.

Opinnäytetyössä käyttöliittymän komponenttikirjastoksi valittiin *Angular Material* -käyttöliittymäkomponenttikirjasto, joka toteuttaa *Material Design* -käyttöliittymäideologiaa tarjoamalla ohjelmoijan käyttöön koko joukon erinäisiä Angular-direktiivejä. Osa näistä hyödyntää olemassa olevaa web-selaimen toiminnallisuutta, mutta joukossa on myös merkittävä määrä täysin erikseen toteutettuja ominaisuuksia. Material Design on asettelultaan hyvin kevyt ja väljä, sekä sen värit ovat jo vakiona keskenään hyvin sointuvat.

### 4.6 Työpöytäsovellus

#### 4.6.1 Yleistä

Opinnäytetyössä toteutetulle työpöytäsovellukselle antaa pohjan sen vaatimusmäärittely. Vaatimusmäärittelyssä todetaan, että sovelluksen tulee kyetä nauhoittamaan

päätelaitteeseen kytkettyjen syöttölaitteiden tapahtumia ja käyttäjän käyttämiä sovelluksia globaalissa kontekstissa Windows-, OSX- ja Linux-käyttöjärjestelmissä. Monialustaisuus on yleensä vaikea toteuttaa yksinkertaisella tavalla, ja kun kyse on edelleen pääsystä tietokoneen syöttölaitteisiin, on tehtävä aikaisempaakin hankalampi.

#### 4.6.2 Teknologiavaihtoehdot

Monialustaisia sovelluskehyskiä työpöytäsovelluksille on verrattain vähän. Monialustaiseen toteutustapaan lähdettäessä luovutaan aina jostain, koska kaikkea tietyllä käyttöjärjestelmällä mahdollistettua toimintaa ei välttämättä voita taata myös kaikilla muilla käyttöjärjestelmillä. Vaihtoehtoisia tapoja monialustaisten sovellusten toteuttamiseen on avattu seuraavissa kappaleissa.

##### **Java**

Kaikista monialustaisista sovelluskehyskiistä vanhin ja kokenein lienee Java, jonka pohjimmainen ideologia pitää sisällään käyttöjärjestelmärajapinnan abstrahoimisen siinä määrin, että itse sovellus on jopa täysin suoraan siirrettävissä alustalta toiselle. Java suorittaa sovellusta omalla virtuaalikoneellaan, jonka täytyy olla asennettuna käyttäjän laitteelle sovelluksen toimimiseksi. Javan heikkouksiin kuuluu myös käyttöliittymien todella vaikea kehitys. (Cross Platform Development For Desktops -- 2016.)

##### **QT**

QT on ohjelmointikirjasto, jolla ohjelmoidut sovellukset voidaan kääntää erikseen kaikille tarvittaville käyttöjärjestelmille. QT käyttää ohjelmointikielensä C++:aa, joten siltä osin kirjasto on vakaalla pohjalla. QT on kuitenkin suhteellisen vähän käytetty, eikä kolmannen osapuolen kirjastoja juuri ole saatavilla, joten monessa tilanteessa erikoisempien ratkaisujen toteuttaminen saattaa olla haastavaa. (Cross Platform Development For Desktops -- 2016.)

##### **NWJS**

NWJS, joka aiemmin tunnettiin nimellä node-webkit, on täysin varusteltu monialustainen sovelluskehys, jossa hyvin karkeasti käytännön tasolla on yhdistetty Node.js ja Chromium-selain. Tällöin sovelluskehityksessä käyttöliittymän toteuttamiseen voi-

daan käyttää tehokkaita web-teknologioita ja säilyttää silti yhteys käyttöjärjestelmätasolle Node.js:n kautta. NWJS:n tukena on myös siis mittava määrä eri Node.js:lle toteutettuja ensimmäisen ja kolmannen osapuolen kirjastoja, joilla käyttäjä voi päästä käsiksi esimerkiksi tiedostojärjestelmään ja tietokoneeseen kytkettyihin laitteisiin. (Cross Platform Development For Desktops -- 2016.)

#### 4.6.3 Yhteenveto

Opinnäytetyössä työpöytäsovellukseen valittavalla teknologialla ei ollut varsinaisesti niin suurta merkitystä kuin esimerkiksi palvelinsovellukseen käytettävillä sovelluskehysillä, koska työpöytäsovelluksen idea on lähinnä pysyä piilossa käyttäjän huomiopalkissa, nauhoittaa tietoa taustalla ja lähettää sitä edelleen web-sovellukselle analysoitavaksi.

Avoimen lähdekoodin suosion myötä on todennäköistä, että kaikista joustavimmat ja kekseliäimmät ratkaisut ovat toteutettuna Node.js:lle, joskin ne usein hyödyntävät esimerkiksi Javaa siihen toteutettujen alustariippumattomien ratkaisuiden vuoksi. Toteutus suunnattiinkin lähtökohtaisesti toteutettavaksi NWJS-ympäristöön, jolloin voidaan mahdollisesti hyödyntää selainsovellukseen ohjelmoitavaa sovelluslogiikkaa käytettäessä AngularJS:ää molempien sovellusten, niin varsinaisen selainsovelluksen kuin työpöytäsovelluksenkin ohjelmointiin.

### 4.7 Rajapintaratkaisut

Kun sovellus toteutetaan selkeästi osioituna täysin eri kokonaisuuksiin käyttöliittymän ja web-palvelinsovelluksen osalta, tarvitaan jonkinlainen ohjelmointirajapinta, jota hyväksi käyttäen nämä sovelluksen eri osat voivat viestiä keskenään. Rajapinta on hyvin väljä käsite ja se tarkoittaa yleisesti kaikkia sellaisia väyliä, jolla sovelluksen osat tai eri sovellukset voivat vaihtaa keskenään tietoa tai jonka kautta niitä voidaan hyödyntää.

Web-sovelluksissa käytetään hyvin yleisesti muutamaa eri tyyppistä rajapintaratkaisua. Joskus rajapinta toteutetaan hyvin tiukasti sidottuna sovelluksen sisään siten, ettei sitä ole tarkoituskaan käyttää toisaalta yleiskäyttöisesti. Usein kuitenkin kaivataan yleiskäyttöisempää rajapintaa, jota voidaan mahdollisesti käyttää myös pääasiallisen sitä käyttävän sovelluksen ulkopuolelta.

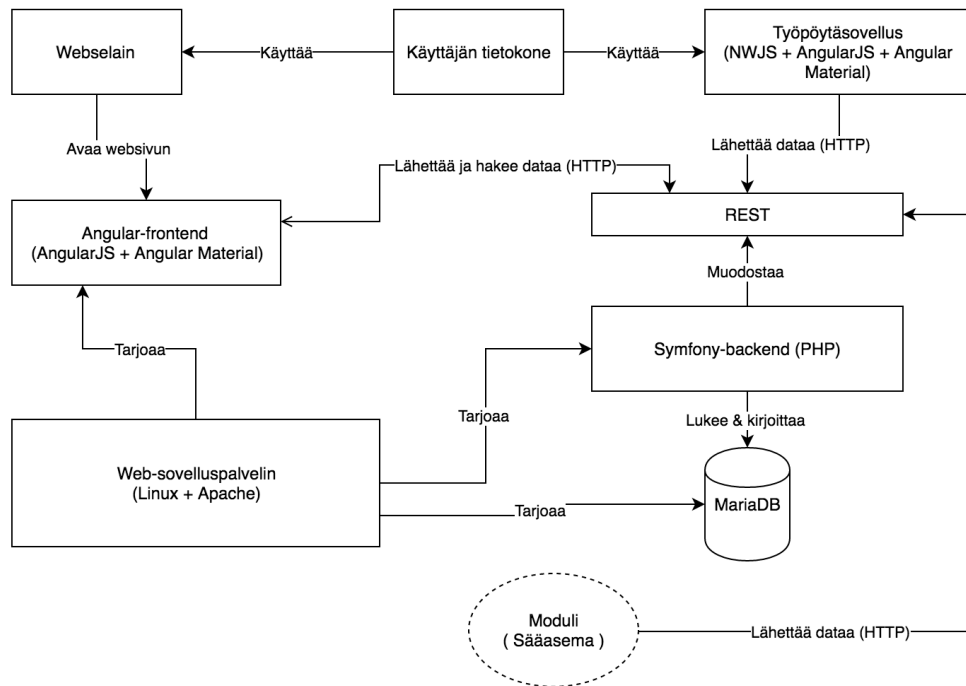
Erilaisia rajapintavaihtoehtoja on tarjolla runsaasti, joskin niiden välillä rajat ovat hyvin häilyviä. Kaikille rajapinnoille on olemassa standardit ja tiukat määritelmät, mutta esimerkiksi REST-rajapintamääritelmää käytetään hyvin vapaamielisissä yhteyksissä. Lyhyesti voitakoon todeta, että rajapinnat toteutetaan yleisimmin käyttäen XML- tai JSON-formaattia niiden datankäsittelymuotona.

Symfony tarjoaa sisäänrakennettuna keinon tehdä määrittämiä, jotka ohjaavat tiettyyn polkuun tulevan HTTP-pyynnön tiettyyn PHP-luokan metodiin. Opinnäytetyössä käytettiin tätä keinoa ja toteutettiin sen avulla REST-tyyppinen JSON-rajapinta, jonka kautta sovellukseen voidaan viestiä myös varsinaisen selainsovelluksen ulkopuolelta. Opinnäytetyössä ei pyritty varsinaisesti toteutettavan REST-rajapinnan täydelliseen REST-määritelmän täyttämiseen, vaan toteutettava rajapinta muodostettiin soveltaen tarpeen mukaiseksi kokonaisuudeksi.

#### 4.8 Teknologiaavalinnat yhteenvetona

Opinnäytetyössä sovellukseen käytetyt teknologiat noudattelevat hyvin pitkälti perinteisenä tunnettua web-sovelluksen teknologiakattausta. LAMP-sovelluskokoelma on varma valinta moneen pieneen kuin suurempaankin ohjelmistoprojektiin. Pitkä historia, suuri ja vakaa käyttäjäkunta höystettynä aktiivisella kehittäjäkunnalla tuovat luotettavuutta ja vakautta ohjelmistokehitykseen.

Teknologiat huomioon ottaen sovelluksen kokonaisarkkitehtuuri muodostui kuvion 5 mukaiseksi. Kuviosta voidaan havaita myös eri sovelluskomponenttien yhteydet keskenään.



Kuvio 5. Sovelluksen kokonaisrakenne avattuna yksittäisten teknologioiden tasolle

## 5 Toteutus

### 5.1 Yleistä

Moniosaisen sovelluksen toteuttaminen yhden hengen tiimillä on tasapainottelua eri osioiden ohjelmoinnin välillä. Toteutettavan järjestelmän moniosaisuus tuo toisaalta haasteita, mutta toisaalta myös selkeyttää ja osittaa työskentelyä hyvällä tavalla. Kunkin sovelluksen vastuualueet on jaoteltava selkeästi, jottei päällekkäistä toiminnallisuutta ohjelmoida. Osaa järjestelmän ominaisuuksista on myös kehitettävä rinnan, jotta toisella sovelluksella voidaan testata, toimiiko toinen sovellus oikein.

Sovelluksien toteutus voidaan karkeasti jakaa seuraaviin vaiheisiin:

1. Suunnittelu – mitä tehdään?
2. Järjestelmien ja kehitysympäristöjen pystytys (versionhallinta, tuotantoympäristö, eri sovelluskehysten alustaminen versionhallintaan)
3. Palvelinsovelluksen alustava toiminta (autentikoituminen, käyttäjien lisäys-, muokaus- ja poisto-operaatiot)
4. Selainsovelluksen alustava toiminta (käyttöliittymän pohjat, peruskonfiguraatiot)
5. Tarvittavien yleiskäyttöisten toimintojen kehittäminen rinnakkain palvelinsovellukseen ja selainsovellukseen
6. Työpöytäsovelluksen alustava toiminta kehitysympäristöineen
7. Palvelinsovelluksen työpöytäsovellusrajapinnan toteuttaminen

8. Globaalien syöttölaitteiden kuuntelijoiden ja muiden статистиikkaa mittaavien ominaisuuksien toteuttaminen työpöytäsovellukseen
9. Tiedon tallentaminen työpöytäsovelluksesta palvelinsovellukseen
10. Erilaiset työpöytäsovelluksen dataa ilmentävien raporttien toteuttaminen rinnan palvelinsovellukseen ja selainsovellukseen

Koska eri vaiheiden esittäminen kronologisessa järjestyksessä ei ole mielekästä eikä selkeyden näkökulmasta mitattuna järkevää, esitellään kukin sovelluskokonaisuus omana alalukunaan sisältäen kyseiselle sovellukselle ominaiset teknologiset erityispiirteet ja tärkeimmät käytetyt ratkaisumallit.

## 5.2 Suunnittelu

### 5.2.1 Yleistä

Sovelluksen vaatimusmäärittely antaa reunimmaiset raamit sille, mitä sovelluksen täytyy kyetä tekemään. Vaatimusmäärittely ei kuitenkaan ota opinnäytetyön tapauksessa kantaa siihen, mitä teknologioita sovelluksen ohjelmoinnissa tulee hyödyntää. Opinnäytetyössä ohjelmistosuunnittelun tavoitteena on avata niin ajatuksen tasolla kuin myös visuaalisesti sitä, miten eri sovelluksen toiminnalliset vaatimukset tullaan toteuttamaan ohjelmoinnillisesta näkökulmasta tarkasteltuna.

### 5.2.2 Dynaaminen tiedon tallentaminen

Suurin tekninen haaste palvelinsovelluksen toteuttamisessa oli mahdollistaa käytännössä vapaa käyttäjän rajapintojen hallintamahdollisuus. Mahdollisuus lisätä ja muokata rajapintoja tarkoittaa käytännön tasolla sitä, että käyttäjä voi määrittää tietokantaan tallennettavaksi tietyn muotoista dataa, jota käyttäjä voi joko automatisoidusti tai manuaalisesti tallentaa tietokantaan. Tästä käyttäjän lisäämästä datasta tulee myöhemmin voida tehokkaasti muodostaa erilaisia raportteja, joissa kaikkea käyttäjän lisäämää dataa voidaan verrata päällekkäin toisiinsa siten, että mahdollisia korrelaatioita eri tunnuslukujen kesken voidaan havaita. Tähän käyttäjän lisäämään dataan on voitava edelleen verrata sovellukseen toteutettujen eri moduulien tallentamaa dataa.

Koska tietokantaratkaisuna käytetään SQL-tietokantaa, on monimutkaistenkin kyselyiden muodostaminen tietokantaan mahdollista. Kyselyt välitetään merkkijonoina tietokantamoottorin pureskeltavaksi, jolloin ne voidaan täysin dynaamisesti muodostaa

palvelinsovelluksen sovelluslogiikassa esimerkiksi tiettyjen hakuehtojen mukaan. Seuraavassa esimerkki yksinkertaisesta SQL-lausekkeesta:

```
SELECT * FROM USER;
```

Edellä mainittu SQL-lauseke hakee kaiken tiedon tietokannan user-taulusta. SQL-lausekkeisiin voidaan kirjoittaa myös esimerkiksi liitoksia toisiin tietokannan tauluihin, ehtoja, joiden mukaan hakutulosta rajataan, ryhmitellään tai järjestellään sekä niissä voidaan käyttää erinäisiä hakutulosta muokkaavia joko tietokannan sisään rakennettuja tai itse ohjelmoituja funktioita.

Tämän huomioon ottaen toteutettiin tietokantarakenne sellaiseksi, että tietokannan tauluja muodostetaan dynaamisesti käyttäjän datan mukaisesti. Tällöin tietokantaan tulee paljon tauluja, mutta niistä hakeminen ja niiden käyttäminen raporttien laadintaan pysyy yksinkertaisena ja nopeana. Kun huolehditaan siitä, että käyttäjien toimien johdosta määritellyt tietokannan taulut nimetään eri tavoin kuin järjestelmän käyttämät taulut, ei konflikteja pitäisi syntyä. Vaihtoehtoinen tapa olisi luoda esimerkiksi kullekin käyttäjälle yksi tietokannan taulu, johon tallennettaisiin tietoa käyttäjän toimien perusteella.

### 5.2.3 Rajapintasuunnittelu

Palvelinsovelluksen pääasiallinen tehtävä on ylläpitää rajapintaa, jonka kautta voidaan ikään kuin pyytää palvelinsovellusta antamaan tai tallentamaan dataa. Tämän tarjotun rajapinnan tulee olla siis ainut kanava, jonka välityksellä millään sovelluksella on yhteys järjestelmään tallennettuun dataan.

Rajapinnan voi suunnitella hyvin monella eri tavalla. Hyvin yleinen tapa REST-tyyppisen rajapinnan toteuttamiseen on käyttää HTTP-pyynnön metodeita (Request Methods) rajapinnan päätepisteen (endpoint) tyyppin kuvaamiseen. Usein puhutaan CRUD-operaatioista (Create, Read, Update, Delete), joita siten vastaavat POST-, GET-, PUT- ja DELETE-metodit. (Pautasso 2009.)

Toisaalta rajapinta voidaan määrittää enemmänkin verbaalisesti kuvailevampaan muotoon, jolloin dataa lisäävä rajapinnan päätepiste voisi olla esimerkiksi muotoa */api/post/create*, joka nimensä mukaisesti voisi esimerkiksi luoda uuden artikkelin mukana välitettävistä parametreista (payload). Tällainen tapa on yleisesti käytetty, mutta

toisaalta ei kovinkaan suositeltu sen sisältämien huonojen puolien vuoksi. Koko rajapinnan tunnelointi GET-metodin kautta rajoittaa tietyiltä osin lähetettävän datan määrää ja sotii täysin sitä ajatusta vastaan, ettei GET-metodin tulisi missään tilanteessa muuttaa palvelinsovelluksen tilaa. Toisaalta tunnelointi POST-metodin kautta tekee välimuistin käytöstä mahdotonta tekemällä mahdottomaksi etukäteen tietää, onko pääte piste mahdollisesti jokin web-resurssi, joka selaimen tulisi mahdollisesti voida tallentaa välimuistiin. (Pautasso 2009.)

Työssä toteutettua rajapintaa lähdettiin vastoin yleistä suositusta toteuttamaan kuvailvasti ja rajapinnan pääte pisteissä käytetään selkeyden vuoksi aina POST-metodia. Tämä yksinkertaistaa kaikkia rajapinnan kutsuoperaatioita niitä suorittavissa lähteissä, koska niiden tarvitsee huolehtia vain yhdenlaisesta rajapintakutsusta – vain pääte piste ja parametrit muuttuvat. Rajapinta toteutetaan luokittelemalla rajapinnan tarvittavat pääte pisteet tiettyjen avainsanojen alle, jolloin eri aihealueisiin liittyvät toiminnot pysyvät organisoituna.

### 5.3 Kehitystyökalut ja käytetyt palvelut

Sovelluskehitykseen käytettävät työkalut määrittyvät yleensä tarve pohjaisesti, mutta suuri osa valinnoista muodostuu myös ohjelmoijan mieltymysten mukaisesti. Ohjelmointityökaluina käytetäänkin yleensä niitä työkaluja, jotka jostain tietystä syystä ovat vakuuttaneet ohjelmoijan suuremmin kuin jokin toinen työkalu.

#### 5.3.1 Sovelluskehitysympäristö

Varsinaiseen ohjelmointiin, eli ohjelmakoodin kirjoittamiseen, riittää käytännössä aivan tavallinen tekstieditori. Kuitenkin alan kehittyessä entistä kiivastahtisemmaksi uusien teknologioiden ja kasvavien kirjastojen myötä on tullut tarve pystyä integroimaan ohjelmointia helpottavia työkaluja sinne, missä niitä oikeasti tarvitaan. Merkittävin ohjelmointia helpottava toiminto lienee automaattinen täydennys, joka tarkoittaa käytännön tasolla sitä, että sovelluskehitysympäristö osaa ehdottaa jo etukäteen, mitä mahdollisesti ohjelmoija haluaa seuraavaksi kirjoittaa. Lisäksi sovelluskehitysympäristöt osaavat yleensä ilmoittaa syntaksisista tai tietyissä tapauksissa myös semanttisista virheistä käyttäjää, jonka myötä ohjelmoijan tekemien virheiden määrää voidaan tältä osin pienentää jo ennakoon.

Opinnäytetyössä käytettiin sovelluskehitysympäristönä JetBrainsin tuottamaa PhpStorm-sovelluskehitysympäristöä, joka on yksinomaan web-sovelluskehitykseen suunnattu työkalu. Ennakoivan tekstinsyötön ja virheiden tarkastelun lisäksi PhpStormiin on toteutettu integraatioita esimerkiksi versionhallintaan ja tietokantojen käsittelyyn.

### 5.3.2 Versionhallinta

Sovelluksen kehittäminen voidaan jakaa osiin, yksittäisiin pieniin muutoksiin. Esimerkiksi jonkin pienenkin lisätoiminnon toteuttaminen voidaan jakaa esimerkiksi sataan pieneen osaseen. Kunkin osan tekemisen jälkeen muutokset tallennetaan *versionhallintaan*, joka mahdollistaa muutosten ja esimerkiksi näistä aiheutuneiden vikojen etsinnässä tulevaisuudessa. Erilaisia versionhallintajärjestelmiä on useita, joista suosituimmat ja tunnetuimmat lienevät SVN (Subversion) ja Git. (Nagele n.d.)

Opinnäytetyössä jokaiselle toteutettavalle sovelluskomponentille oli oma git-repositorionsa. Näin eri sovellukset olivat jo siltä tasolta irrotettu toisistaan, mikä omalta osaltaan auttoi pitämään ne erossa toisistaan. Käytettävät repositoriot hankittiin GitHub-palvelusta, jolloin sovellukset lähdekoodeineen ovat helposti saatavilla ja tarkasteltavissa suoraan web-selaimesta.

### 5.3.3 Testausympäristö

Ohjelmistotestaus on prosessi, joka kannattaa aloittaa heti ohjelmiston alkumetreiltä alkaen. Testaus on mahdollista toteuttaa jälkeenpäin, mutta sen upottaminen varsinaisen ohjelmointityön lomaan tekee siitä jatkuvan prosessin ja auttaa mahdollisten vikojen etsimisessä jo ohjelmointivaiheessa.

Kuten kaikkien muidenkin ohjelmoinnin vaiheiden ja tuotteiden käsittelyyn, on myös testaamiseen olemassa huima määrä eri työkaluja. Viime vuosina näiden työkalujen määrä on vain entisestään noussut kiinnostuksen nousun myötä automaattista ohjelmistotestausta kohtaan. Monesti erilaiset arkkitehtuurilliset ja sovelluskehitykselliset ratkaisut valitaankin jopa pelkästään sen mukaan, miten helppoja ne ovat testata automaattisesti.

Opinnäytetyössä käsiteltävän testauksen pääpaino jakautui web-palvelinsovelluksen yksikkö- ja integraatiotestaukseen sekä web-selainsovelluksen automaatiotestaukseen. Web-palvelinsovelluksen testauksessa käytettiin suosittua PHPUnit-testaustyökalua, joka on sellaisenaan upotettuna jo palvelinohjelmoinnissa käytettävään Symfony-sovelluskehikseen. Web-selainsovelluksen automaatiotestaus toteutettiin sen sijaan käyttämällä robot-sovellustestauskehystä, joka mahdollistaa niin kutsutun avainsanapohjaisen testauksen toteuttamisen. Robot-sovellustestauskehys käyttää web-selaimessa toimimiseen edelleen Selenium2-käyttöliittymätestaustyökalua ja toimii näin ollen eräänlaisena abstraktion tasona yksinkertaistaen testitapausten ohjelmointia.

Web-palvelinsovellukseen ohjelmoidut PHPUnit-testitapaukset määriteltiin ajettavaksi Travis CI -palvelussa. Tämä tarkoittaa sitä, että joka kerta, kun versionhallintaan tehtiin jokin muutos, suoritetaan kaikki ohjelmoidut testit automaattisesti viimeisimmällä ohjelmaversiolla Travis CI -palvelussa. Mikäli jokin testeistä epäonnistuu, lähettää palvelu määrittelyn mukaisesti siitä ilmoituksen, minkä johdosta ohjelmoija tietää rikkoneensa sovelluksesta jotain viimeisimmän muutoksen yhteydessä.

## 5.4 Palvelinsovellus

### 5.4.1 Symfony-sovelluskehiksen käyttöönotto

Symfony-sovelluskehiksen käyttöönotto projektiin on helpointa tehdä ennen mitään muuta. Kehiksen upottaminen valmiiseen projektiin on työlästä ja vaatii paljon selvittelyä. Uuden projektin luominen Symfony-sovelluskehiksellä on kuitenkin helppoa noudattamalla Symfonyn omia ohjeita.

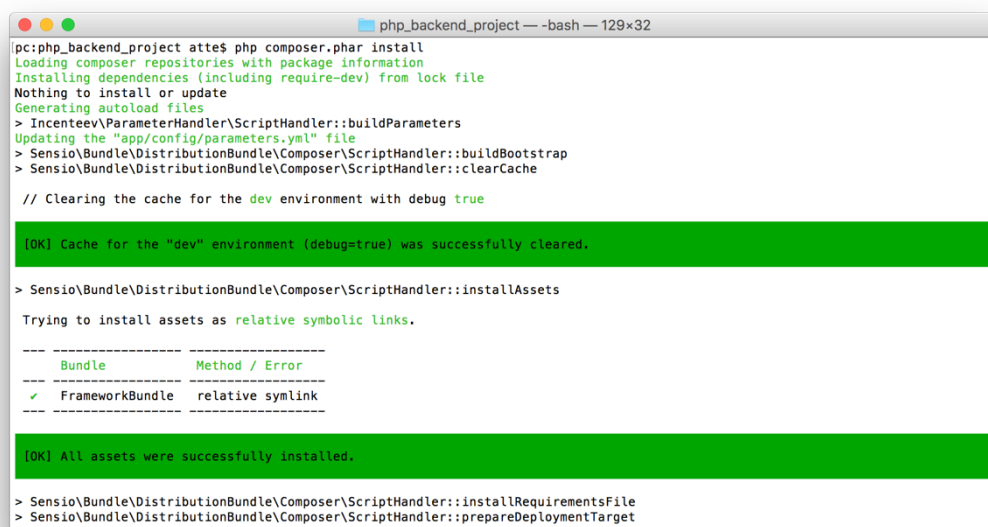
Uuden Symfony-projektin luonti onnistuu käyttämällä Symfony-asennusohjelmaa. Asennusohjelma ladataan, jonka jälkeen se asetetaan ajettavaksi polusta siirtämällä se polkuun määriteltyyn hakemistoon. Tämän jälkeen uusi projekti luodaan Symfonyn *new*-komentoa käyttäen antamalla tälle parametrina uuden projektin nimi. Uuden Symfony-projektin luonnin vaiheet ovat luettavissa kuviossa 6.



Asetuksien määrittelyn jälkeen luotu projekti on valmis ohjelmoitavaksi. Konfiguraatiot voidaan määrittellä erikseen kaikille eri Symfonyn ympäristömuuttujien arvoille, jolloin esimerkiksi PHPUnit voidaan asettaa käyttämään eri konfiguraatiotiedostoa kuin muu sovellus.

### Composer-paketinhallintajärjestelmä

Symfony käyttää paketinhallintajärjestelmänään composer-paketinhallintajärjestelmää, joka PHP-projekteissa hyvin yleisesti käytössä oleva työkalu. Koska Symfony käyttää lisämoduulit, bundlet, ovat poistettu versionhallintaan lisättävistä tiedostoista, on tärkeää muistaa tietyin väliajoin päivittää sovelluspaketit suorittamalla composerin *install*-komento ja päivittää näin käytössä olevat sovelluskomponentit ajan tasalle. Composer voidaan asentaa joko polkuun tai vaihtoehtoisesti voidaan ladata composer.phar-tiedosto, joka suoritetaan parametrisoituna tarvittavilla lisäparametreilla PHP-komentorivisovelluksella sovellusprojektin hakemiston juuressa (ks. kuvio 8).



```

pc:php_backend_project attes$ php composer.phar install
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Nothing to install or update
Generating autoload files
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
Updating the "app/config/parameters.yml" file
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::buildBootstrap
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::clearCache

// Clearing the cache for the dev environment with debug true

[OK] Cache for the "dev" environment (debug=true) was successfully cleared.

> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installAssets
Trying to install assets as relative symbolic links.

-----
Bundle           Method / Error
-----
✓ FrameworkBundle relative symlink

[OK] All assets were successfully installed.

> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::installRequirementsFile
> Sensio\Bundle\DistributionBundle\Composer\ScriptHandler::prepareDeploymentTarget
  
```

Kuvio 8. Composer-paketinhallintajärjestelmän asennustoiminnon tuloste

Kun composer-paketinhallintajärjestelmän asennustoiminto ajetaan projektissa ensimmäistä kertaa, käynnistyy Symfonyn interaktiivinen toiminto, joka kysyy käyttäjältä yksitellen parameters.yml-tiedostossa määritellyt kohdat. Tällöin käyttäjä voi suoraan

syöttää uuden asennuksen käyttämän tietokannan ja eri palveluiden käyttäjätunnukset ja salasanat käsittelemättä mitään tiedostoja erikseen. Tällä myös varmistutaan siitä, että kaikki parametrit tulevat täytettyä ainakin tietyllä tasolla.

#### 5.4.2 Symfony ja tietokanta

Projektiin valittua relaatiotietokantaa ohjaillaan SQL-lausekkeilla (ks. luku 5.2.2), mikä tarkoittaa sitä, että kaikki tietokantaan kohdistuvat muutokset toteutetaan kirjoittamalla SQL-lausekkeitä niitä tulokkaavalle tietokantamoottorille. Suuressa projektissa esimerkiksi tietokannan rakenteen ylläpito ja kehitys voi kuitenkin käydä tällä tavoin melko työlääksi. Onkin toteutettu koko joukko erilaisia tietokannan ja sovelluksen välistä kommunikaatiota helpottavia teknologioita, joiden tarkoituksena on vähentää toistuvaa ylläpito- ja muutostarvetta, mitä tietokannan ylläpitämiseen ja synkronointiin ohjelmakoodin välillä tulee.

#### ORM

Symfony käyttää sisäisesti Doctrine-tietokantakirjastoa, joka sallii niin kutsutun ORM-ohjelmointitekniikan käyttämistä PHP-sovelluskehityksessä. Käytännössä ORM on yksi sovelluksen taso, jonka läpi kulkiessaan data muuttaa muotoaan joko taulukosta olioksi tai oliosta taulukoksi. Tietokannasta dataa haettaessa muutetaan saatu data tiettyjen määrittelyiden mukaan PHP-olioiksi, joita voidaan käyttää siinä missä mitä tahansa muitakin luokkien instansseja. Kun tietokantaan taas halutaan lisätä dataa, voidaan PHP-koodissa luoda uusi instanssi tietystä entiteettiluokasta, joka ORM-kirjastosta riippuen tietyllä tavalla voidaan tallentaa tietokantaan. Taustalla ORM-taso kuitenkin muodostaa määrittelyiden pohjalta SQL-lausekkeitä, jotka se myöhemmin ajaa tietokantaan.

Symfonyyn on toteutettu myös suuri joukko eri komentoriviltä ajettavia toimintoja. Myös useat kolmannen osapuolen ladattavat komponentit tarjoavat toimintansa komentoina, jolloin käyttäjä voi hyödyntää näitä ohjelmoidessaan sovellusta. Esimerkki tällaisesta kehitystä nopeuttavasta komennosta on *doctrine:generate:entity* (ks. kuvio 9), jolla voidaan generoida annetuista määrittelyistä uusi entiteettiluokka ohjelmakoodiin (ks. kuvio 10).

Entiteettiluokassa määritellään kaikki tietokannan taulun ja suhteiden luontiin tarvittavat määrittelyt käyttämällä *annotaatioita* (ks. luku 5.4.6). Määrittelyissä kerrotaan esimerkiksi käytettävä tietokannan taulun nimi sekä kaikkien sarakkeiden vastaavuus suhteessa luokkamuuttujiin. Näiden määritelmien pohjalta ORM osaa muodostaa ja myöhemmässä vaiheessa muutosten kyseessä ollessa päivittää tietokantaa tarvittavilta osin. Luodun entiteetin mukainen tietokantarakenne muodostetaan käyttämällä komentoa *doctrine:schema:update* (ks. kuvio 11), jonka ajaminen toteuttaa varsinaisen tietokanta-ajon. Komennon suorittamisen jälkeen sen muodostama user-taulu ja sen rakenne voidaan havaita tietokannasta käyttämällä SQL:n DESCRIBE [table]-komentoa (ks. kuvio 12).

```

pc:php_backend_project attes php bin/console doctrine:generate:entity

Welcome to the Doctrine2 entity generator.

This command helps you generate Doctrine2 entities.
First, you need to give the entity name you want to generate.
You must use the shortcut notation like AppBundle\Post.

The Entity shortcut name: AppBundle\User

Determine the format to use for the mapping information.
(Configuration format (yaml, xml, php, or annotation) [annotation]):

Instead of starting with a blank entity, you can add some fields now.
Note that the primary key will be added automatically (named id).

Available types: array, simple_array, json_array, object,
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
date, time, decimal, float, binary, blob, guid.

New field name (press <return> to stop adding fields): username
Field type [string]:
Field length [255]:
Is nullable [false]:
Unique [false]: true

New field name (press <return> to stop adding fields):

Entity generation
> created ./src/AppBundle/Entity/User.php
> Generating entity class src/AppBundle/Entity/User.php: OK!
> Generating repository class src/AppBundle/Repository/UserRepository.php: OK!

Everything is OK! Now get to work :).

pc:php_backend_project attes
```

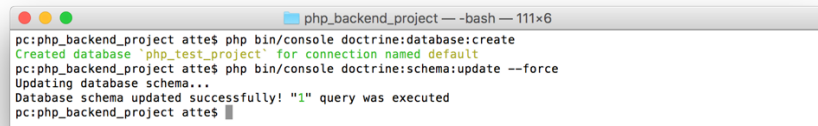
Kuvio 9. Interaktiivinen entiteetin generoiva näkymä

```

1  /**
2   * User
3   *
4   * @ORM\Table(name="user")
5   * @ORM\Entity(repositoryClass="AppBundle\Repository\UserRepository")
6   */
7  class User
8  {
9      /**
10       * @var int
11       *
12       * @ORM\Column(name="id", type="integer")
13       * @ORM\Id
14       * @ORM\GeneratedValue(strategy="AUTO")
15       */
16      private $id;
17
18      /**
19       * @var string
20       *
21       * @ORM\Column(name="username", type="string", length=255, unique=true)
22       */
23      private $username;
24  }

```

Kuvio 10. Symfony:n komennon automaattisesti generoima entiteettiluokka

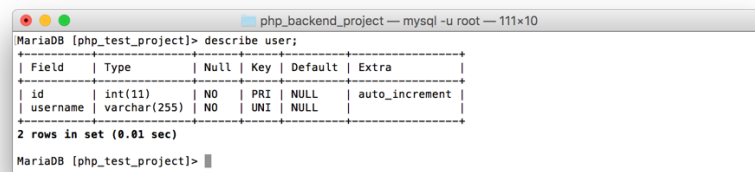


```

pc:php_backend_project attes php bin/console doctrine:database:create
Created database 'php_test_project' for connection named default
pc:php_backend_project attes php bin/console doctrine:schema:update --force
Updating database schema...
Database schema updated successfully! "1" query was executed
pc:php_backend_project attes

```

Kuvio 11. Tietokannan skeeman päivityskomennon ajaminen



```

MariaDB [php_test_project]> describe user;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | NO | PRI | NULL | auto_increment |
| username | varchar(255) | NO | UNI | NULL |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)

MariaDB [php_test_project]>

```

Kuvio 12. Tietokannan user-taulun rakenne komennon suorittamisen jälkeen

### 5.4.3 Tietokannan versionhallinta

Ohjelmakoodin versionhallinnan ohella myös tietokannan versionhallinta on tärkeää. Tietokannan versionhallinnasta käytetään myös usein termiä *tietokantamigraatio*, joka onkin sinällään versionhallintaa kuvaavampi tietokantoja käsiteltäessä.

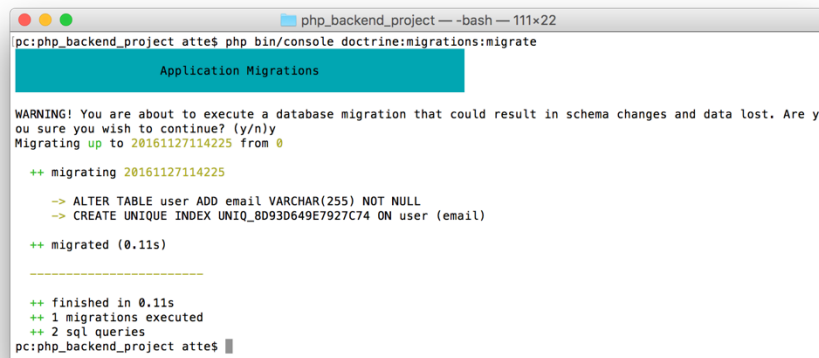
Symfonyyn on saatavilla tietokantamigraatioiden hallintaan lisämoduuli, DoctrineMigrationsBundle, jolla migraatioiden muodostaminen voidaan käytännössä automatisoida yhden komennon, *doctrine:migrations:diff*, suorittamiseen. Kun aiemmin luodulle User-entiteettiluokalle lisätään yksi ominaisuus ja generoidaan migraatiotiedosto, nähdään luodusta tiedostosta, kuinka migraatiot muodostetaan eteen (up)- ja taaksepäin (down) yhteensopiviksi (ks. kuvio 13). Tällöin tietokannan versioissa voidaan joustavasti palata myös taaksepäin. Kuviosta 14 voidaan havaita, kuinka migraatiojärjestelmä ajaa puuttuvat tietokannan migraatiot ja merkitsee ne tämän jälkeen ajetuiksi.

```

8 class Version20161127114225 extends AbstractMigration
9 {
10     /**
11      * @param Schema $schema
12      */
13     public function up(Schema $schema)
14     {
15         $this->addSql('ALTER TABLE user ADD email VARCHAR(255) NOT NULL');
16         $this->addSql('CREATE UNIQUE INDEX UNIQ_8D93D649E7927C74 ON user (email)');
17     }
18
19     /**
20      * @param Schema $schema
21      */
22     public function down(Schema $schema)
23     {
24         $this->addSql('DROP INDEX UNIQ_8D93D649E7927C74 ON user');
25         $this->addSql('ALTER TABLE user DROP email');
26     }
27 }

```

Kuvio 13. Muuttuneesta entiteettiluokasta generoitu migraatiotiedosto



```

pc:php_backend_project attes php bin/console doctrine:migrations:migrate
Application Migrations

WARNING! You are about to execute a database migration that could result in schema changes and data lost. Are you
sure you wish to continue? (y/n)y
Migrating up to 20161127114225 from 0

++ migrating 20161127114225
--> ALTER TABLE user ADD email VARCHAR(255) NOT NULL
--> CREATE UNIQUE INDEX UNIQ_8D93D649E7927C74 ON user (email)

++ migrated (0.11s)

-----

++ finished in 0.11s
++ 1 migrations executed
++ 2 sql queries
pc:php_backend_project attes

```

Kuvio 14. Tietokantamigraatioiden ajaminen tietokantaan

#### 5.4.4 REST-rajapinnan toteutus

Symfony-sovelluksessa REST-rajapinnan toteuttamiseen käytetään Symfony:n vakiona sisältämää route-toiminnallisuutta. Kukin erillinen sovelluksen osio jaetaan omaan kontrolleritiedostoonsa, johon määritellään kyseiselle komponentille kohdistettavat rajapinnan polut. Rajapinnan polut määritellään kontrollerin metodeihin käyttämällä PHP:n annotaatioita, joissa määritellään kyseinen polku mahdollisine lisäparametreineen (ks. kuvio 15). Parametreille voidaan asettaa myös tiettyjä muotovaatimuksia, jolloin käyttäjän syötettä voidaan validoida jo annotaatioissa ennen kuin ohjelma edes siirtyy varsinaiseen metodin toimintalogiikkaan. Annotaatioilla voidaan määrittää myös koko luokalle niin sanottu peruspolku, jonka alle luokan metodeihin määritellyt polut siten asettuvat.

```

17  /**
18   * @Route("api/auth/")
19   * @Method("POST")
20   */
21  class AuthController extends CController
22  {
23      /**
24       * @Permission
25       *
26       * @Route("me")
27       *
28       * @return JsonResponse
29       */
30      public function meAction()
31      {
32          /**
33           * @var EntityManager $em
34           */
35          $em = $this->getDoctrine()->getManager();
36
37          $query = $em->createQueryBuilder()
38              ->from('App:User', 'u')
39              ->select(array('partial u.{id, firstname, lastname, username, email, apiKey}'))
40              ->where('u.id = ' . $this->getUser()['uid'])
41              ->setMaxResults(1)
42              ->getQuery();
43
44          $user = $query->getArrayResult()[0];
45
46          return new JsonResponse($user);
47      }
48  }

```

Kuvio 15. Omien tietojen hakupolku määriteltynä annotaatioilla

## Rajapinnan testattavuus

Symfony toteuttaa sisäisesti HTTP-pyynnön ja varsinaisen rajapinnan muodostavan metodin käsittelemisen välillä eräänlaisen HTTP-pyynnön abstraktion tason. Tämä taso muuttaa PHP:n käyttämät superglobaalit muuttujat luokkapohjaiseksi kerrokseksi, jota käyttämällä voidaan testausvaiheessa virtuaalisesti luoda HTTP-pyyntöjä matkivien luokkien instansseja, välittää niitä parametreina rajapinnan toteuttaville metodeille ja tutkia, mitä metodi tietyissä tilanteissa palauttaa. Täten rajapintaa voidaan testata ilman, että varsinaisesti tehdään aitoja HTTP-pyyntöjä asiakas-palvelinyhteydessä.

### 5.4.5 Käyttäjän autentikointi ja istunnonhallinta

Sovelluksen tietoturvallisuuden kannalta käyttäjän aitouden ja oikeuden tarkastamiseen käytetty *autentikointijärjestelmä* esittää merkittävää roolia. Istunnon hallintaan ja käyttäjän tunnistautumisen toteuttamiseen on olemassa paljon eri vaihtoehtoja, joista jokainen pohjautuu jonkinlaisen tunnuksen välittämiseen asiakassovelluksen ja palvelinsovelluksen välillä. Tunnuksen perusteella myöhemmin päätetään, onko käyttäjällä oikeutta tiettyyn sovelluksen osaan.

Sovelluksen autentikointiin valitaan käytettäväksi JWT (JSON Web Token), joka on käytännössä kolmesta osasta muodostuva base64-enkoodattu merkkijono. JWT kuljettaa

kaiken tarvittavan tiedon itsessään, minkä vuoksi sitä on helppo liikutella eri tyyppisten sovellusten välillä.

Tarkastellaan esimerkkinä seuraavaa JWT-muotoista tunnistetta.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJoaWxsZXZpIiwiaWF0IjoxNDgwMjYwMTU5LCJleHAiOjE1MTE3OTYxNTksImF1ZCI6ImhpbGxldmkueHl6Iiwic3ViIjoiaGlsbGV2aS54eXoiLCJ1c2VybmFtZSI6ImhpbGxldmkifQ.Dcku7hX5XcHVwZjTvb5HfFr0zvvc9bdAvCPm6qQ7wkI
```

Tunnisteen ensimmäisestä pisteellä erotetusta osasta,

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

muodostuu tunnisteen otsaketiedot (header). Otsaketiedoissa välitetään tunnisteen tyyppi ja myöhemmin esitettävän avaimen salaukseen käytetty salausalgoritmi. Varsinainen tunnisteen osa muodostuu, kun tiedot sisältävä JSON-objekti base64-enkoodataan yhdeksi merkkijonoksi.

Tunnisteen toinen osa,

```
eyJpc3MiOiJoaWxsZXZpIiwiaWF0IjoxNDgwMjYwMTU5LCJleHAiOjE1MTE3OTYxNTksImF1ZCI6ImhpbGxldmkueHl6Iiwic3ViIjoiaGlsbGV2aS54eXoiLCJ1c2VybmFtZSI6ImhpbGxldmkifQ
```

sisältää tunnisteen mukana kuljetettavan julkisen tiedon. Tunnisteessa voidaan kuljettaa mukana esimerkiksi tietoja käyttäjästä, jolloin käyttäjän tietoja ei tarvitse välttämättä erikseen hakea. Toisessa osassa voidaan välittää myös esimerkiksi tieto tunnisteen viimeisestä voimassaoloajasta, jolla tunnistetta validoitaessa voidaan päätellä, onko henkilön istunto mahdollisesti jo päättynyt.

Tunnisteen kolmannessa osassa,

```
Dcku7hX5XcHVwZjTvb5HfFr0zvvc9bdAvCPm6qQ7wkI
```

on valitulla salausalgoritmilla salattuna tunnisteen ensimmäisen ja toisen osan enkoodatun muodon, sekä sovellukseen määritellyn salausavaimen yhdistelmä. Kolmatta osaa tarkastelemalla voidaan tunnisteen tiedot validoida web-palvelinsovelluksessa ja päättää, voidaanko tunnisteeseen luottaa vai onko sitä mahdollisesti käsitelty matkan varrella.

Näin menetellessä voidaan tunnisteessa välittää niin julkista tietoa ja toisaalta myös varmistua siitä, että tunnistetta ei ole mahdollisesti muokattu tai yritetty väärentää. Tunnistetta säilytetään asiakassovelluksessa ja aina palvelinsovellukselta jotain pyydettyä tunniste lisätään HTTP-otsakkeisiin, joista se palvelinsovelluksessa luetaan ja tarkastetaan. Jos tunnisteessa ilmenee jotain vikaa tai se on vanhentunut, mitään tietoa ei anneta vaan informoidaan käyttäjää virheellisestä tai mahdollisesti vanhentuneesta tunnisteesta.

#### 5.4.6 Annotaatioiden käyttö PHP-ohjelmoinnissa

Jo aiemmin mainittujen *annotaatioiden* käyttö PHP-ohjelmoinnissa on hiljalleen yleistynyt ohjelmoijien todetessa niiden helpottavan merkittävästi toistuvien operaatioiden suorittamista sovelluksessa. Alkujaan annotaatioiden perimmäinen ajatus oli sitoa metadataa metodin mukaan, jotta esimerkiksi automaattisesti dokumentaatiota generoivat sovellukset saavat metodeista enemmän irti. Annotaatioilla hoidetaan nykyään sovelluksissa kuitenkin suunnattomia määriä erilaisia toimenpiteitä, joilla ei välttämättä ole enää mitään tekemistä metadatan välittämisen kanssa.

#### **Sovelluksen käyttöoikeudet**

Sovelluksessa, jossa käyttäjälle näkyvää ja saatavilla olevaa dataa säädellään käyttöoikeuksilla, on oltava tarkastuksia käyttöoikeuksien suhteen niin palvelinsovelluksessa kuin käyttöliittymässäkin. Opinnäytetyössä palvelinsovelluksen käyttöoikeuksien tarkastamiseen toteutettiin yksinkertainen `@Permission`-annotaatio, jolle voidaan välittää lista tarkastettavista käyttöoikeuksista, jotka tarkastetaan ennen kuin varsinaisen metodin sovelluslogiikkaa päästään suorittamaan (ks. kuvio 16). Käytännön tasolla annotaatio hakee käyttäjän käyttöoikeudet ja tarkastaa, onko käyttäjälle myönnetty oikeutta määritelyihin toimintoihin. Mikäli käyttäjällä ei ole oikeuksia kaikkiin määritelyihin toimintoihin, ilmoitetaan käyttäjälle käyttöoikeuksien puutteesta ja palautetaan viesti HTTP-statuskoodilla 401 (Unauthorized). Mikäli annotaatiolle ei anneta mitään tarkastettavia käyttöoikeuksia, tarkastetaan vain, että käyttäjä on kirjautuneena sisään huomioimalla HTTP-otsakkeissa välitettävä JWT.

```

85  /**
86   * The route for creating new interface.
87   *
88   * @Permission("interface:own:write")
89   *
90   * @Route("create")
91   * @Method("POST")
92   *
93   * @param Request $request
94   *
95   * @return Response
96   */
97  public function addInterfaceAction(Request $request)
98  {

```

Kuvio 16. @Permission-annotaation käyttö metodin käyttöoikeustarkastelussa

#### 5.4.7 Sisäänluetun datan aukilasku

Koska sovellukseen tallennetaan koskematonta dataa useista lähteistä vaihtelevilla tarkkuuksilla, on sovelluksen ripeän toiminnan kannalta oleellista datan laskeminen valmiiksi koostetauluun. Sovelluksessa myöhemmin tehtävät datan hakukyselyt voidaan siten suunnata suoraan tähän tauluun suorittamatta laskentoja enää uudestaan. Tällöin hakuoperaatiot ovat huomattavasti nopeampia verraten tapaan, jossa laskennat toteutetaan aina erikseen tietokantakyselyssä. Tällaisia laskennallisia arvoja ovat esimerkiksi erinäiset näppäinten painalluksista laskettavat tunnusluvut, kuten kirjoitusnopeus ja kopioidun ja kirjoitetun tekstin suhde.

#### Event Scheduler

Käytetty tietokantamoottori tarjoaa Event Schedulerin myötä väylän suorittaa ajastetusti operaatioita tietokannan sisällä. Event Scheduler vastaa peruseriaatteeltaan cron-toimintoa, joka hoitaa säännöllisiä töitä Linux-käyttöjärjestelmässä. Jotta tietokannassa voidaan käyttää Event Scheduleria, täytyy se ensin aktivoida asettamalla globaali muuttuja komennolla `SET GLOBAL EVENT_SCHEDULER = ON`.

Muuttujan asettamisen jälkeen tietokantaan voidaan määritellä toistuvaksi tapahtumia (event) (ks. kuvio 17), jotka tietokanta automaattisesti suorittaa tietyn ajan täytyessä. Tapahtuman määrittely noudattaa samaa linjaa tietokantafunktioiden ja proseduurien määrittelyn kanssa: sillä on alku ja loppu, joiden välillä voidaan suorittaa lähtökohtaisesti kaikkia SQL-lausekkeita muutamia poikkeuksia lukuun ottamatta.

```

3 DROP EVENT IF EXISTS e_CalculateInspectionDataSummaries;
4
5 CREATE EVENT e_CalculateInspectionDataSummaries
6 ON SCHEDULE EVERY 1 MINUTE STARTS NOW()
7 DO BEGIN
8
9     DECLARE i          INT DEFAULT 0;
10    DECLARE n          INT DEFAULT 0;
11    DECLARE u          INT DEFAULT 0;
12    DECLARE startTime DATETIME;
13    DECLARE endTime   DATETIME;
14
15    -- We will call the procedure for the last 5 minutes
16    SET startTime = DATE_SUB(NOW(), INTERVAL 5 MINUTE);
17    SET endTime = NOW();
18
19    SELECT COUNT(id) FROM user INTO n;
20
21    -- Iterate through the users list
22    WHILE i < n DO
23        SELECT id FROM user LIMIT i,1 INTO u;
24
25        CALL sp_CalculateInspectionDataSummaries(u, startTime, endTime);
26
27        SET i = i + 1;
28    END WHILE;
29
30 END;
31

```

Kuvio 17. Tietokannan eventin määrittely SQL-kielillä

## Proseduurit

Opinnäytetyössä datan laskenta toteutettiin tietokantatasolle käyttämällä tallennettuja prosedureja (stored procedure), jotka ovat tietokannan keino säilöä jokin parametrisoitava toimenpide. Proseduurin sisällä voidaan SQL-kieltä käyttäen suorittaa toimenpiteitä tietokannan tasolla nopeasti ja irrallaan varsinaisesta palvelinsovelluslogiikasta. Toteutettu proseduuuri (ks. kuvio 18) käytännön tasolla poistaa vanhat auki lasketut rivit, hakee ja laskee dataa tiettyjen sääntöjen mukaan useista eri tauluista ja kirjoittaa sitten uudet rivit koostetauluun. Opinnäytetyössä toteutetuille proseduurille välitetään sitä kutsuttaessa parametreina henkilön tunniste sekä laskennan alku- ja lopetusaika.

```

1 DROP PROCEDURE IF EXISTS sp_CalculateInspectionDataSummaries;
2
3 CREATE PROCEDURE sp_CalculateInspectionDataSummaries (
4     UserId INT, -- The user's id
5     StartDate DATETIME, -- Start date time of the recalculation
6     EndDate DATETIME -- End date time of the recalculation
7 )
8 BEGIN
9
10    -- Define the interval: how many seconds is the 'duration' of a single row
11    SET @Interval = 10 * 60; -- 10 minutes (600 seconds)
12
13    SELECT COALESCE(StartDate, '1990-01-01'),
14           COALESCE(EndDate, '2036-01-01')
15    INTO StartDate,
16         EndDate;
17
18    DROP TEMPORARY TABLE IF EXISTS tmp_data;
19    CREATE TEMPORARY TABLE tmp_data (
20        UserId INT DEFAULT 0,
21        StartTime DATETIME,
22        EndTime DATETIME,
23        KeysTyped INT DEFAULT 0,
24        TypingSpeed DECIMAL(10,2) DEFAULT 0,
25        KeyCombos INT DEFAULT 0,
26        Pasted BIGINT DEFAULT 0,
27        ActiveTimePercentage DECIMAL(10, 2) DEFAULT 0,
28        MouseTravelDistance BIGINT DEFAULT 0
29    ) ENGINE = Memory;
30
31    /** 1. Delete all existing data between the date range */
32
33    DELETE FROM inspection_data_summary
34    WHERE user_id = UserId AND
35          startTime >= StartDate AND
36          endTime <= EndDate;
37
38    START TRANSACTION;
39
40

```

Kuvio 18. Dataa auki laskevan proseduurin alkusanat

Edellä toteutettuun tietokantaan määritettyyn toistuvaan tehtävään ohjelmoitu loogikka suorittaa edellä kuvatun tietokannan proseduurin kerran minuutissa viimeisen viiden minuutin ajalle kaikille järjestelmään tallennetuille henkilöille. Tämä tarkoittaa sitä, että tietyin väliajoin on hyvä ajaa koko proseduuuri läpi ilman aikarajauksia, jolloin kaikki mahdollisesti pois tippunut data poistetaan myös aiemmin auki lasketuilta riveiltä. Tätä varten toteutettiin erillinen toistuva tapahtuma, joka laskee kaiken datan auki ilman aikarajauksia joka yö. Tapahtuma suoritetaan harvemmin ja suoritusajaksi valitaan yö, koska datamäärien kasvaessa datan laskenta voi kestää todella kauan ja näin ollen hidastaa sovelluksen käyttöä.

#### 5.4.8 Datat haku

Sovelluksen vaatimusmäärittely toteaa, että kaikkea sovellukseen tallennettavaa dataa on kyettävä vertaamaan ajan suhteen toisiinsa. Tämä asettaa datanhauille omanlaisen haasteensa, kun huomioidaan se, miten monipuolisilla aikajaksoilla dataa on kyettävä vertaamaan. Eri tunnusluvut voivat olla keskenään ajallisesti mitattuna hyvinkin kaukana toisistaan – puun kasvunopeutta mitataan kuukausittain tai jopa vuosittain,

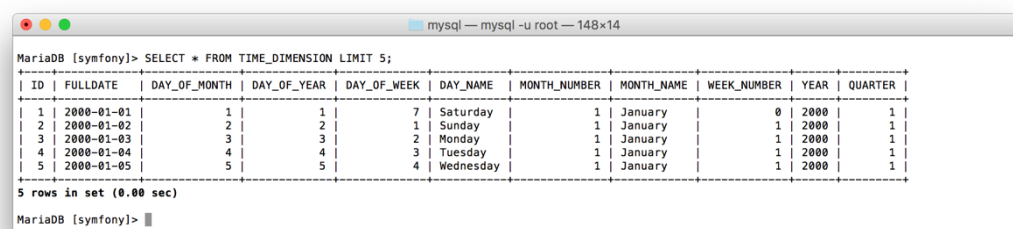
kun taas parhaassa tapauksessa ihmisen virkeystasoa äskettäin syötyyn suklaapatukkaan voidaan haluta tutkia minuuttien tarkkuudella.

Sovellukseen toteutettiin yksi toiminto, jossa käyttäjä voi verrata haluamiensa tunnuslukujen suhdetta toisiinsa ajan mukaan. Käyttäjä voi lisäksi valita, lasketaanko haetavasta tunnusluvusta keskiarvo, summa, merkintöjen määrä, suurin arvo aikavälillä vai pienin arvo aikavälillä. Lisäksi käyttäjä voi valintansa mukaan muodostaa datasta pylväsdiagrammin, piirakkakaavion tai viivakaavion.

### Aikadimensiotaulun käyttäminen datan ryhmittelyssä

SQL sisältää suuren määrän erilaisia vaihtoehtoja datan käsittelyyn ja ryhmittelyyn. Tietokannan sisäistenkin funktioiden käyttö tietyissä kyselyn kohdissa on etenkin suurilla datamäärillä kyselyitä huomattavasti hidastava ja monimutkaistava tekijä. Kun tietoa haetaan ajan suhteen, voidaan avuksi ottaa niin kutsuttu aikadimensiotaulu.

Aikadimensiotaulu on tavallinen tietokannan taulu, johon on etukäteen laskettu auki erinäistä tietoa kullekin vuorokaudelle tietylle aikavälille (ks. kuvio 19). Tekemällä kyselyn tähän tauluun ja liittämällä dataa muista tauluista vuorokauden perusteella voidaan kyselyn ryhmittelyehdossa käyttää esimerkiksi auki laskettua viikko-, kuukausi- tai vuosinumeroa sen sijaan, että ryhmittelyehdossa käytettäisiin jotain tietokannan omaa sisäistä ajanmuotoilufunktiota. Lisäksi aikadimensiotaulua käyttämällä voitaisiin helposti lisätä kyselyyn ehto, joka hakee pelkästään esimerkiksi tiistaipäiville merkityt rivit.



```

MariaDB [symfony]> SELECT * FROM TIME_DIMENSION LIMIT 5;
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | FULLDATE | DAY_OF_MONTH | DAY_OF_YEAR | DAY_OF_WEEK | DAY_NAME | MONTH_NUMBER | MONTH_NAME | WEEK_NUMBER | YEAR | QUARTER |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 2000-01-01 | 1 | 1 | 7 | Saturday | 1 | January | 0 | 2000 | 1 |
| 2 | 2000-01-02 | 2 | 2 | 1 | Sunday | 1 | January | 1 | 2000 | 1 |
| 3 | 2000-01-03 | 3 | 3 | 2 | Monday | 1 | January | 1 | 2000 | 1 |
| 4 | 2000-01-04 | 4 | 4 | 3 | Tuesday | 1 | January | 1 | 2000 | 1 |
| 5 | 2000-01-05 | 5 | 5 | 4 | Wednesday | 1 | January | 1 | 2000 | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

MariaDB [symfony]>

```

Kuvio 19. Aikadimensiotaulun rakenne ja viisi ensimmäistä riviä

Sovellukseen toteutettuihin tietokantaratkaisuihin toteutettiin kuitenkin kullekin taululle omat päivämääräkentät. Tämä voitaisiin myöhemmässä vaiheessa poistaa ja toteuttaa tallentamalla kullekin riville vain liitos-id aikadimensiotauluun. Tällöin aikadimensiotauluun liitoksen tekeminen olisi nopeampaa kuin tällä hetkellä liitosehdon vaatiessa DATE-muotoilufunktion käyttämistä.

Datan haku ajanjaksoittain toteutettiin kuvion 20 esittämällä SQL-lausekkeella. Lauseke muodostetaan dynaamisesti käyttäjän käyttöliittymässä tekemien valintojen perusteella sovellukseen esiohjelmoiduista arvoista siten, ettei käyttäjän syötettä missään vaiheessa liitetä kyselyyn. Kyselyyn haetaan dynaamisesti käytettävä tietokannan taulu sekä haettava arvo. Kun dataa hakeva raportti sitten ajetaan, ajetaan sama kysely kaikille näytettäväksi valituille tunnusluvuille erikseen.

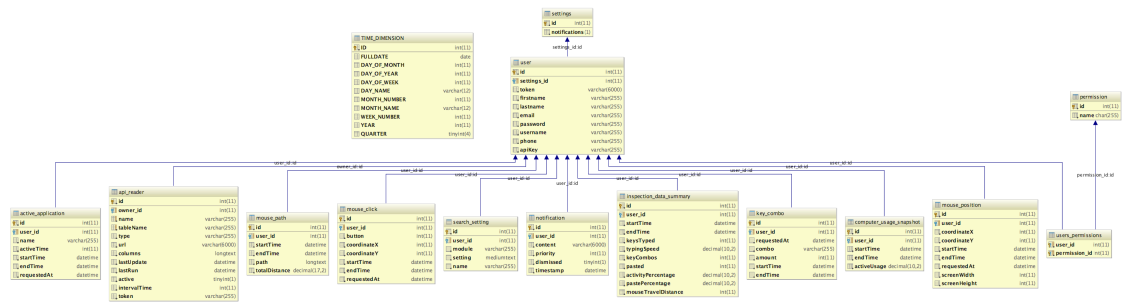
```

1  SELECT
2      DATE_FORMAT(T.FULLDATE, "%m/%Y") AS label,
3      ROUND(AVG(x.temp) * 1, 2) AS temp
4  FROM
5      TIME_DIMENSION T
6  JOIN user_admin_def_HomeWeather x ON DATE(x.REQUESTED_AT) = T.FULLDATE
7  WHERE T.FULLDATE BETWEEN "2016-11-01 00:00:00" AND "2016-11-07 23:59:59"
8  GROUP BY T.MONTH_NUMBER, T.YEAR
9  ORDER BY T.FULLDATE;
```

Kuvio 20. Esimerkki sovelluksen tekemästä datanhaun toteuttavasta SQL-kyselystä

#### 5.4.9 Tietokannan kokonaisrakenne

Sovelluksen käyttämä tietokanta muodostuu yhteensä vain 16:sta taulusta. Kuviosta 21 voidaan havaita, että taulujen kesken suhteita on hyvin vähän, sillä kaikki tieto liitetään sovelluksessa käytännössä vain henkilöön. Suurin osa tauluista onkin vain tietyn tarkasti määritellyn datan keräämistä varten, ja tästä datasta liitos löytyy vain henkilöön. Kuviossa 21 on tietokannan kokonaisrakenne avattu kuvaajaksi, joskin siitä on poistettu näppäinpainalluksia kuvaava taulu, joka on todella suuren kokonsa vuoksi epämieliekäs esittää samassa kuvaajassa kaikkien muiden suhteessa pienempien taulujen kanssa.



### Kuvio 21. Tietokannan kokonaisrakenne

## 5.5 Selainsovellus

### 5.5.1 SPA – yhden sivun sovellus

Web-selainsovellus toteutettiin SPA (Single Page Application) -mallia mukaillen, mikä tarkoittaa käytännössä sitä, että sivu ladataan auki kerran, jonka jälkeen siihen ladataan dataa dynaamisesti sen mukaan, kuinka käyttäjä sivulla liikkuu. Yhden sivun sovelluksella pyritään tuomaan käyttökokemuksellisesti samaa tunnetta, jota voidaan saada työpöytäsovelluksen reaaliaikaisuudesta – siitä, että uuteen toimintoon siirryttäessä koko sovellusta ei tarvitse ikään kuin avata aina uudestaan.

Yhden sivun sovelluksissa on yleisesti käytössä kahta eri toteutustapaa. Kaikki sovelluksen vaatimat tiedostot voidaan koota yhdeksi kokonaisuudeksi ja ladata sovelluksen avautuessa, jolloin ne ovat aina käytössä välittömästi. Tällainen menetelmä takaa välittömyyden tunteen sivustolla liikuttaessa, kun kaikki tarvittava on jo sivun avautuessa latautunut. Toisaalta sovelluksen kasvaessa todella suureksi on tämä toteutustapa hieman liian raskas lähdekoodin määrän kasvaessa epäkäytännöllisen isoksi.

Vaihtoehtoinen tapa kaiken koodin kerralla lataamiselle on ladata vain tarvittava koodi dynaamisesti aina silloin, kun käyttäjä siirtyy sivustolla sellaiseen paikkaan, jonka tarvitsemia koodeja ei olla vielä ladattu. Tällöin käyttäjää odotutetaan sen aikaa, että tarvittavat koodit ja tyylitiedostot ovat ehditty ladata palvelimelta, jonka jälkeen kuitenkin sivua päivittämättä käyttäjä ohjataan uuteen sijaintiin.

Molemmissa edellä mainituissa tavoissa on omat etunsa. Kaikessa yksinkertaisuudessaan ensimmäiseksi kuvailtu tapa on helppokäyttöisempi ja sen toteuttaminen on vähemmän altis virheille. Lisäksi tällä tavoin saatu käyttäjäkokemus tietyissä tilanteissa ohittaa toisen mallin mahdollistaman kokemuksen, koska vaikka dynaamisesti haettavat sisällöt ja koodit olisivat kooltaan hyvinkin pieniä, haku tuo kuitenkin aina pienen viiveen toimintaan. Pyydettyjen resurssien määrän pienentyessä myös palvelinohjelman kuormitus vähenee.

### **Minifiointi ja koostaminen**

Sovelluksen koodin hallittavuuden ja organisoinnin puitteissa ohjelmakoodia pyritään pilkkomaan pienempiin kokonaisuuksiin. Tämä johtaa suureen määrään tiedostoja, joissa kussakin on kuitenkin verrattain vähän itse ohjelmakoodia. Jotta selain ei sivun latauksessa lähettäisi satoja pyyntöjä palvelinohjelmalle ja pyytäisi jokaista JavaScript- ja CSS-tiedostoa erikseen omina resursseinaan, pyritään ohjelmakoodi ja tyylitiedostot ensin minifioimaan ja sitten koostamaan yhteen ladattavaan resurssitiedostoon, joka selainsovelluksen toimesta sitten pyydetään palvelimelta.

Minifionnissa ohjelmakoodista poistetaan kaikki turha metatieto, joka sovelluksen toiminnan kannalta on täysin epäoleellista. Lisäksi ohjelmakoodissa esiintyvien muuttujien nimet supistetaan mahdollisimman lyhyiksi ja kaikki tyhjä tila (whitespace) poistetaan. Minifioinnilla tiedostojen kokoa saadaan pudotettua reilusti, mikä edesauttaa edelleen sivuston nopeampaa latautumista.

### **Grunt.js**

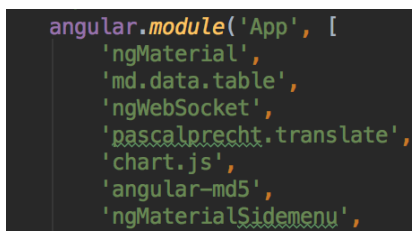
Grunt on eräs JavaScript-automaatiotyökalu, jonka pääasiallinen tehtävä on automatisoida toistuvia ohjelmoijan tekemiä työtehtäviä. Gruntilla ohjelmoija voi määrittää avainsanoja, joita käyttämällä hän voi käskää Gruntia hoitamaan jonkin tietyn tehtävän. Gruntiin on saatavilla paljon kolmannen osapuolen kehittämiä kirjastoja, joissa jonkin aiemmin käsin tehdyn toimenpiteen sisältö kääritään Grunt-tehtäväksi. Opin näytetyössä web-selainsovelluksen kehittämisen aikana Gruntia käytettiin koodin minifiointiin ja koostamiseen. Lisäksi tiettyjä tiedostoja siirretään Gruntin avulla hakemistosta toiseen, jotta ne ovat myöhemmin kyseisestä hakemistosta selainsovelluksen ladattavissa.

### 5.5.2 AngularJS:n käyttöönotto ja konfigurointi

AngularJS:n käyttöönotto web-sivulle tapahtuu yksinkertaisimmillaan lataamalla sivustolle AngularJS:n lähdekoodi. Tämä tapahtuu linkittämällä lähdekoodi HTML-koodin sekaan käyttämällä *script*-tagia ja määrittämällä *src*-attribuutiksi tiedoston sijainnin, jolloin selain osaa pyytää kyseistä tiedostoa määrätystä osoitteesta ja ladata tiedoston sisällön sitten käyttöönsä. Opinnäytetyössä Angularin ytimen lisäksi tarvitaan muitakin Angularin moduuleita, jotka mahdollistavat esimerkiksi polkujen määrittämisen asiakassovelluksen sisällä navigoimiseksi päivittämättä sivua uudelleen.

#### Modulien injektointi

Jotta itse ohjelmoituja ja kolmannen osapuolen AngularJS-moduuleita voidaan käyttää omassa AngularJS-sovelluksessa, on ne kaikki injektoitava sovellukseen. Käytännössä kaikki AngularJS-sovelluksen moduulit on injektoitava sinne missä niitä aiotaan käyttää. Tämä tapahtuu yksinkertaisesti siten, että alustetaan moduuli, jonka jälkeen listataan sille injektoitavat muut moduulit. Kuviosta 22 voidaan huomata, kuinka App-moduulin alustuksessa sille injektoidaan koko joukko kolmannen osapuolen toteuttamia AngularJS-moduuleita.

A screenshot of a code editor showing the configuration of an AngularJS module named 'App'. The code is as follows:

```
angular.module('App', [  
  'ngMaterial',  
  'md.data.table',  
  'ngWebSocket',  
  'pascalprecht.translate',  
  'chart.js',  
  'angular-md5',  
  'ngMaterialSidemenu',  
  // ...  
])
```

Kuvio 22. Moduulien injektointi AngularJS-sovellukseen

#### NgRoute

NgRoute, Angular Route, on Angularin moduuli, joka mahdollistaa navigoinnin lataamatta sivua uudelleen. NgRoutelle voidaan määritellä kullekin polulle erikseen sen käyttämä näkymä (template) ja mahdollinen kontrolleri, sekä muuta harvemmin käytettyä lisätietoa (ks. kuvio 23). Tämän määritelmän pohjalta ngRoute osaa sitten ohjata sovelluksen oikeaan sijaintiin osoiteriviltä paljastuvan osoitteen mukaisesti.

```
// Define routes
$routeProvider
  .when('/', {
    redirectTo: '/dashboard'
  })
  .when('/login', {
    templateUrl: 'web/templates/login/'
  })
  .when('/401', {
    templateUrl: 'web/templates/401.html'
  })
  .when('/settings', {
    templateUrl: 'web/templates/settings/'
  })
  .when('/dashboard', {
    templateUrl: 'web/templates/dashboard/'
  })
  .when('/apis', {
    templateUrl: 'web/templates/api-manager/'
  })
  .when('/apidator', {
```

Kuvio 23. NgRouten polkumäärittelyjä

### 5.5.3 Arkkitehtuuri AngularJS-sovelluksessa

AngularJS-sovellus muodostuu kontrollereista (controller), palveluista (service), direktiiveistä (directive) ja suodattimista (filter) (ks. kuvio 24). Kun tämä kattaus asetetaan rinnan MVC-mallin kanssa, voidaan näistä termeistä löytää parit hyvinkin pienellä vai-  
valla. AngularJS ymmärretäänkin yleensä MVC-arkkitehtuuria toteuttavana sovellus-  
kehiksenä, joskin sitä voidaan käyttää hyvinkin monella eri tavalla.

#### Model

AngularJS-sovelluksessa modelin virkaa toimittavat palvelut eli servicet. Servicet ovat niitä sovelluksen osia, jotka hoitavat kaiken dataan liittyvän – niihin kääritään dataan liittyvät haku, lähetys ja muotoiluoperaatiot, jolloin logiikka ei ole sidoksissa kontrol-  
lereihin ja siten niitä voidaan käyttää yli sovelluksen. Servicet voivat olla hyvin yleis-  
käyttöisiä ja toisaalta ne voivat olla hyvinkin liittymäkohtaisia hoitaen vain täsmälleen  
tiettyä spesifistä tehtävää.

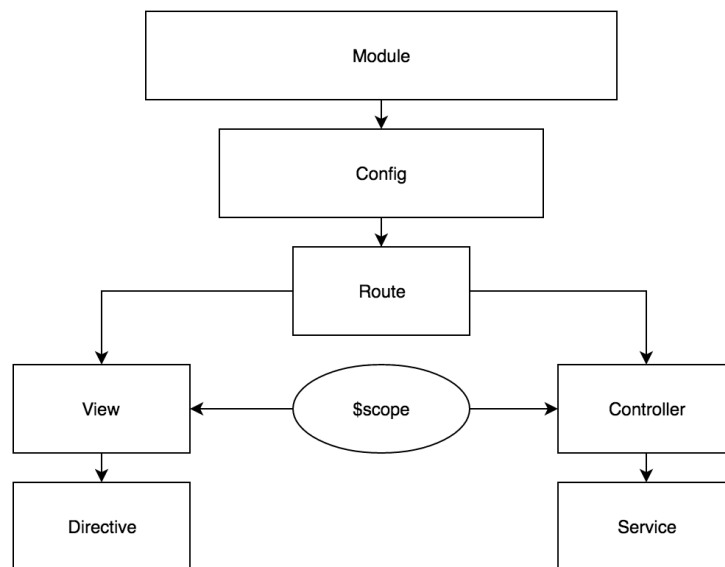
#### View

Viewejä eli näkymiä AngularJS-sovelluksessa vastaavat käytännön tasolla HTML-  
tiedostot. Näistä tiedostoista muodostettavaan DOMiin sidotaan dataa kontrollereista  
käyttämällä AngularJS:n tarjoamia väyliä hyödyntäen esimerkiksi attribuutteja ja An-  
gularJS:n sidontasyntaksia HTML-kielen seassa. Lisäksi näkymiin voidaan lukea myös

direktiivit, jotka ovat eräänlaisia näkymien muodostajia. Direktiiveillä voidaan toteuttaa yleiskäyttöistä toiminnallisuutta tai toteuttaa yleiskäyttöisiä komponentteja käytettäväksi yli sovelluksen.

## Controller

Controller eli kontrolleri on sovelluksen komponentti, joka yhdistää palvelulta haetun tiedon näkymään sekä toisaalta lähettää näkymässä mahdollisesti lisätyn tai muokatun datan palvelulle, joka edelleen lähettää sen palvelinsovellukselle jatkokäsiteltäväksi. Kontrolleriin ei tulisi upottaa liikaa toiminnallisuutta, vaan sen tulisi olla nimenomaan vain välittäjäkerros palvelun ja näkymän välillä.



Kuvio 24. AngularJS-sovelluksen arkkitehtuuri

### 5.5.4 Käyttöliittymän toteuttaminen Angular Material -kirjastolla

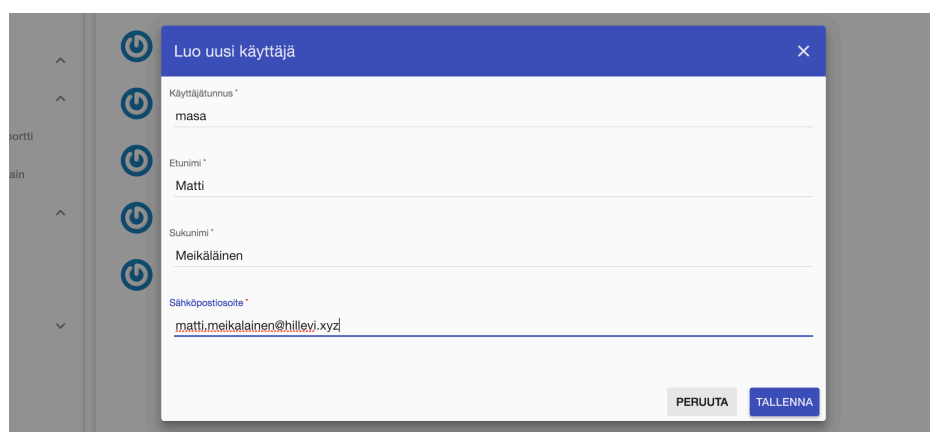
Angular Material on käyttöliittymäkomponenttikirjasto, joka kaikessa yksinkertaisuudessaan on kokoelma Angularin direktiivejä, joihin on ohjelmoitu tavallisesta web-se-laimen toiminnasta eriytettyä toimintalogiikkaa. Angular Material on toteutettu hyödyntäen Material Design -ideologiaa toteuttavaa tyyliä ja onkin ulkoasultaan hyvin pitkälti Googlen sovellusten kaltainen.

Angular Materialin käyttöönotto vaatii varsinaisen AngularJS-moduulin sekä Angular Animaten että Angular Arian injektioonin sovellukseen. Näiden lisäksi myös Angular

Materialin tyylitiedosto täytyy sisällyttää tyylimäärittelyjen toimimiseksi. Tämän jälkeen kirjasto on käyttövalmis, eikä muuta konfigurointia tarvita.

Työskentely Angular Materialin kanssa muodostuu hyvin pitkälti sen tarjoamien elementtien tai attribuuttien käytöstä. Tietyissä tapauksissa elementeille täytyy antaa myös tiettyjä luokannimiä, jotta ne käyttäytyvät halutulla tavalla. Angular Material tarjoaa käytännössä kaiken, mitä yksinkertainen web-selainsovellus kaipaa – yksinkertaisia dialogeja, nappuloita, syöttökenttiä, valintalistoja, luetteloita, ilmoituksia, latausindikaattoreita, ryhmittely- ja asetteluvaihtoehtoja sekä suuren määrän erilaisia säätö- ja konfigurointimahdollisuuksia sen varalle, että käyttäjä haluaa muokata Angular Material -sovelluksensa ulkoasua vakiotyylistä poikkeavaksi (ks. kuvio 25).

Lähestulkoon kaikki Angular Materialin käyttöliittymäkomponentit ovat jollain tapaa animoituja, mikä tuo modernin vivahteen ja pysähtymättömyyden tunteen sovelluksen toimintaan.



Kuvio 25. Angular Materialin avulla web-selainsovellukseen toteutettu lomakenäkymä

Angular Materialissa on kuitenkin joitain puutteita – esimerkiksi minkäänlaista taulukkonäkymää ei ole saatavilla käyttämättä kolmannen osapuolen toteutuksia. Kolmannen osapuolen tuottamia lisämoduuleita on saatavilla runsaasti ja näistä sopivan löytäminen ei ole kovinkaan haasteellista. Lisäksi omat räätälöityjä komponentteja vaativat tarpeet ovat helppo tyydyttää ohjelmoimalla komponentteja itse. Käyttämällä omissa komponenteissa Angular Materialin tyylejä, sulautuvat ne hyvin joukkoon muiden kanssa.

Suurin etu Angular Materialin käytössä on se, että kun oleelliset pointit ensin oppii, on käyttöliittymän kehittäminen äärimmäisen vaivatonta ja mutkatonta sekä kaiken lisäksi äärimmäisen nopeaa. Käyttöliittymästä ei tule suinkaan yhtä persoonallinen verraten tapaan, jossa kaikki tyylit määritellään alusta saakka itse. Myöhemmässä vaiheessa Angular Materialin tyylejä voidaan kuitenkin yliajaa ja tehdä käyttöliittymästä enemmän ohjelmoijan näköinen.

#### 5.5.5 Käyttäjän istunnon ja käyttöoikeuksien hallinta

Perinteinen istunnonhallinta toimii sillä perusperiaatteella, että kullakin sivun latauksella käyttäjän istunnon voimassaolo tarkastetaan ja tarvittaessa uudelleenohjataan käyttäjä esimerkiksi sisäänkirjautumissivulle istunnon puutteen tai vanhentumisen myötä. Koska ohjelmoitu sovellus on kuitenkin yksisivuinen, eikä varsinaisia sivunlatauksia käytännössä tapahdu, täytyi istunnon hallinta hoitaa toisella tapaa.

Käyttäjän istunnon voimassaolon tarkastamiseksi voitaisiin lähtökohtaisesti luottaa esimerkiksi tunnistautumisen yhteydessä palvelimelta noudettuun JWT-tunnisteeseen. Tunnisteesta voidaan sen koodaus purkamalla saada selville sen viimeinen voimassaoloaika, jonka ylittyttyä sovellus mahdollisesti ohjaisi käyttäjän sisäänkirjautumissivulle.

Käyttäjän istunto voi kuitenkin tietyissä tilanteissa päättyä jo ennen tunnisteesta havaittavaa päättymisaikaa. Tällöin sovelluksen tulee osata ilmoittaa istunnon puute käyttäjälle uudelleenohjaamalla käyttäjä sisäänkirjautumissivulle. Kyseinen tilanne kierretään ohjelmoimalla sovellukseen jokaisella sijainnin muutoksella toteutettavaksi niin kutsuttu me-kysely, jonka paluuarvolla sovelluksessa voidaan päätellä, onko sisään kirjautuneen käyttäjän istunto vielä voimassa (ks. kuvio 26).

```

30  /**
31   * Define the location change callback.
32   *
33   * @param {*} $rootScope
34   * @param {*} DataService
35   * @param {*} api
36   */
37  function run ($rootScope, DataService, api) {
38    api.route('auth/me').then(onData, onError);
39
40    // Check the local storage for jwt every
41    $rootScope.$on('$locationChangeStart', function () {
42      api.route('auth/me').then(onData);
43    });
44
45    function onData (data) {
46      DataService.storage.set('user', data.data);
47      $rootScope.$emit('userUpdate', data.data);
48    }
49
50    function onError (err) {
51      DataService.storage.set('user', null);
52      $rootScope.$emit('userUpdate', null);
53    }
54  }
55  }();

```

Kuvio 26. Me-kyselyn suorittava määrittely AngularJS-sovelluksessa

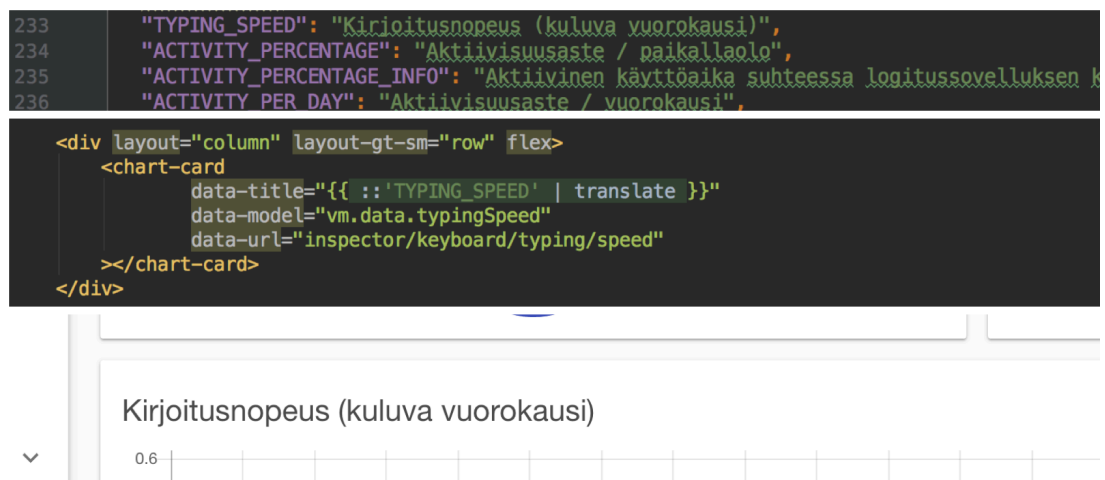
Lisäksi kaikilla niillä datanhauilla, joiden vastauksen HTTP-statuskoodina on 401 (Unauthorized), ohjataan käyttäjä sisäänkirjautumissivulle. Tämä on tilanne, jonka ei pitäisi olla toistettavissa tavallisella sovelluksen käyttämisellä ollen kuitenkin mahdollinen skenaario siinä tapauksessa, että joku yrittää esimerkiksi web-selaimen kehitystyökalujen kautta syöttökenttien arvoja muokkaamalla päästä käsiksi häneltä kiellettyyn dataan.

### 5.5.6 Kieliversiot

Sovelluksen kieliversioiden hallinta on myös eräs niistä sovelluksen osa-alueista, joiden toteuttaminen on järkevintä ottaa osaksi sovelluskehitysprosessia heti projektin alusta alkaen. Käännösten lisääminen ohjelmakoodin sekaan on jälkikäteen tuskaista ja aikaa vievää, kun taas toimivan kieliversiojärjestelmän käyttö on täsmälleen yhtä nopeaa kuin sovelluksen tekstien ohjelmoiminen kovakoodatusti HTML-tiedostoihin.

AngularJS-sovelluksissa kieliversioiden hallintaan suunnattujen kirjastojen markkinoita käytännössä monopolin lailla hallitsee Angular Translate -kieliversiomoduuli, joka mahdollistaa eri kielten käyttämisen halki AngularJS-sovelluksen. Kielet määritellään esimerkiksi JSON-tiedostoihin, joista ne valinnan mukaan ladataan sivulle. Sivulla tietyssä sijainnissa esiintyvä teksti voidaan sen sijaan määritellä useallakin eri tavalla Angular Translaten mahdollistaessa käännöksen noutamisen esimerkiksi filtterin tai

servicen avulla. Esimerkki Angular Translaten käytöstä AngularJS-suodattimena voidaan havaita kuvista 27.



Kuvio 27. Angular Translate -moduulin käyttö käännojen hallintaan

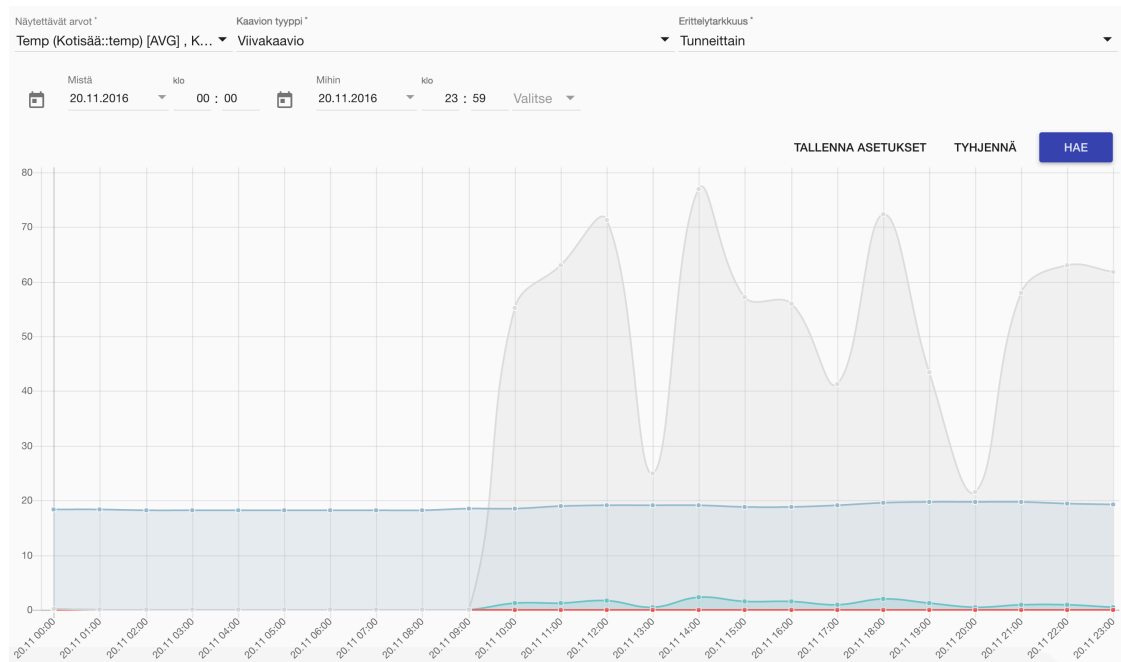
### 5.5.7 Datan haku ja haetun datan visualisointi

Datan haun toteuttavia sovelluksen liittymiä on useita, joista jokaisella on oma erityinen tarkoituksensa. Yleiskäyttöisellä ajanjaksoittain dataa vertailevalla liittymällä käyttäjän tulee voida verrata useita eri datalähteitä toisiinsa. Sen sijaan tietokonetta analysoivan datan tulkitsemiseen erikoistuneet liittymät analysoivat yhtä kohdetta kerrallaan – hiiren analysoinnin toteuttava liittymä muodostuu siis täysin eri pohjalle kuin esimerkiksi näppäimistön käyttöä analysoiva toiminto.

Opinnäytetyössä sovellukseen varastoidun datan visualisointiin käytettiin erilaisten kaavioiden muodostamiseen suunnattua AngularJS-moduulia. Moduuli on toisen, varsinaisen kaavion piirtologiikan toteuttavan, Chart.js-kirjaston päälle toteutettu AngularJS-abstraktio. Tämä helpottaa varsinaisen kaaviokirjaston käyttöä toteutettaessa AngularJS-sovellusta mahdollistaen muun muassa datan sidonnan suoraan kaavion muodostavaan AngularJS-direktiiviin.

Sovellukseen tallennetun datan vertailuliittymä mahdollistaa käytettäväksi kolme eri kaaviotyyppiä: viivakaavio, piirakkakaavio ja pylväskaavio. Käyttäjä voi halutessaan vaihtaa kaavion tyyppiä kesken kaiken tulkitakseen ja vertaillakseen hakemaansa dataa käyttämällä mahdollisuuksien mukaan eri kaaviotyyppisiä. Kaavioon haetaan kaikki

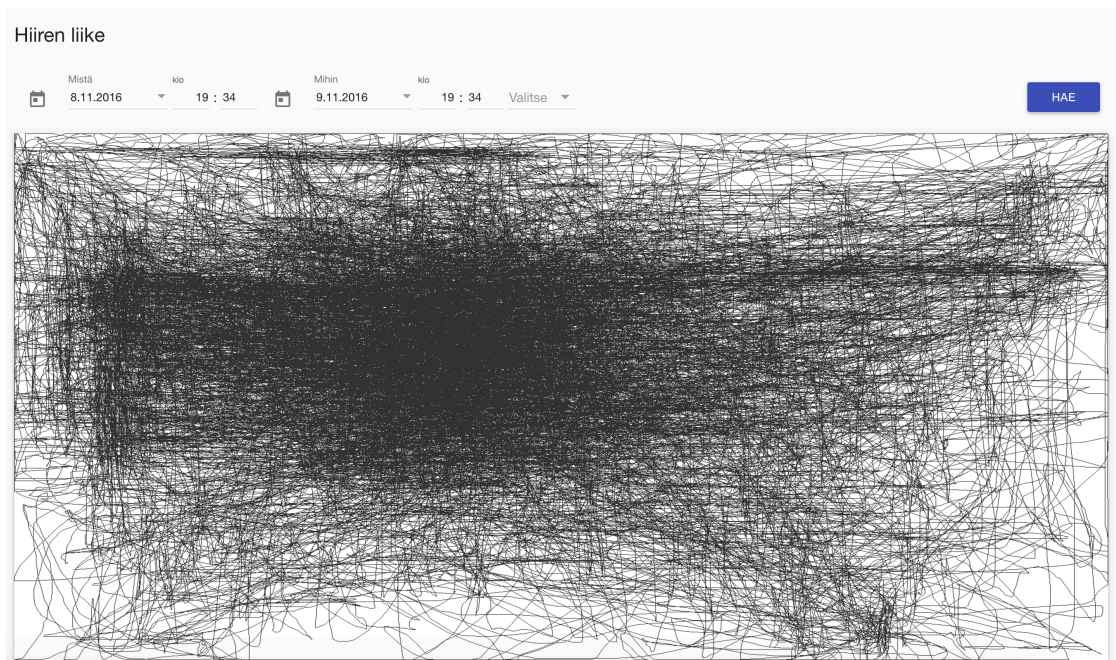
käyttäjän hakuehdot täyttävät datasarjat, joiden tiedot näytetään sitten kuvaajalla päällekkäin mahdollistaen eri sarjojen vertailun keskenään (ks. kuvio 28).



Kuvio 28. Päällekkäin verrattuja datasarjoja esitettynä viivakaaviossa

Käyttäjän tietokoneelta kerätyn datan analysointiin perinteinen kuvaaja on kuitenkin käytännössä hyödytön pois lukien esimerkiksi yksittäiset kerättävät numeeriset arvot. Erinäisiä numeerisesti mittaamattomissa olevia arvoja kuvatessaan sovellus käyttää täysin räätälöityjä kuvaajia. Kuvaajien idea on myös osaltaan kertoa kehitysvaiheesta siitä, että dataa tallennetaan sovellukseen alimmalla tasolla oikein, toisin sanoen data tallennetaan mahdollisimman tarkasti ja kuvaajat ilmaisevat tämän toimintamallin toimivuuden. Tarkasti tallennetusta datasta voidaan myöhemmin eri tavoin laskemalla toteuttaa hyvinkin paljon monimutkaisempia raportteja, kuvioita ja kaavioita.

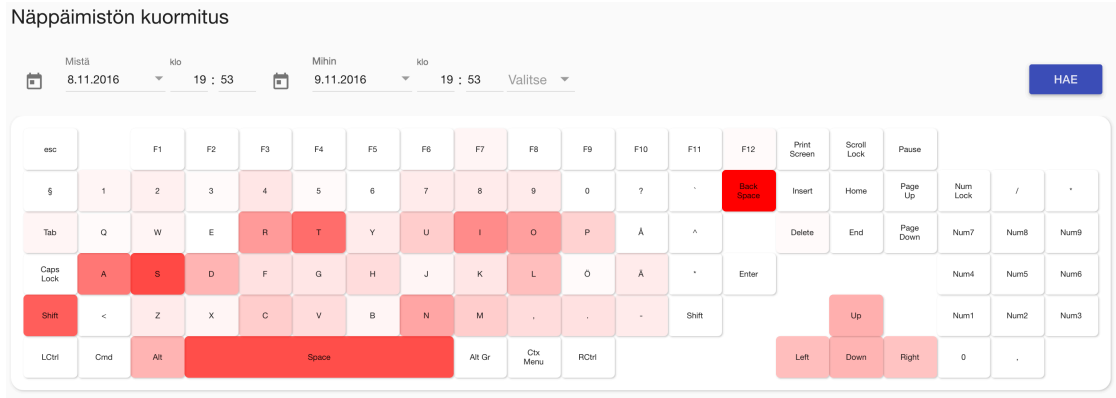
Käyttäjän hiirestä kerätään dataa käytännössä tarkimmalla mahdollisella tavalla kuitenkaan tallentamatta varsinaista ajanhetkeä kullakin tallennetulla koordinaatilla. Jokainen hiiren käytöstä rekisteröity koordinaatti tallennetaan tietokantaan ja tästä kerätystä datasta voidaan piirtää hiiren kokonaisliikkeen kuvaava jatkuva viiva. Kun hiiren liikettä tarkastellaan tällä tavoin pidemmältä aikaväliltä, voidaan selkeästi rajata alue, jossa hiiren kursori on liikkunut eniten (ks. kuvio 29).



Kuvio 29. Hiiren liikettä tietyllä aikavälillä kuvaava raportti

Kuvaaja piirretään datan haun yhteydessä web-palvelinsovelluksessa, josta se lähetetään web-selainsovellukseen piirrettäväksi ruudulle. Kuvaaja muodostetaan käyttämällä PHP:n GD-kuvanmanipulointikirjastoa luomalla tausta ja tämän jälkeen yhdistämällä kaikki tietylle aikavälille haetut hiiren liikkeestä tallennetut koordinaatit tummalla ohuella viivalla. Web-selaimeen kuva ilmestyy todellisena JPG-muotoisena kuvana, jonka käyttäjä voi halutessaan tallentaa päätelaitteelleen käyttämällä web-selaimen sisäänrakennettua tallennustoiminnallisuutta.

Käyttäjän näppäimistön kuuntelusta kerättävä data esitetään käyttöliittymässä käyttäen näppäimistöä esittävää käyttöliittymäkomponenttia. Komponenttiin haetaan näppäimenpainallusten lukumäärä ryhmiteltynä näppäimittäin tietylle aikajaksolle ja sidotaan näistä suhteutettuna tietty punaisen väri kuhunkin komponentin näppäintä kuvastavaan elementtiin (ks. kuvio 30). Tällä tavoin käyttäjälle voidaan informatiivisesti esittää graafisesti näppäimistön eniten käytetyt kohdat sekä edelleen näyttää myös tarkasti näppäimenpainallusten lukumäärä kullekin näppäimelle valitulla aikavälillä.



Kuvio 30. Näppäimistön kuormituskuvaaja web-selainsovelluksessa

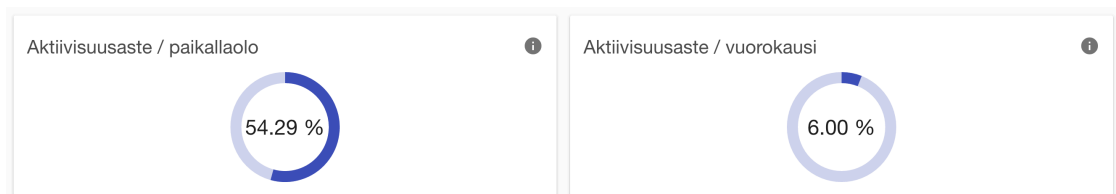
Kuormituskuvaaja on toteutukseltaan hyvin yksinkertainen: JavaScript-objektiin noudetaan tieto näppäimenpainallusten lukumäärästä. Tämän jälkeen käyttäen AngularJS:n ngRepeat-direktiiviä iteroidaan kukin objektin elementti käyttöliittymään siten kuhunkin elementtiin sitä vastaavassa JavaScript-objektissa määritellyn arvon mukaisesti web-palvelinsovellukselta haetusta datasta tietty arvo (ks. kuvio 31). Väriytyy kuhunkin komponentin näppäinelementtiin saadaan vertaamalla kyseiseen näppäimeen kohdistuneiden painallusten määrää kovimmalle rasitukselle joutuneen näppäimen kokonaispainallusmäärään ja muuttamalla tämän jälkeen näppäimen punaisen taustavärin läpinäkyvyyttä.

```
{ key: 'keyF1', cols: 1, text: 'F1' },
{ key: 'keyF2', cols: 1, text: 'F2' },
{ key: 'keyF3', cols: 1, text: 'F3' },
{ key: 'keyF4', cols: 1, text: 'F4' },
{ key: 'keyF5', cols: 1, text: 'F5' },
{ key: 'keyF6', cols: 1, text: 'F6' },
{ key: 'keyF7', cols: 1, text: 'F7' },
{ key: 'keyF8', cols: 1, text: 'F8' },
{ key: 'keyF9', cols: 1, text: 'F9' },
{ key: 'keyF10', cols: 1, text: 'F10' },

<div class="kboard" layout="column" flex>
  <div layout="row" flex data-ng-repeat="row in vm.layout">
    <div data-ng-class="{ kbutton: button.text !== '', kspace: button.text === ''}"
      flex="{ ::button.cols * 5 }"
      data-load-bgc="vm.data[button.key].mapped"
      data-ng-repeat="button in row">
      <span>{{ ::button.text }}</span>
      <md-tooltip>{{ vm.data[button.key].original }}</md-tooltip>
    </div>
  </div>
</div>
```

Kuvio 31. Näppäimistökomponentin toteuttava ohjelmakoodi

Edellä kuvailtujen räätälöityjen sekä perinteisten kuvaajien lisäksi sovellus esittää dataa numeerisena erinäisissä yhteenvedonäkymissä. Yksittäinen numero ei sinällään kerro paljon, mutta kun se sidotaan esimerkiksi aikaan, ja muihin yksittäisiin numeroihin samassa kontekstissa, voidaan siitä saada paljon esiin. Numeroarvon visualisointiin sovellus käyttää Angular Materialin tarjoaman latausindikaattorin avulla toteutettua tietyn asian täydellisyyttä tai osuutta kuvastavaa komponenttia (ks. kuvio 32). Yksinkertaisten suhdelukujen esittämiseen tällainen komponentti on omiaan tuoden käyttöliittymään samalla ilmapuutusta ja kontrastia.



Kuvio 32. Angular Materialin latausindikaattorista räätälöityjä komponentteja

## 5.6 Työpöytäsovellus

### 5.6.1 NWJS-sovelluskehityksen käyttöönotto

NWJS perustuu lyhyesti ilmaistuna ajatukseen, jossa työpöytäsovellus toteutetaan käyttäen perinteisesti web-sovelluksiin rajoittuviksi käsitetyillä teknologioilla. Tämä tarkoittaa käytännön tasolla sitä, että sovelluksen arkkitehtuuri vastaa pääosin täsmälleen sitä, mitä perinteisenkin web-sovelluksen. Poikkeuksen tekee asiakas-palvelin-yhteyden erilainen luonne – palvelintietokoneena toimiikin käyttäjän oma tietokone. Lisäksi NWJS:n asiakas-palvelin -yhteys on käytännössä olematon, jolloin esimerkiksi käyttöliittymän muodostavan HTML-kielen sekaan voidaan suoraan upottaa niin sanottua palvelintietokoneella ajettavaa JavaScript-ohjelmakoodia.

Ohjelmointityö käyttäen NWJS:ää sovelluskehityksenä ei juurikaan poikkea web-selainsovelluksen ja Node.js-palvelinsovelluksen ohjelmoinnista – sovelluksen ohjelmointiin voidaan käyttää niin Chromium-selainympäristön tarjoamaa ohjelmointirajapintaa kuin myös Node.js APIa. Näiden lisäksi NWJS tarjoaa vielä oman ohjelmointirajapinnan, joiden avulla selaimen voidaan toteuttaa perinteisesti työpöytäsovelluksiin kuu-

luviksi ymmärrettyjä ratkaisuja ja toimintoja esimerkiksi sovelluksen valikoiden määrittelyn osalta. Lisäksi sovelluksen ikkunan kokoa, näkyvyyttä ja muita ominaisuuksia voidaan määritellä suoraan JavaScript-sovelluksen lähdekoodissa.

### 5.6.2 Globaalit syöttölaitteiden tapahtumankuuntelijat

Etukäteen ajateltuna sovelluskehityksen suurimpana haasteena mieltä vaivasivat työpöytäsovellukseen vaadittavat globaalit syöttölaitteiden kuuntelijat. Globaalien kuuntelijoiden toteuttaminen on sinällään jo hieman epäilyttävää, koska niin kutsutuilla näppäilyn tallentajilla (keylogger) on jo lähtökohtaisesti huono maine niiden ollessa hyvin käytettyjä esimerkiksi salasanojen ja käyttäjätunnusten urkintaan ihmisten tietokoneilta. Kuitenkin varsinaisen tavoitteen täyttämiseksi vaihtoehtoisia keinoja ei käytännössä ole olemassa.

Koska opinnäytetyössä työpöytäsovellus toteutettiin käyttämällä sinällään monialustaista NWJS-sovelluskehystä, voitiin mahdollisen natiivin toteutustavan puutoksen korvata käyttämällä käyttöjärjestelmäkohtaista työkalua, jonka käyttämisestä toteutetaan uusi Node.js-moduuli. Moduuli on täten käytettävissä käyttöjärjestelmästä riippumatta, kun moduuli sisäisesti Node.js:n API:n avulla selvittää käytettävän käyttöjärjestelmän.

Opinnäytetyön toteuttamisessa pyrittiin hyödyntämään mahdollisimman paljon valmista toteutusta. Tämän vuoksi ratkaisumallien etsiminen alkoi valmiiden kirjastojen tutkimisesta ja testaamisesta. Selvityksen ja testaamisen jälkeen oli selkeästi huomattavissa, ettei mahdollisia valmiita natiiveja monialustaisia ratkaisuja ole juurikaan olemassa. Node.js:lle löytyi kuitenkin muutamakin eri moduuli, jotka käyttävät hyväkseen muita ympäristöjä ja käyttöjärjestelmäriippuvaisia komentoja eri toimintojen toteuttamiseen.

#### **GKM**

GKM (Global Keyboard and Mouse) on Node.js:n moduuli, joka mahdollistaa globaalin näppäimistön ja hiiren kuuntelun Node.js-ympäristössä. GKM pohjaa toimintansa Javalla toteutettuun JNativeHook-kirjastoon, minkä vuoksi Java on oltava asennettuna tietokoneelle (gkm n.d). JNativeHook mahdollistaa hyvin monipuolisen näppäimistön

ja hiiren kuuntelun pohjautuen matalamman tason käyttöjärjestelmäkohtaisiin kuuntelijoihin, joiden käyttö GKM:ssä abstraktistuu käytännössä yhdeksi JavaScript-funktioksi (About n.d). GKM:n heikkoutena voidaan pitää sen vaatimaa toimivaa Java-asennusta käyttäjän päätelaitteelta, mutta tilanteet, joissa käyttäjän päätelaitteelta puuttuu Java, lienevät melko harvassa Javan näytellessä hyvin merkittävää roolia useissa työpöytäsovelluksissa.

GKM:n ohella muita yhtä helppokäyttöisiä ja varmatoimisia ratkaisuja ei selvityksen aikaan löytynyt, joten opinnäytetyössä päädyttiin käyttämään globaalin syöttölaitteen kuuntelijan toteuttamiseen GKM-moduulia. Moduulin käyttö on itsessään yksinkertaista: kullekin halutulle tunnusluvulle kirjoitetaan sovelluksen käynnistytksen yhteydessä oma kuuntelija, jolle määritelty funktio ajetaan aina JNativeHookin laukaistessa kyseisen tapahtuman (ks. kuvio 33). Tapahtumaan välitetään parametrina tietoa tapahtumasta, joka voidaan sitten käsitellä halutulla tavalla sovelluslogiikassa.

```
// Global listener for the mouse press event
gkm.events.on('mouse.pressed', function (e) {
    InspectorDataService.registerMouseClicked(e[0]);
    resetIdleListener();
});
```

Kuvio 33. Globaali hiiren kytkimen painalluksen nappaava kuuntelija

Kuuntelijoita toteutettaessa sovelluksen sisään muodostettiin oma taso, joka yhtenäistää datan käsittelyn sovelluksen sisällä. Tämä mahdollistaa myöhemmässä vaiheessa esimerkiksi pohjalla käytetyn näppäimistöä ja hiirtä kuuntelevan sovellutuksen helpon vaihtamisen toiseen, mikäli jokin luotettavampi ja entistäkin alustariippumattomampi ratkaisu jossain vaiheessa onnistutaan toteuttamaan.

### 5.6.3 Aktiivisen sovelluksen selvittäminen alustariippumattomasti

Sovelluksen vaatimusmäärittelyssä työpöytäsovelluksen toiminnallisiin vaatimuksiin syöttölaitteiden kuuntelijoiden lisäksi nostettiin myös käyttäjän aktiivisten sovellusten käyttöajan mittaus. Tämän mittausdatan pohjalta voitaisiin myöhemmin toteuttaa raporteja, joista voidaan saada selville esimerkiksi tiettyjen sovellusten käyttöastetta verraten tätä esimerkiksi vuorokaudenaikaan.

Aktiivisten sovellusten määrittämisessä pätevät samat pelisäännöt kuin syöttölaitteidenkin globaaleissa kuuntelijoissa. Suurin osa Node.js:lle toteutetuista moduuleista ovat edelleen suunnattu nimenomaan tietylle käyttöjärjestelmälle, jolloin niiden käyttäminen järjestelmäriippumattomaksi tarkoitettussa kontekstissa on käytännössä mahdotonta. Viimeisenä vaihtoehtona on tietysti etsiä kaikille alustoille toimivat ratkaisut ja kehittää näitä käyttämällä oma moduuli, joka käytetyn käyttöjärjestelmän mukaan valitsee käytettävän moduulin.

Node.js:n moduuleita testatessa parhaiten toimivaksi ratkaisuksi nousi kolmannen osapuolen ylläpitämä Active-Window -moduuli, jolla voidaan yhdellä komennolla lukea käyttäjän sillä hetkellä aktiivisen sovelluksen järjestelmänimi ja otsikko. Moduulin lähdekoodia tutkittaessa havaittiin, että sovellus pääättelee sisäisesti käytettävän teknologian käyttäjän käyttöjärjestelmän perusteella. Moduulia testattiin eri käyttöjärjestelmillä, jolloin sen huomattiin tekevän sen mitä lupaakin noutaessaan käyttäjän aktiivisen sovelluksen nimen ja otsikon.

Aktiivisen sovelluksen määrittäminen toteutettiin opinnäytetyössä kutsumalla aktiivisen sovelluksen määrittävää funktiota yhden kerran sekunnissa ja kirjaamalla ylös sekunti lisää käyttöaika kyseisellä hetkellä aktiivisena olleelle sovellukselle. Tämä tieto lähetetään sitten muun datan yhteydessä web-palvelinsovellukselle, joka tallentaa tiedon sovelluskohtaisesti tietokantaan. Ratkaisumalli on hieman epätarkka sen tarkastusvälinsä vuoksi, koska kyseisen niin sanotun hiljaisen sekunnin sisällä tapahtuvia aktiivisen sovelluksen muutoksia toteutettu sovellus ei osaa ottaa millään tavalla huomioon. Sekunnin aikajakson todettiin kuitenkin olevan tässä sovelluksen kehityksen vaiheessa riittävän tarkka toteuttaakseen vaatimusmäärittelyn mukaisen toiminnan.

#### 5.6.4 Datan kerääminen ja lähettäminen

Työpöytäsovellus toteutettiin käytännössä vain ja ainoastaan keräämään dataa. Sovelluksen ei periaatteessa ole tarkoitus kyetä mihinkään muuhun kuin datan keräämiseen ja sen lähettämiseen tämän jälkeen web-palvelinsovellukselle. Tämän vuoksi ratkaisumalliksi valittiin yksinkertainen menettelytapa kerätä dataa yhtäjaksoisesti tietyn ajanjakson verran ja lähettää tämä koko tuona aikana kerätty data yhdellä kertaa web-palvelinsovellukselle tallennettavaksi. Tällöin palvelimelle lähetettävien viestien

määrä vähenee, jolloin se joutuu tekemään toistuvaa työtä suhteessa vähemmän verraten tilanteeseen, jossa esimerkiksi jokaisella näppäimenpainalluksella lähetettäisiin kyseisestä painalluksesta tallennettava data palvelimelle.

Sovellus kerää dataa mahdollisimman tarkalla tasolla kuitenkin siten, että tallennetusta ja lähetetystä datasta ei voida päätellä, mitä käyttäjä on kirjoittanut tai hiirtään heiluttanut missäkin sovelluksessa. Aktiivisten sovellusten osalta aika lasketaan ennen lähetystä yhteen, jolloin eri sovellusten käyttöjärjestystäkään ei voida kerätystä datasta suoraan päätellä. Datan tarkkuuden rajoittaminen on toisaalta myöhempää datan esittämistä ja raporttien muodostamista rajoittava, mutta samalla myös osaltaan tietoturvaa lisäävä tekijä.

Dataa kerätään NWJS-sovelluksen sisällä tietokoneen muistiin, josta sitä samaan aikaan tallennetaan myös sovelluksen käyttämän selaimen paikalliseen muistiin, localStorageiin. Kun muistissa oleva data on tietyin väliajoin saatu onnistuneesti lähetettyä ja tästä on saatu kuittaus web-palvelinsovellukselta, poistetaan kyseinen tieto paikallisesta muistista. Tämä menettelytapa auttaa esimerkiksi internetyhteyshäiriöiden ylläpitäen epäonnistuneiden datanlähetysten myötä heitteille jääneen datan muistissa ja lähettämällä tämän internetyhteyden jälleen toimiessa.

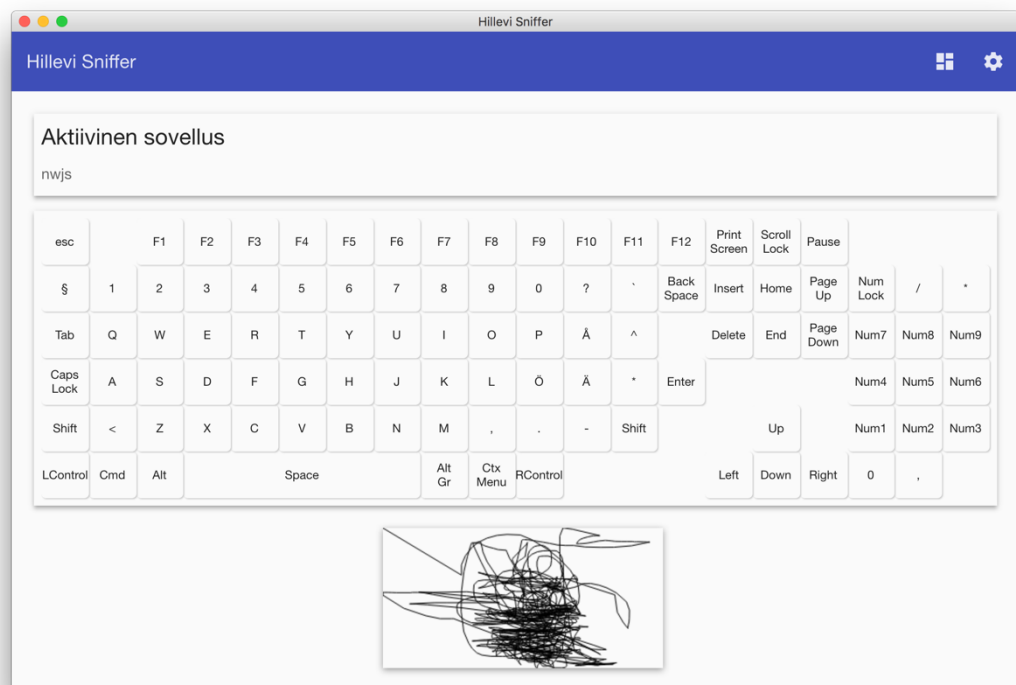
#### 5.6.5 Työpöytäsovelluksen käyttöliittymä

Työpöytäsovelluksen varsinaiseen toiminnallisuuteen ei lähtökohtaisesti kuulunut mitään käyttäjän itse suoritettavaa. Minimissään sovelluksen oli kuitenkin tarjottava käyttäjälle yksinkertainen tapa kertoa, kuka hän on ja mihin sovelluksen tulisi dataansa lähettää. Tätä varten hyvin yksinkertainen käyttöliittymä oli käytännössä ehdottomuus.

Koska NWJS on käytännössä yhtä web-selaimen kanssa, voitiin käyttöliittymä ohjelmoida käyttäen täsmälleen samoja teknologioita kuin web-selainsovelluksen toteutuksessa käytettiin. Varsinainen sovelluslogiikka toteutettiin käyttäen AngularJS:ää ja käyttöliittymän komponentit muodostettiin Angular Materialin tarjoamia käyttöliittymäkomponentteja hyväksi käyttäen. Lisäksi käyttöliittymään toteutettiin kehitystä

helpottamaan reaaliajassa päivittyvät indikaattorit aktiiviselle sovellukselle, näppäimistölle ja hiirelle (ks. kuvio 34). Nämä komponentit olivat pääosin samoja jo aiemmin web-selainsovellukseen toteutettujen komponenttien kanssa.

Käyttäjän tunnistautumiseen sovellus käyttää web-sovelluksen tarjoamaa rajapinta-avainta. Avain voidaan noutaa käyttäjän omista asetuksista web-selainsovelluksen kautta ja liittää sitten työpöytäsovelluksen asetussivulle (ks. kuvio 35). Tämän jälkeen yhteys voidaan testata ja kun se ollaan todettu toimivaksi, on sovellus käyttövalmis. Tunnistautumisen jälkeen sovellus alkaa automaattisesti lähettää kerättyä dataa web-selainsovellukseen aina välittömästi käynnistymisensä jälkeen.



Kuvio 34. Työpöytäsovelluksen käyttöliittymään toteutetut reaaliaikakomponentit

Web-rajapinnan asetukset

Rajapintasi verkko-osoite (esim. <https://hillevi-instanssi.com/api>) \*  
<https://demo.hillevi.xyz/api>

Tunnistautumisavain \*

TALLENNJA TESTAA YHTEYS

Kuvio 35. Web-rajapinnan asetuskäyttöliittymä työpöytäsovelluksessa

## 5.7 Järjestelmän käyttöönotto Ubuntu-käyttöjärjestelmälle

### 5.7.1 Yleistä

Sovelluksen vaatimusmäärittelyssä todettiin, että sovelluksen on oltava asennettavissa Linux-käyttöjärjestelmälle. Opinnäytetyössä tavoitteena oli lisäksi asentaa valmis sovellus Raspberry Pi 3 -minitietokoneelle siten, että sitä voitaisiin käyttää paikallisesti esimerkiksi muutoin internetyhteydettömässä tilassa, kuten opinnäytetyön yhteistyökumppanin työtiloissa. Sovellukseen käytettävät komponentit tuli siis valita siten, että tarvittaessa sovellus voidaan myös kaikkine toimintoineen asentaa onnistuneesti myös tälle yleisesti palvelintietokoneita rajoittuneemmalle alustalle.

Työssä sovellus asennettiin yhdelle web-palvelimelle kaikki sovelluksen palvelintietokoneelle asennettavat sovellukset: PHP-pohjainen web-palvelinsovellus, sen käyttämä tietokanta sekä web-selainsovelluksen resurssitiedostot tarjoava Apache-palvelinohjelma mukanaan kaikki tarvittavat resurssitiedostot. Kappaleissa 5.7.2-5.7.5 kuvataan sovelluksen käyttöönoton eri vaiheet. Sovellukset asennettiin puhtaalle Ubuntu Server 16.04 -alustalle ilman aikaisempia asennuksia tai konfigurointeja.

### 5.7.2 Tarvittavat sovellukset ja mahdolliset lisäosat

Järjestelmän asentaminen aloitettiin asentamalla sovellukseen tarvittavan tietokannan toteuttamiseen vaadittava MariaDB-tietokannanhallintajärjestelmä. Tietokannanhallintajärjestelmän asentamisen jälkeen on suositeltavaa suorittaa *mysql\_secure\_installation*-komento, joka ohjaa muun muassa vaihtamaan pääkäyttäjän salasanan (ks. kuvio 36).

Tietokannan asentamisen jälkeen luotiin tietokantaan uusi käyttäjä, jona web-palvelinsovellus myöhemmin tulisi kirjautumaan tietokantaan. Uusi käyttäjä voidaan luoda tietokantaan kirjautumalla ensin tarvittavat oikeudet omaavalla käyttäjällä sisään ja tämän jälkeen edelleen SQL-lausekkeita käyttämällä luomalla uusi käyttäjä ja antamalla tälle tarvittavat oikeudet (ks. kuvio 37). Tällä tavoin sovellusta käyttävällä tietokantakäyttäjällä on vain tiettyjä oikeuksia, jolloin mahdollista vahingon määrää mahdollisten väärinkäytösten osalta rajoitetaan jo tietokannan tasolla.

```

you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorisation.

Set root password? [Y/n] y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
... Success!

By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them. This is intended only for testing, and to make the installation
go a bit smoother. You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y

```

Kuvio 36. mysql\_secure\_installation-asennusohjelma

```

MariaDB [(none)]> CREATE USER 'hillevi'@'localhost' IDENTIFIED BY 'thisisyourpassword';
Query OK, 0 rows affected (0.01 sec)

MariaDB [(none)]> GRANT SELECT, INSERT, DELETE, DROP, CREATE, ALTER ON *.* TO 'hillevi'@'localhost';
Query OK, 0 rows affected (0.00 sec)

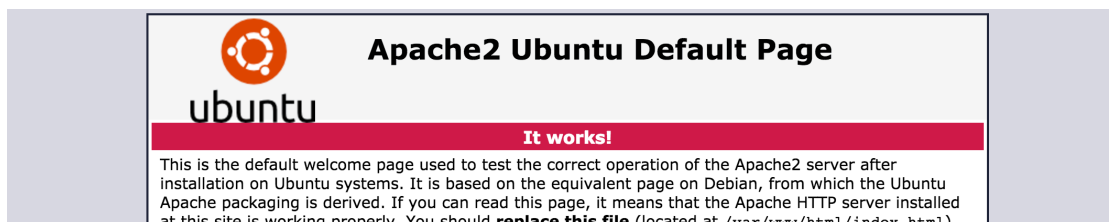
MariaDB [(none)]>

```

Kuvio 37. Uuden käyttäjän luonti tietokantaan

Tietokannan ja käyttäjien määrittelyn jälkeen asennettiin varsinaiselle web-palvelinsovellukselle oleelliset kirjastot. Samalla asennettiin myös Apache-palvelinohjelma, joka myöhemmässä vaiheessa tulisi tarjoamaan web-selainsovelluksen käyttämät resurssit. Asennus aloitettiin Apache-palvelinohjelmasta, jonka jälkeen asennettiin PHP7 ja sille koko joukko lisäpaketteja laajentamaan ja tuomaan tehokkuutta PHP7:n toimintaan. Lisäksi jotkin paketeista ovat jopa välttämättömiä web-palvelinsovelluksen toiminnan kannalta.

Asennuksien jälkeen Apache-palvelinohjelma käynnistettiin uudestaan kaikkien muutosten voimaan saattamiseksi. Uudelleenkäynnistytksen jälkeen kaikki järjestelmän riippuvuudet olivat asennettuina ja siirryttäessä selainsovelluksella osoitteeseen 192.168.100.15 eteen aukeni Apache-palvelinohjelman oletussivu (ks. kuvio 38).



Kuvio 38. Apache-palvelinohjelman oletussivu

Web-selainsovelluksen käyttöönotto vaati muutamien JavaScript-työkalujen asentamisen web-palvelimelle. Sovellus käyttää riippuvuuksien hallintaan npm-paketinhallintajärjestelmää ja tiedostojen pakkaamiseen Grunt-työkalua, joka taas vaatii toimiakseen toimivan Node.js-asennuksen. Käyttöönotossa ladattiin ensin Node.js ja npm käyttämällä apt-paketinhallintajärjestelmää, minkä jälkeen loput riippuvuudet voitiin ladata käyttämällä asennettua npm:ää.

### 5.7.3 Web-palvelinsovelluksen asentaminen

Kaikkien riippuvuuksien ja järjestelmien ollessa asennettuina jäi jäljelle tehtäväksi enää ladata kaikki sovelluksen lähdekoodit, määrittää asetukset ja todeta sovelluksen toiminta. Järjestelmien asennus aloitettiin web-palvelinsovelluksen lataamisella. Tiedostot ladattiin GitHub-palvelusta käyttäen Ubuntu 16.04 -versioon esiasennettua git-komentorivityökalua. Myöhemmin sovelluksen päivittäminen voidaan tehdä käyttämällä käytännössä samaa työkalua.

Tiedostojen lataamisen jälkeen sovelluksen kaikki riippuvuudet täytyi asentaa käyttämällä composer-paketinhallintajärjestelmää. Asennuksen pystyi hoitamaan suorittamalla php-komentorivityökalulla mukana tulleen composer.phar-tiedoston avulla composerin *install*-komennon. Tämä asensi kaikki sovelluksen riippuvuudet ja avasi lopuksi interaktiivisen näkymän, jossa kysytään käyttäjältä sovellusta varten tarvittavat parametrit (ks. kuvio 39). Tässä vaiheessa on hyvä olla perillä käytettävistä asetuksista, tunnuksista ja salasanoista, sillä määrittäykset tallentuvat suoraan parameters.yml-tiedostoon (ks. kuvio 7), josta Symfony lukee käyttämänsä parametrit.

```

Creating the "app/config/parameters.yml" file
Some parameters are missing. Please provide them.
database_host (127.0.0.1):
database_port (null):
database_name (symfony):
database_user (root): hillevi
database_password (null): thisisyourpassword
mailer_transport (smtp):
mailer_host (127.0.0.1):
mailer_user (null):
mailer_password (null):

```

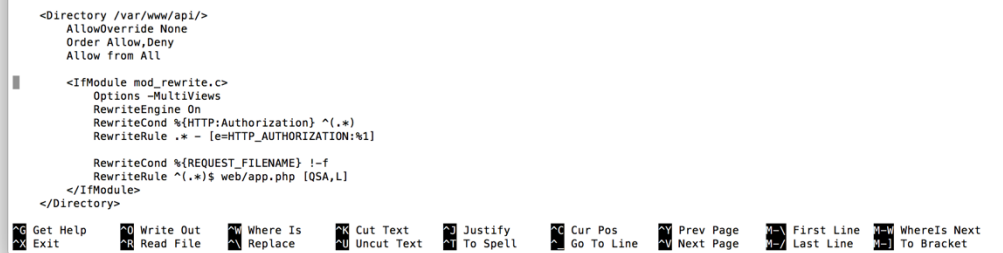
Kuvio 39. Symfonyn interaktiivinen parametrienkyselytoiminto

## Tietokannan populointi

Parametrisoinnin jälkeen voitiin ajaa erikseen tietokannan muodostamista varten toteutettu, useista Symfonyn komennoista muodostuva shell-skripti *create\_database.sh*. Skripti luo tietokannan parametreissa määritellyllä nimellä, jonka jälkeen muodostaa tietokantaan skeeman sovelluskoodissa esiintyvistä entiteeteistä. Näiden jälkeen tietokantaan ajetaan kaikki tallennetut proseduurit ja funktiot sekä muodostetaan aikadimensiotaulu. Tietokantamigraatiot ajetaan yksi kerrallaan tietokantaan, minkä jälkeen tietokantaan ladataan myös oletusdata, jonka avulla ensimmäinen käyttö on mahdollista. Oletusdataan kuuluu pääkäyttäjän oikeuksilla varustettu käyttäjä sekä tälle käyttäjälle yksi esimerkkirajapinta aloituskynnyksen pienentämiseksi. Lisäksi Symfonyn välimuistituksen vuoksi tuotantoympäristössä Symfonyn välimuisti oli muutosten voimaansaattamiseksi tyhjennettävä Symfonyn komennolla *cache:clear*.

## Apache-palvelinohjelman määrittely

Jotta REST-rajapinta voisi toimia, täytyy tiettyyn polkuun tulleet kyselyt ohjata tietylle PHP-sovellukselle, jotta tämä voi edelleen ohjata kyselyn oikeaan sovelluskoodin haaraan. Ohjaus toteutettiin käyttämällä Apache-palvelinohjelmaan tätä varten kehitettyjä toimintoja. Apache-palvelinohjelman määrittelytiedostoon lisättiin ehto uudelleenohjaukselle, joka ohjaa kaikki /api-polkuun tulleet HTTP-pyynnöt Symfony-sovellukselle koko sovelluksen alkupisteeseen, app.php-tiedostoon (ks. kuvio 40). Näin myöhemmin web-selainsovelluksen tekemät HTTP-pyynnöt kulkeutuvat web-palvelinsovelluksessa aina oikeaan palvelinsovelluksen metodiin saakka.



```

<Directory /var/www/api/>
    AllowOverride None
    Order Allow,Deny
    Allow from All

    <IfModule mod_rewrite.c>
        Options -MultiViews
        RewriteEngine On
        RewriteCond %{HTTP:Authorization} ^(.*)
        RewriteRule .* - [e=HTTP_AUTHORIZATION:%1]

        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteRule ^(.*)$ web/app.php [QSA,L]
    </IfModule>
</Directory>

```

The screenshot shows a text editor window with an Apache configuration file. The file contains a `<Directory /var/www/api/>` block with `AllowOverride None`, `Order Allow,Deny`, and `Allow from All`. Inside this block, there is an `<IfModule mod_rewrite.c>` section. This section includes `Options -MultiViews`, `RewriteEngine On`, a `RewriteCond` that checks for an `HTTP:Authorization` header, a `RewriteRule` that appends the authorization header to the request, another `RewriteCond` to ensure the file does not exist (`!-f`), and a `RewriteRule` that routes all requests to `web/app.php` with the `QSA,L` flags. The bottom of the window shows a toolbar with various editing functions like 'Get Help', 'Exit', 'Write Out', 'Read File', 'Where Is', 'Replace', 'Cut Text', 'Uncut Text', 'Justify', 'To Spell', 'Cur Pos', 'Go To Line', 'Prev Page', 'Next Page', 'First Line', 'Last Line', 'WhereIs Next', and 'To Bracket'.

Kuvio 40. Apache-palvelinohjelman uudelleenohjauksen konfigurointi

Määrittelyjen tekemisen Apachelle tuli sallia *rewrite*-moduulin käyttö komennolla *a2enmod rewrite*, jonka jälkeen muutokset tuli ottaa voimaan lataamalla Apachen asetukset uudestaan komennolla *service apache2 reload*. Tämän jälkeen sovellus oli valmis toimintaan.

#### 5.7.4 Web-selainsovelluksen asentaminen

Käytettäessä Apache-palvelinohjelmaa on selainsovelluksen tarjoaminen web-palvelimelta kaiken kaikkiaan hyvin yksinkertaista. Apache tarjoaa oletusasetuksillaan staattisesti tiedostot web-palvelimen hakemistosta `/var/www/html`, josta ilman muita määrittelyksiä se etsii *index*-nimistä tiedostoa. Täten siirryttäessä web-selaimella web-palvelimen ulkoiseen osoitteeseen joko erillisen *domainin* tai suoran ip-osoitteen avulla, palauttaa Apache-palvelinohjelma oletushakemistosta *index*-tiedoston, jonka sisällön web-selain omalla tavallaan esittää käyttäjälle.

Opinnäytetyössä web-selainsovelluksen sisältämät tiedostot ladattiin Apache-palvelinohjelman oletushakemistoon. Kaikki sovelluksen tarvitsemat riippuvuudet asennettiin komennolla *npm install*, joka lataa *package.json*-tiedostossa määritellyt riippuvuudet *node\_modules*-kansioon. Tämän jälkeen luotiin konfigurointitiedosto *env/prod.js* kopioimalla *env*-hakemiston sisältä löytyvä *env.example.js*-tiedosto uudeksi tiedostoksi. Tiedostossa määritellyt vakioita sovellus tulisi myöhemmin käyttämään tehdesään kyselyitä web-palvelinsovellukseen (ks. kuvio 41). Parametreihin määritellään käytettävän rajapinnan osoite sekä mahdollisen WebSocket-yhteyden osoite.



```

/**
 * Constant inits.
 */
angular.module('App.Env')
  .constant('API', {
    url: 'https://myapi.myhillviinstance.com'
  })
  .constant('WS', {
    url: 'wss://mysocket.myhillviinstance.com'
  })
;

```

Toolbar icons: Get Help, Exit, Write Out, Read File, Where Is, Replace, Cut Text, Uncut Text, Justify, To Spell, Cur Pos, Go To Line, Prev Page, Next Page, First Line, Last Line, WhereIs Next, To Bracket.

Kuvio 41. Parametrien määrittelytiedoston sisältö

Parametrien määrittelyiden jälkeen sovelluksen tarvitsemat JavaScript- ja CSS-tiedostot pakattiin käyttämällä Grunt.js-työkalua komennolla *grunt*. Kun komento oli ajettu, voitiin sovelluksen toiminta testata siirtymällä web-selaimella web-palvelimen osoitteeseen.

### 5.7.5 Ensimmäinen käyttökerta

Sovelluksen ensimmäinen sisäänkirjautuminen voitiin suorittaa käyttämällä tietokannan oletusdatan lataamisen yhteydessä sinne lisätyn pääkäyttäjän tunnuksia. Tällöin käyttäjätunnukseksi on hyvin yleisesti käytetty *admin* ja salasana *admin*. Ensimmäisen kirjautumisen yhteydessä salasana vaihdettiin käyttäjän asetusvälilehden kautta, jonka jälkeen sovellukseen luotiin uusi käyttäjä varsinaista sovelluksen käyttöä varten. Uudelle käyttäjälle annettiin tarvittavat käyttöoikeudet sovellukseen, minkä jälkeen sovelluksen käyttö jatkui käyttämällä uutta käyttäjätunnusta. Uuden käyttäjän oletus-salasanaksi generoitiin automaattisesti järjestelmän puolesta käyttäjätunnus + kuluva vuosi, jolloin esimerkiksi luodulle käyttäjätunnukselle *jorma* tätä vastaava salasana on vuonna 2016 *jorma2016*.

## 5.8 Lisämoduulit ja ohi vaatimusmäärittelyn toteutetut ominaisuudet

### 5.8.1 Lämpötilaa mittaava moduuli

Opinnäytetyössä käytännössä välittömästi datan tallentamisen mahdollistavien ominaisuuksien toteuttamisen jälkeen toteutettiin yksittäisen huoneen lämpötilaa mit-

taava moduuli, joka lähettää dataa toteutettuun web-palvelinsovellukseen. Datan tallennus aloitettiin aikaisessa vaiheessa, jotta myöhemmän vaiheen raporttien ja kuvaajien toteutuksessa voitaisiin käyttää niin sanotusti aitoa dataa.

Lämpötilaa mittaavan moduulin toteutuksessa periaatteellisena ajatuksena oli niin kutsuttu *quick and dirty* -menetelmä, joka tarkoittaa käytännössä tavoitteen saavuttamista mahdollisimman nopeasti sen suuremmin keskittymättä esimerkiksi arkkitehtuurillisiin ratkaisuihin tai ylläpidettävyyteen. Moduuli toteutettiin käyttäen Arduino Uno-kehitysalustaa, johon kytkettiin TMP36-lämpöanturi. Tämä kokonaisuus kytkettiin sitten Raspberry Pi -minitietokoneeseen, jolle ohjelmoitiin Node.js:ää käyttämällä datan edelleen web-palvelinsovellukselle lähetävä sovellus.

Arduinolle ohjelmoitiin sovellus, joka lukee lämpöanturilta lukeman sekunnin välein. Mitatuista arvoista sovellus laskee minuutin ajalle keskiarvon, jonka lähettää sitten sarjaliikenneväylää pitkin Raspberry Pi -minitietokoneelle jatkokäsiteltäväksi. Minuutin jakson jälkeen mittaustulokset nollataan ja keskiarvon laskeminen aloitetaan alusta.

Raspberry Pi -minitietokoneelle toteutettiin sovellus, joka toimii ikään kuin välittäjänä Arduinon ja web-palvelinsovelluksen välillä. Toteutettu Node.js-sovellus kuuntelee sarjaliikenneväylästä Arduinolta saapuvia viestejä ja viestin saapuessa lähettää tämän web-palvelinsovellukselle. Mikäli internetyhteyttä ei ole saatavilla, lisätään saatu mittaustulos jonoon muiden mittaustulosten kanssa ja edelleen internetyhteyden jälleen ollessa avoin lähetetään kaikki data kerralla web-palvelinsovellukselle tämän tarjoamaa rajapintaa hyödyntäen.

### 5.8.2 Tietokannan varmuuskopiointi

Tietyssä vaiheessa sovelluksen kehitystä tuli vastaan tilanne, jossa tietokantaan oli keräytynyt niin paljon mitattua dataa, että sen menettäminen ei olisi ollut mielekäästä ja uudelleen kerääminen käytännössä mahdotonta. Tietokannan tietojen säilömisen varmistamiseksi toteutettiin sovellukseen automaattinen tietokannan varmuuskopiointi.

Varmuuskopiointiin toteuttamiseen hyödynnettiin Symfony-yhteensopivaa Dizda-CloudBackupBundlea, jonka käyttöönottamiseksi riittää pelkkä tietyn tai tiettyjen pil-

vipalveluiden tietojen sekä näiden rajapinta-avainten lisääminen sovelluksen konfigurointitiedostoon. Varmuuskopiointi suoritetaan määrittelyjen jälkeen suorittamalla komentorivillä komento `php bin/console --env=prod dizda:backup:start`, jonka tuloksena syntyvä varmuuskopiotiedosto lähetetään Dropbox-pilvipalveluun käyttämällä Dropboxista saatavaa rajapinta-avainta. Lisäksi manuaalisen varmuuskopioinnin tarpeen välttämiseksi lisättiin edellä mainitun komennosta uusi säännöllinen tehtävä (cronjob), joka suorittaa varmuuskopioinnin joka yö kello kolme.

### 5.8.3 Ilmoitusjärjestelmä

Sovelluskehityksen aikaan huomattiin, että sovellukseen olisi hyvä voida lähettää web-palvelimelta ilmoituksia tiettyjen sen suorittamien operaatioiden lopputuloksista tai mahdollisista virhetilanteista. Jotta web-selainsovelluksen ei tarvitsisi kysellä mahdollisia uusia viestejä web-palvelimelta tietyin väliajoin, toteutettiin opinnäytetyössä myös osittain kokeilumielessä reaaliaikainen viestijärjestelmä käyttäen WebSocket-teknologiaa.

#### **WebSocketit lyhyesti**

WebSocketeja käytettäessä mahdollistetaan kaksisuuntainen vuorovaikutus asiakaspalvelin -rajapinnassa, mikä antaa myös web-palvelimelle mahdollisuuden olla vuorovaikutteen aloittaja. Avattu yhteys on siis jatkuva eikä sitä käytännössä suljeta välillä ollenkaan. Avoinna olevassa yhteydessä molemmat yhteyden osapuolet voivat lähettää keskenään viestejä toisilleen, mikä mahdollistaa ketterän ja nopean viestiliikenteen. (West 2013.)

Ilmoitusjärjestelmä toteutettiin irralleen koko muusta järjestelmästä, jolloin sen käyttäminen tai edes mahdollistaminen ei ole välttämätöntä muun sovelluksen toiminnalle. Web-palvelinsovelluksessa WebSocket-ohjelmointiin käytettiin Ratchet-kirjastoa. WebSocket-palvelimen muodostamiseen toteutettiin yksi Symfony-komento ja varsinainen viestien lähettäminen yleistettiin yhden staattisesti käytettävän metodin tasolle.

Web-selainsovelluksen osalta WebSocket-käsittelyyn käytettiin AngularJS-yhteensopivaa ngWebSocket-moduulia. Moduuli yksinkertaistaa WebSocket-yhteyden käsittelyä AngularJS-sovelluksessa toteuttaen samaa arkkitehtuurillista ideologiaa muidenkin

AngularJS-moduulien kanssa. WebSocket-yhteyden ollessa mahdollinen yhdistää web-selainsovellus itsensä automaattisesti web-palvelinsovelluksen avaamaan WebSocket-yhteyteen. Tämän jälkeen kaikista WebSocket-yhteydestä vastaanotetuista viesteistä muodostetaan välittömästi viestin saapuessa Angular Materialin mdToast-komponenttia käyttäen viesti käyttäjän web-selaimen ruudulle. Lisäksi yhteyden katkaisemiseen toteutettiin myös erillinen toiminto siltä varalta, että käyttäjä ei jostain syystä haluaisikaan nähdä ilmoituksia ruudullaan.

## 6 Lopputulokset ja johtopäätökset

### 6.1 Lopullinen tuote













Opinnäytetyössä konkreettisena lopputuloksena saatiin toteutettua web-sovellus varsinaisen datan käsittelyyn ja hallintaan sekä työpöytäsovellus tietokoneen käytön analysointidatan keräämiseen ja lähettämiseen edelleen web-sovellukselle. Lisäksi toteutettiin lämpötilaa mittaava moduuli sekä muutama eri datan lähettämiseen erikoistunut työkalu. Lisäksi sovellukseen saatiin toteutettua muutamia lisäominaisuuksia, joita ei alkuperäisessä vaatimusmäärittelyssä mainittu lainkaan.

Web-sovellukseen saatiin toteutettua idean toimivaksi osoittavalla tasolla muutama eri raportti ja dataa visualisoiva kuvaaja, joista voidaan havainnoida eri tunnuslukujen suhteita keskenään. Lisäksi työpöytäsovellukselta kerätyn datan esittämiseen saatiin toteutettua kuvaajia, joista kerättyä dataa voidaan tutkia ja käyttää niitä myöhemmin jatkokehityksen lähtökohtina.







Lopputulos saatiin toteutettua sellaiseksi, että se voidaan myöhemmin ottaa käyttöön muissakin käyttöympäristöissä. Opinnäytetyön aikaan sovellus saatiin asennettua myös Raspberry Pi 3 -minitietokoneelle, jonka myötä sen käyttöönotto opinnäytetyön yhteistyökumppanin tiloissa voidaan myöhemmin tulevaisuudessa toteuttaa.

Toteutetun sovelluksen nimeksi valittiin Hillevi. Nimen valintaan vaikutti suuresti varsinaisen aiheen idean keksimisen päivänä nimipäiväänsä viettävien henkilöiden nimet. Sovelluksen eri moduulit nimettiin laajentamalla Hillevi-sanaa, jonka myötä tietokoneen käyttöä analysoivan sovelluksen nimeksi valittiin Hillevi Sniffer – ”haistelija”.










Sovelluksen lähdekoodin määrää arvioitaessa ja peilatessa lopputulokseen ja sen moninaisuuteen voidaan todeta lähdekoodin määrän jäävän verrattain pieneksi. Opinnäytetyössä käytetyn käyttöliittymäkomponentin ansiosta etenkin web-selainsovelluksen tyylimäärittelyiden teko jäi melko vähäiseksi muodostuen hädin tuskin muutamasta sadasta rivistä tyylimäärittelyjä (ks. kuvio 42). Web-palvelinsovelluksen haukatessa merkittävimmän osan käytetystä lähdekoodista (ks. kuvio 43) voidaan todeta logiikan painottuvan eritoten rajapinnan toteutukseen. Työpöytäsovellukseen kulutetut resurssit olivat sen sijaan ennako-odotusten mukaisesti kaikista pienimmät ja kirjoitetun ohjelmakoodin määrä sen osalta kaikista vähäisin (ks. kuvio 44).

Extension ▲	Count	Size	Lines
 <b>css</b> (CSS files)	2x	 4kB	 292
 <b>html</b> (HTML files)	42x	 84kB	 2003
 <b>js</b> (JS files)	66x	 131kB	 5147
 <b>json</b> (JSON files)	3x	 9kB	 277

Kuvio 42. Web-selainsovelluksen lähdekoodin määrä

Extension ▲	Count	Size	Lines
 <b>php</b> (PHP files)	98x	 277kB	 13370
 <b>sql</b> (SQL files)	6x	 10kB	 346

Kuvio 43. Web-palvelinsovelluksen lähdekoodin määrä

Extension ▲	Count	Size	Lines
 <b>css</b> (CSS files)	1x	 1kB	 117
 <b>html</b> (HTML files)	4x	 3kB	 100
 <b>js</b> (JS files)	16x	 39kB	 1361

Kuvio 44. Työpöytäsovelluksen lähdekoodin määrä

## 6.2 Testauttaminen

Tärkeä osa sovelluksen toteutusta on myös sen testauttaminen sillä tai vastaavalla henkilöryhmällä, jolle sovellus tullaan myöhemmin ottamaan käyttöön. Testauttaminen on tärkeää siksi, että sovellusta ohjelmoiva ryhmä tulee usein hyvin sokeaksi omalle tekemiselleen, jonka myötä ulkopuolinen objektiivinen näkökulma sovelluskehityksessä avaa usein uusia näkökulmia eri toimintojen tiimoilta.

Opinnäytetyössä toteutettuja sovelluksia testautettiin kehityksen aikaan useilla hyvin eri tyyppisillä käyttäjillä, joiden iät vaihtelivat aina ala-asteikäisestä eläkeikää lähesty-

vään. Sovelluksesta testattiin eniten sen käytön aloituskynnystä – miten helppo henkilön on ymmärtää, mitä sovellus tekee ja miten sitä käytetään? Toisaalta testautuksen yhteydessä havainnoitiin myös sitä, kokeeko käyttäjä sovelluksen käytön myötä saaduista tuloksista olevan mitään hyötyä.

Testausasetelmassa käyttäjälle tehtiin järjestelmään uusi käyttäjä, ja hänen annettiin vapaasti käyttää sovellusta parhaaksi kokemallaan tavalla. Käyttäjälle kerrottiin, että sovellukseen voidaan lukea automaattisesti tai sinne voidaan lähettää dataa, jota tallennetaan sovellukseen käyttäjän määrittelemällä tavalla tietokantaan. Lisäksi kerrottiin, että tietokoneella on käynnissä sovellus, joka mittaa tietokoneen käyttöä ja kerätty data lähetetään kyseiseen sovellukseen, jonka kautta dataa voidaan analysoida visuaalisesti. Käyttäjälle eri siis erikseen annettu listaa tehtäviä vaan tutkittiin, miten käyttäjä liikkuu järjestelmässä käyttäen omaa intuitiotaan.

Testautuksen tuloksista voitiin päätellä, että sovellus on suhteellisen vaikeasti lähestyttävä ilman varsin kattavaa alustusta. Haasteita toivat esimerkiksi tuntemattomat käsitteet ja yleisesti koko rajapintojen maailman tuntemattomuus. Toisaalta sovellus vaatii datan keräämiseksi jonkinlaisia rajapintoja, joita arvatenkaan keskivertokansalaisella ei ole ihan heti tarjota. Varsinainen datan hakunäkymä ja sen vertailu olivat kuitenkin helpommin ymmärrettäviä – testihenkilöt osasivat valita eri vertailtavat arvot sekä käyttää haussa hakusuodattimia. Sovelluksen voidaankin todeta olevan osittain niin sanottu insinööriytyökalu, jonka käyttö vaatii omanlaistaan vihkiytymistä asiaan.

### 6.3 Havaitut viat ja puutteet

Yksikään sovellus ei ole täydellinen. Opinnäytetyön toteutuksen sekä testautuksen yhteydessä havaittiin sovelluksen toiminnassa epäloogisuuksia ja jopa suoranaisia virheitä, joiden korjausta ei kuitenkaan ehditty opinnäytetyön aikaan toteuttaa. Jotkin sovellusten toiminnassa havaitut virheet paikannettiin jo tarkasti tietyn ohjelmakoodin kohtaan, mutta jotkin vioista jäivät edelleen selvittämättä.

Sovelluksessa havaittiin selkeitä virheitä pääosin tietokonetta kuuntelevan sovelluksen toiminnassa. Sovellus toimii näennäisesti oikein yhdellä käyttöjärjestelmällä, mutta kun sovellus otetaan samalla käyttäjällä käyttöön toisessa käyttöjärjestelmässä,

esimerkiksi aktiivisten sovellusten tallentaminen sotkeutuu täysin. Tämä johtuu siitä, että aktiivisen sovelluksen etsiminen perustuu täysin käyttöjärjestelmän tarjoamaan sovelluksen nimeen ja otsikkoon, jotka saman sovelluksen kyseessä ollessakin poikkeavat täysin eri käyttöjärjestelmien välillä. Lisäksi puutteita havaittiin erikoisnäppäinten painallusten rekisteröinnissä eri näppäimistöasetteluilla, minkä myötä näppäinten kuormituskuvaajaan täytyy käytetystä käyttöjärjestelmästä ja näppäimistöä riippuen suhtautua hieman skeptisesti.

Varsinaisten ohjelmointivirheiden lisäksi toteutuksesta jäi puuttumaan myös vaatimusmäärittelyyn kirjattuja kohtia. Merkittävin sovelluksen puute on dokumentaatio – niin sovelluksen tekninen kuin myös semanttinen dokumentaatio on käytännössä olematon. Sovellukseen ei ehditty toteuttaa varsinaista käyttöohjetta ollenkaan. Vaikeimman vaiheen, sovelluksen käyttöönoton, ohjeet saatiin kuitenkin dokumentoitua. Myöskään työpöytäsovellukselle asetettua vaatimusta sovelluksen asentamisen ja käyttöönoton helppoudesta ei saatu toteutettua.

## 6.4 Kehitysideat ja –haaveet

Sovelluksen kehityksen aikaan esiin nostettiin useita eri jatkokehitysideoita. Ideat kumpusivat täysin uusista toiminnoista ja toisaalta myös jo toteutetuista toiminnoista sekä näiden parempaan käyttökokemukseen ja informatiivisuuteen liittyvistä ominaisuuksista. Osaltaan kehitysideat huomioitiin jo opinnäytetyön toteutuksen aikaan, jolloin esimerkiksi sovelluslogiikassa tehtäviä kaikkia toimintoja pohjustavia ratkaisuja mietittiin ei pelkästään opinnäytetyössä toteutettavan sovelluksen osalta vaan myös tulevaisuus huomioon ottaen.

### 6.4.1 Sovelluksen käyttö ryhmien tietojen seuraamiseen

Eräs suurimmista tulevaisuuden jatkokehitysjatoksista on mahdollistaa käyttäjäryhmien käsittely sovelluksen sisällä. Tällöin voitaisiin kerätä dataa käyttäjäkohtaisesti, mutta esimerkiksi järjestelmärajapintojen osalta kerättyä tietoa voitaisiin verrata myös eri henkilöiden kesken. Lisäksi usean henkilön datan käsittelyn tuennassa on myös se etu, että toinen henkilö voisi mahdollisesti päästä tarkastelemaan käyttöoikeuksiensa puitteissa toisen henkilön dataa. Näin esimerkiksi sovellusta voitaisiin käyttää esimerkiksi ohjelmistokehitystiimin sisällä keräten dataa tietokoneen käytöstä ja

tehdn näin mahdollisesti tiettyjä johtopäätöksiä. Jaoteltaessa käyttäjiä eri ryhmiin voitaisiin dataa hakea esimerkiksi ryhmittäin ja verrata esimerkiksi kahden eri tiimin yhteenlaskettuja dataja keskenään.

#### 6.4.2 Ylenpalttinen modulaarisuus

Tulevaisuuden haaveena olisi myös mahdollistaa sovelluksen sisällä eri moduulien käyttöönotto pelkästään määrittämällä tiettyyn tiedostoon kaikki käytettävät moduulit. Sovellus osaisi ladata itse kaikki tarvittavat moduulit, tehdä tarvittavat tietokanta-ajot ja ottaa moduulin käyttöönsä. Tällöin sovelluksessa voisi mahdollisesti ottaa käyttöön myös tiettyjä kolmannen osapuolen moduuleita, joita käyttäjä voisi mahdollisesti jopa käyttöliittymän kautta käydä valitsemassa ja ottamassa käyttöön. Tällöin itse sovellus pysyisi pienenä ja kevyenä, johon lisämoduuleita tuodaan täysin käyttäjien oman valinnan mukaan.

Sovellukseen on ideoitu jo useita lisämoduuleita. Tällaisia moduuleita ovat esimerkiksi autosta tietoa keräävä moduuli, joka esimerkiksi auton ajajan puhelinta ja tämän internetyhteyttä käyttämällä keräisi auton käytöstä tiettyä dataa ja lähettäisi tämän edelleen web-palvelinsovellukselle. Lisäksi kodin tunnuslukuja mittaavaa järjestelmää on suunniteltu laajennettavaksi siten, että se sisältää useita eri mittausyksiköitä, joiden sijoittamisella eri puolelle huoneistoa voidaan verrata useista eri mittalähteistä mitattua lämpötila-, kosteus- ja ilmanpaine-arvoa toisiinsa. Lisäksi mittausyksiköihin voitaisiin lisätä esimerkiksi valoisuutta ja hiilidioksidipitoisuutta mittaavia ominaisuuksia.

#### 6.4.3 Rajapintojen käsittelyn edistäminen

Sovellukseen toteutettiin rajapintojen käsittelytoiminnot hyvin karkealla tasolla: sovellukseen voidaan automaattisesti lukea yksitasoisia rajapintoja, joista luettu data voidaan käskä muotoiltavaksi tietyllä tavalla. Rajapintojen käsittelyn edistäminen onkin yksi ensimmäisistä jatkokehitystoimenpiteistä. Rajapintojen lukuun tullaan toteuttamaan monipuolisempia muotoiluvaihtoehtoja sekä mahdollisuus asettaa tiettyjä laatuvaatimuksia sisään luettavalle datalle. Luettava arvo voitaisiin esimerkiksi pakottaa numeeriseksi välille 0-10, tai muuten sitä ei tallenneta tietokantaan.

Opinnäytetyön aikana sisään luettavan datan luku toteutettiin tukemaan vain JSON-formaattia. Myöhemmin rajapinnat on tarkoitus toteuttaa tukemaan myös muita yleisiä formaatteja, kuten XML:ää ja vaihtoehtoisesti myös käyttäjän itse määrittämää formaattia.

#### 6.4.4 Muut lisätoiminnot

Muita sovellukseen suunniteltuja käyttöä helpottavia toimintoja olisivat esimerkiksi dynaaminen etusivu, datan massatuonti- sekä massavientitoiminnot. Dynaamiselle etusivulle käyttäjä voisi itse määritellä näytettävän tiedon lisäämällä etusivulle tiettyä dataa esittäviä widgettejä. Käytännön tasolla nämä widgetit olisivat ikään kuin pieniä raportteja, joita on samalla sivulla näkyvillä yhtä aikaa useita. Käyttäjä voisi itse määritellä näytettävän datan, jolloin kaikki tärkeimmät raportit aukeaisivat aina käyttäjälle pelkästään sovelluksen etusivun avaamalla.

Massatuonti- ja massavientitoiminnot olisivat suuri etu tuotaessa dataa toisesta järjestelmästä sisään tai mahdollisesti viettäessä dataa ulos esimerkiksi analysoitavaksi sitä toisessa sovelluksessa tai mahdollisesti haluttaessa muotoilla datasta jokin tulostettava dokumentti. Tällöin data tulisi voida määrittelyiden mukaisesti tuoda sovellukseen eri formaateista kuin myös viedä ulos sovelluksesta tietyssä muodossa, kuten JSON- tai XML-formaatissa.

Sovelluksen käytön tehostamiseksi tulisi sovellukseen toteuttaa muutama erilainen useita rajapintoja käsittelevä raportointiliittymä, joissa dataa voitaisiin vertailla moninaisin vielä toteuttamattomin tavoin keskenään. Raportteihin voisi kuulua esimerkiksi listaraportteja, joihin voidaan tuoda numeerista tietoa taulukkomuotoon. Lisäksi raporteissa käytettävien kuvaajatyypien määrää voisi kasvattaa mahdollistaen juuri oikean kaaviotyyppin valinnan visualisoimaan valittua dataa.

### 6.5 Johtopäätökset

Opinnäytetyö oli oppimisprosessina mielenkiintoinen ja yhtenäinen kokonaisuus. Opinnäytetyöhön käytetty ajanjakso oli verrattain lyhyt, kaikkineen hieman yli kaksi kuukautta suunnitteluineen, ohjelmointineen ja raportointineen, mikä sinällään oli

niin haaste kuin toisaalta myös etu. Oman vivahteensa ajankäyttöön toi sen priorisoinnin ylitsepääsemätön tärkeys, mikä toisaalta oli kokemuksena myös perin opettavaista. Lisäksi tasapainottelu täysipäiväisen työnteon ja koulun välillä oli välillä hieman raskasta, joskin myöhemmin ajateltuna todennäköisesti omalla tavallaan palkitsevaa.

Opinnäytetyön toteutuksen aikaan haastavinta oli uusiin teknologioihin tutustuminen nopeasti ja valmiiden komponenttien käyttö. Aiemman ohjelmointihistorian koostuminen käytännössä täysin itse ohjelmoitujen komponenttien käytöstä tuntui toisten tekemien valmiiden ratkaisuiden käyttäminen yhtäältä vieraalta, mutta toisaalta kiehtovalta ajatukselta. Valmiiden komponenttien käytön koettiin aluksi hidastavan tekemistä, mutta myöhemmässä vaiheessa niistä paljastui hyvinkin käyttökelpoisia ominaisuuksia, jotka osaltaan nopeuttivat sovelluskehitystä.

Teknologioiden ohella vähintään yhtä haastavaa oli pitää opinnäytetyön rajausta, joka alussa asetettiin todella tiukaksi koko projektin levähtämisen ja täydellisen sekasorron estämiseksi. Toteutettava sovellus on laajuudeltaan kuitenkin sitä luokkaa, että aihe olisi äkkiä voinut harhautua pois tietokoneen analysointityökalun toteutuksesta, mikä olisi tuonut mukanaan edelleen uusia laajentamisen kohteita. Myös varsinaisen kehitystyön päättäminen opinnäytetyön osalta oli vaikeaa primitiivisen vaiston jatkuvasti ohjatessa ohjelmoimaan aina vain uusia ja uusia toimintoja.

Yhteenvetona opinnäytetyöprojekti onnistuttiin pitämään ehyenä kokonaisuutena, vaikka aihe oli sinällään laaja ja mielenkiintoa olisi riittänyt jatkokehitysideoidenkin edistämiseen jo opinnäytetyön toteutusvaiheen aikaan. Opinnäytetyöhön etukäteen asetetut vaatimukset ja tavoitteet saatiin pääosin toteutettua, minkä myötä opinnäytetyöprojektia voidaan pitää pääosin onnistuneena.

## Lähteet

About. N.d. JNativeHook-Javakirjaston GitHub-sivu. Viitattu 3.12.2016.

<https://github.com/kwhat/jnativehook>.

AngularJS - Overview. N.d. Artikkelin AngularJS:n perusominaisuuksista. Viitattu

21.11.2016. [https://www.tutorialspoint.com/angularjs/angularjs\\_overview.htm](https://www.tutorialspoint.com/angularjs/angularjs_overview.htm).

Brown, M. 2015. Understanding LAMP. Artikkelin LAMP-sovelluskokoelmasta. Viitattu 21.11.2016.

<http://www.serverwatch.com/tutorials/article.php/3567741/Understanding-LAMP.htm>.

Capan, T. 2013. Why The Hell Would I Use Node.js? Artikkelin Node.js:n syvimmästä olemuksesta toptal.com-verkkosivustolla. Viitattu 20.11.2016.

<https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>.

Client Side vs. Server Side. N.d. Artikkelin asiakasohjelmoinnin ja palvelinohjelmoinnin eroista. Viitattu 20.11.2016. <http://www.codeconquest.com/website/client-side-vs-server-side/>.

Cross Platform Development For Desktops: Choosing The Right Technology. 2015. Artikkelin monialustaisten sovelluksien kehittämisestä työpöytäympäristöön mobidev.biz-verkkosivustolla. Viitattu 21.11.2016. <https://mobidev.biz/blog/cross-platform-development-for-desktops-choosing-the-right-technology>.

Documentation. 2016. Firebase-pilvidataratkaisupalvelun dokumentaation etusivu. Viitattu 20.11.2016. <https://firebase.google.com/docs/>.

Eftaiha, D. 2012. An Introduction to Apache. Artikkelin palvelinsovelluksista tutstplus-sivustolla. Viitattu 20.11.2016. <https://code.tutsplus.com/tutorials/an-introduction-to-apache--net-25786>.

Ellingwood, J. 2015. Apache vs. Nginx: Practical considerations. Artikkelin Apachen ja Nginxin ideologisista poikkeavuuksista. Viitattu 20.11.2016.

<https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations>.

Excel 2016. N.d. Microsoft Excel-tilukkolaskentaohjelman esittelysivu Microsoftin kotisivuilla. Viitattu 20.11.2016. <https://products.office.com/fi-fi/excel>.

Gentz, M. 2016. NoSQL vs SQL. Artikkelin relaatiotietokantojen ja epärelaatiotietokantojen eroista. Viitattu 20.11.2016.

<https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql>.

gkm. N.d. GKM-Node.js-moduulin sivu npm-paketinhallintajärjestelmän verkkosivuilla. Viitattu 3.12.2016. <https://www.npmjs.com/package/gkm>.

Introduction To Knockout. N.d. Knockout-sovelluskehityksen esittely tämän itsensä kotisivuilla. Viitattu 21.11.2016.

<http://knockoutjs.com/documentation/introduction.html>.

Importance of Design In Software Development. 2013. Ohjelmistosuunnittelun tärkeydestä kertova artikkeli mobidev-verkkosivustolla. Viitattu 3.12.2016. [https://mobidev.biz/blog/importance\\_of\\_design\\_in\\_software\\_development](https://mobidev.biz/blog/importance_of_design_in_software_development).

Introduction To Laravel Framework. N.d. Johdatteleva artikkeli Laravel-sovelluskehukseen. Viitattu 20.11.2016. <http://laravelbook.com/laravel-introduction/>.

IOGraphica. 2016. IOGraphica-hiirenanalysointisovelluksen kotisivut. Viitattu 20.11.2016. <http://iographica.com>.

Juslin, J. 1999. Olioarkkitehtuuriseminaarin seminaariesitelmä. Viitattu 8.12.2016. <http://zds.iki.fi/zds/tekstit/kehykset.shtml>.

MIT License. N.d. MIT-ohjelmistolisenssi selitettynä yksinkertaisesti tldrLegal-verkkosivustolla. Viitattu 3.12.2016. <https://tldrlegal.com/license/mit-license>.

Mousotron. 2016. Mousotron-hiirianalysointisovelluksen kotisivut. Viitattu 20.11.2016. <http://www.blacksunsoftware.com/mousotron.html>.

Nagele, C. N.d. Versiohallinnan yleisesittely. Viitattu 3.12.2016. <http://guides.beanstalkapp.com/version-control/intro-to-version-control.html>.

Paakki, J. 2016. Ohjelmistojen vaatimusmäärittely. Vaatimusmäärittelyn faktat ja vaihtoehdot, 3. Viitattu 20.11.2016. <https://www.cs.helsinki.fi/u/paakki/Vaatimus-11-Luentokalvot-1.pdf>.

Pautasso, C. 2009. REST-rajapintasuunnittelusta kertova Luganon yliopiston luentomoniste. [http://www.jopera.org/files/SOA2009-REST-Patterns\\_0.pdf](http://www.jopera.org/files/SOA2009-REST-Patterns_0.pdf).

Plot.ly. 2016. Plot.ly bisnesdatan analysointiin suunnatun sovelluksen kotisivut. Viitattu 20.11.2016. <http://plot.ly>.

Pricing. 2016. Firebase-pilvidataratkaisupalvelun hinnoittelu. Viitattu 20.11.2016. <https://firebase.google.com/pricing/>.

Pstatz. 2010. PHP Tutorial For Beginners. Artikkelin PHP:n syvimmästä olemuksesta wired.com-verkkosivustolla. Viitattu 20.11.2016. [https://www.wired.com/2010/02/php\\_tutorial\\_for\\_beginners/](https://www.wired.com/2010/02/php_tutorial_for_beginners/).

Symfony explained to a Developer. N.d. Symfony-sovelluskehystä avaava artikkeli Symfony:n kotisivuilla. Viitattu 20.11.2016. <http://symfony.com/explained-to-a-developer>.

Time Tracking Software. 2016. TimeAnalyzer-analysointisovelluksen kotisivut. Viitattu 20.11.2016 <http://www.neuber.com/timeanalyzer/>.

West, M. 2013. An Introduction to WebSockets. Viitattu 2.12.2016. <http://blog.teamtreehouse.com/an-introduction-to-websockets>.

What Is React?. N.d. ReactJS-sovelluskehysten esittely tämän itsensä kotisivuilla. Viitattu 21.11.2016. <https://facebook.github.io/react/tutorial/tutorial.html>.

XtraDB and InnoDB. 2016. XtraDB:n ja InnoDB:n dokumentaatiota MariaDB:n näkökulmasta. Viitattu 20.11.2016. <https://mariadb.com/kb/en/mariadb/xtradb-and-innodb/>.

Yank, K. 2001. Which Server-Side Language Is Right For You? Artikkelin palvelinohjelmointikielten eroista. Viitattu 20.11.2016.  
<https://www.sitepoint.com/server-side-language-right/>.

Yii Framework. 2016. Yii PHP-sovelluskehysten kotisivut. Viitattu 20.11.2016.  
<http://www.yiiframework.com/>.

# Liitteet

## Liite 1. Kuvakaappauksia web-selainsovelluksen liittymistä

Hiilevi

atte  
hiilevi@hiilevi.xyz

Diagnostiikkatyökalut

- Dashboard
- Trendikuvaaja
- PC-inspektori
- Sovellukset
- Kokonaiskäyttöaika raportti
- Käyttöaste kellonajoittain
- Näppäimistö
- Yhteenveto
- Kuomituskuvaaja
- Hiiri

Hallintatyökalut

- Rajapinnat
- Datan hallinta
- Lisää, poista ja muokkaa dataa
- Järjestelmän asetukset
- Käyttäjät
- Käyttöoikeudet

Rajapinnat

Nimi

☐ Kotisää

☐ Ulkosää

☒ Elintarvikkeet

Nimi

Elintarvikkeet

Tyyppi

JSON

URL

ei\_urlia

Suoritusajaväli (min)

0 1

Kentät

<input type="checkbox"/>	Nimi	Kenttä	Tyyppi	Oletuslaskenta	Yksikkö
<input type="checkbox"/>	Banaani	banana	Kokonaisluku	Summa	kpl
<input type="checkbox"/>	Kahvi	coffee	Kokonaisluku	Summa	kpl

PERUUTA TALLENNA

Hiilevi

atte  
hiilevi@hiilevi.xyz

Diagnostiikkatyökalut

- Dashboard
- Trendikuvaaja
- PC-inspektori
- Sovellukset
- Kokonaiskäyttöaika raportti
- Käyttöaste kellonajoittain
- Näppäimistö
- Yhteenveto
- Kuomituskuvaaja
- Hiiri

Hallintatyökalut

- Rajapinnat
- Datan hallinta
- Lisää, poista ja muokkaa dataa
- Järjestelmän asetukset

Rajapinta

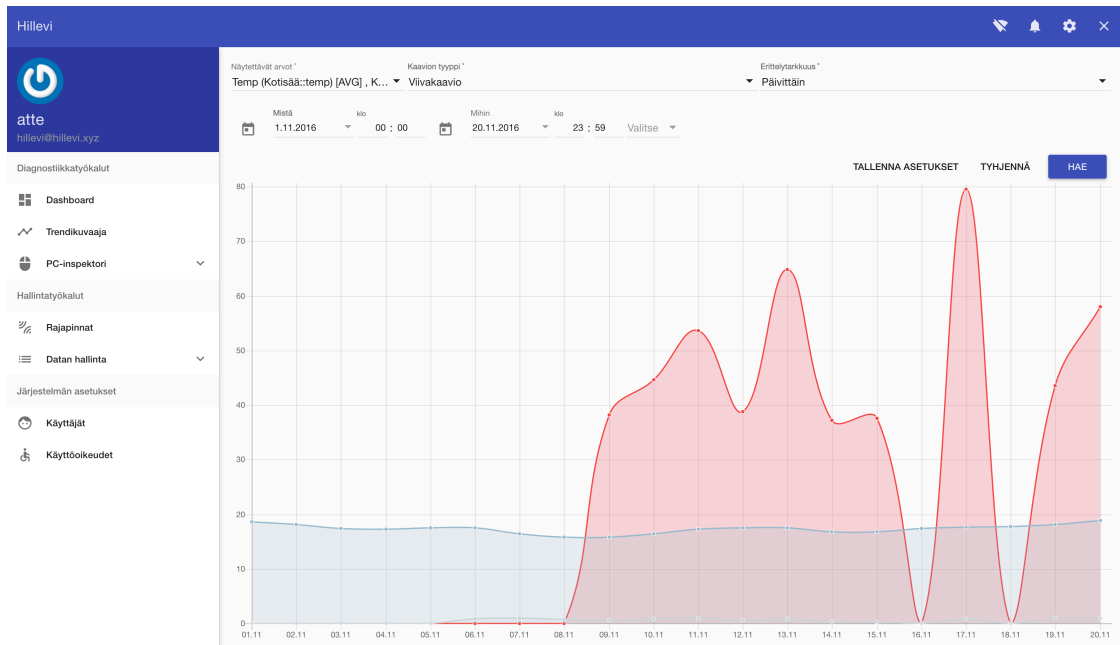
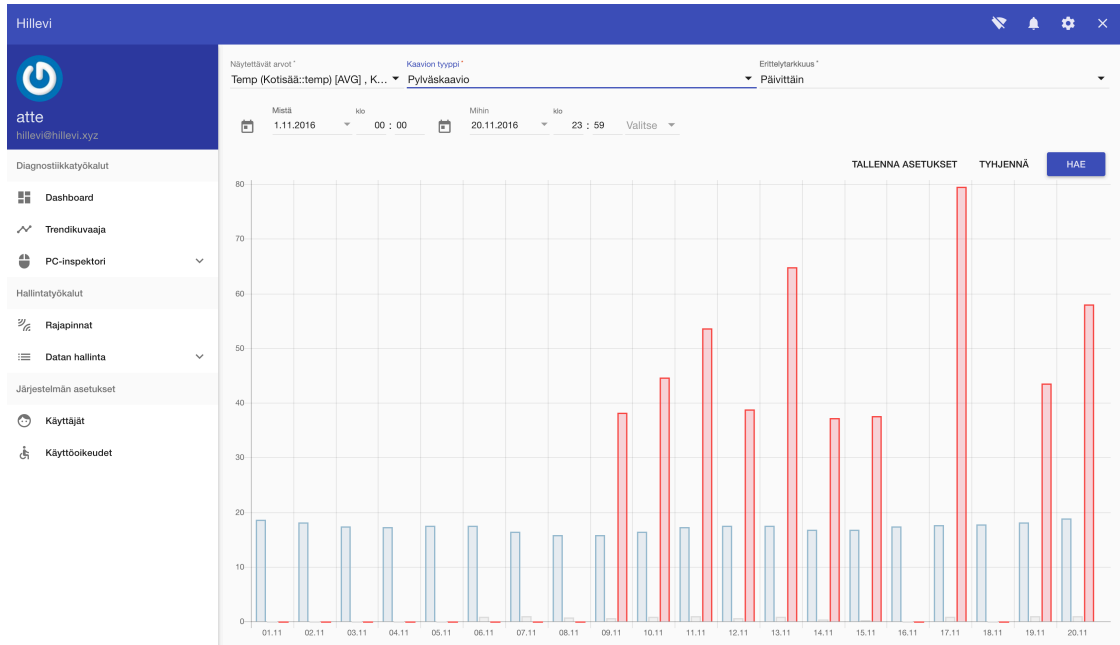
Kotisää

Sarakkeet

Temp (temp)

<input type="checkbox"/>	Id	Aikaleima	temp
<input type="checkbox"/>	1	2016-10-24 23:42:52	19.98
<input type="checkbox"/>	2	2016-10-24 23:43:52	19.70
<input type="checkbox"/>	3	2016-10-24 23:44:52	19.53
<input type="checkbox"/>	4	2016-10-24 23:45:52	19.43
<input type="checkbox"/>	5	2016-10-24 23:46:52	19.43
<input type="checkbox"/>	6	2016-10-24 23:47:52	19.42
<input type="checkbox"/>	7	2016-10-24 23:48:52	19.39
<input type="checkbox"/>	8	2016-10-24 23:49:52	19.40
<input type="checkbox"/>	9	2016-10-24 23:50:52	19.50
<input type="checkbox"/>	10	2016-10-24 23:51:53	19.47

Page: 1 Rows per page: 10 1 - 10 of 43493



Hillevi

atle  
hillevi@hillevi.xyz

Diagnostiikkatyökalut

- Dashboard
- Trendikuvaaja
- PC-inspektori

Sovellukset

- Kokonaiskäyttöaika
- Käyttöaste kellonajoin
- Näppäimistö
- Hiiri

Hallintatyökalut

...

Sovellusten kokonaisaktiivisuus

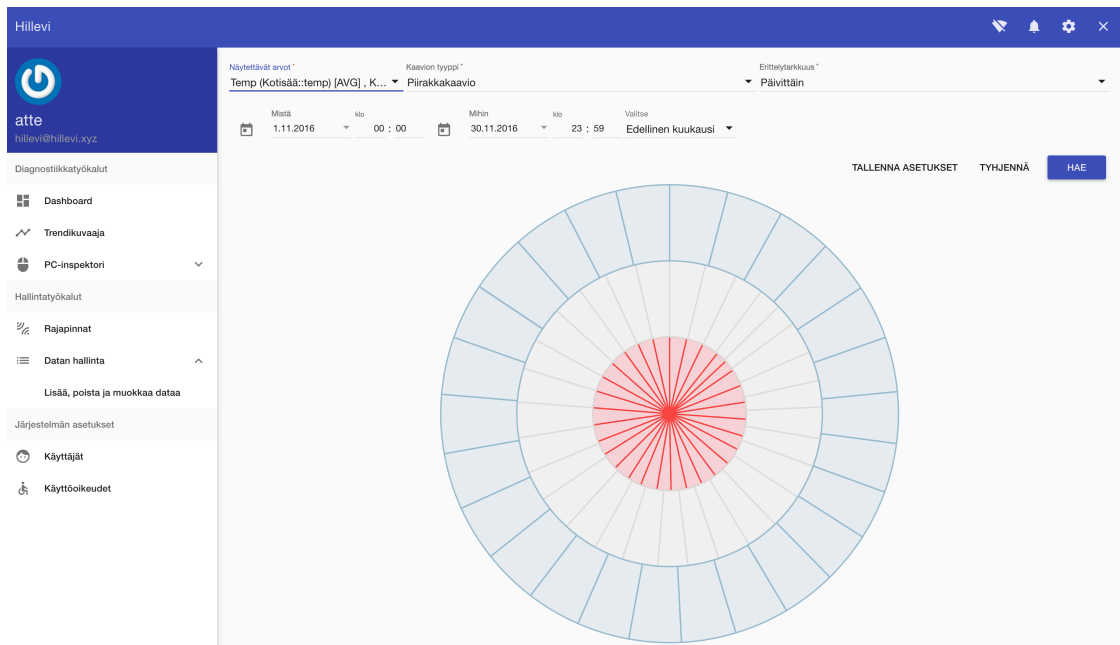
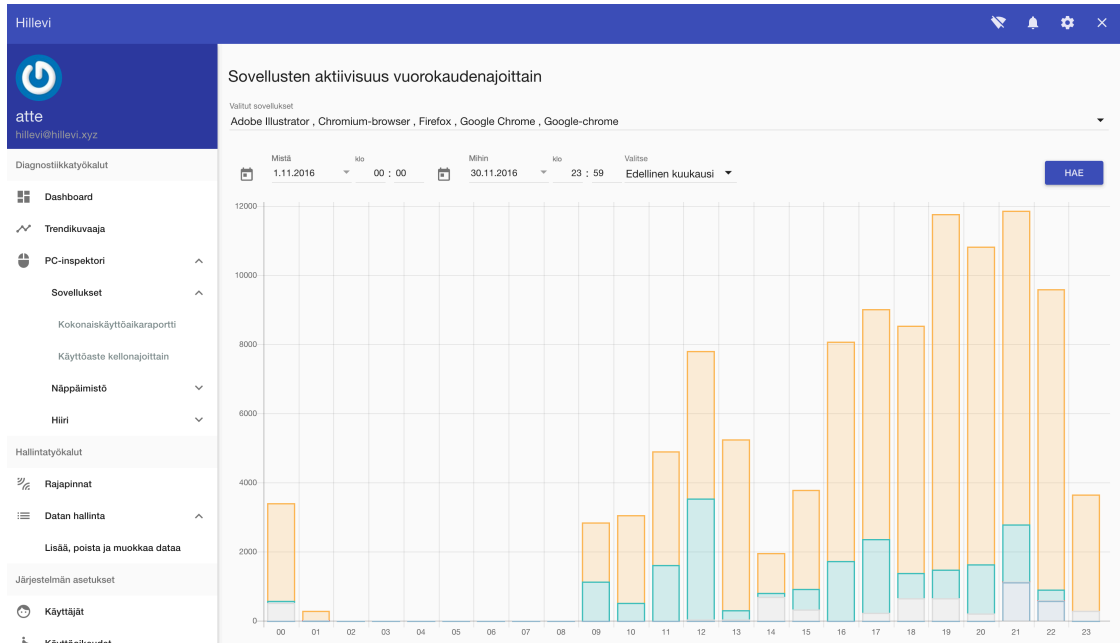
Mistä: 21.11.2016 klo 00:00 Mihin: 27.11.2016 klo 23:59 Valitse

Edellinen viikko

HAE

Nimi	Aktiivinen aika
Microsoft Word	2h, 32min 52s
phpstorm	23min 4s
Google Chrome	17min 45s
Terminal	16min 32s
mejs	3s
Finder	3s

Page: 1 Rows per page: 10 1 - 6 of 6



Hillevi

atle  
hillevi@hillevi.xyz

Diagnostiikkatyökalut


- Dashboard
- Trendikuvaaja
- PC-inspektori
- Sovellukset
- Kokonaiskäyttöaikaraportti
- Käyttöaste kellonajoin
- Näppäimistö
- Hiiri
- Hallintatyökalut
- Rajapinnat
- Datan hallinta
- Lisää, poista ja muokkaa dataa
- Järjestelmän asetukset
- Käyttäjät
- Käyttöoikeudet

### Käyttäjät

+ ↺

Atte Tarvainen	hillevi@hillevi.xyz
Matti Meikäläinen	matti.meikalainen@hillevi.xyz
Jari Jarppa	jari.jarppa@hillevi.xyz
Maija Meikäläinen	majja.meikalainen@hillevi.xyz
Jari Kari	jari.kari@hillevi.xyz

Hillevi

hillevi@hillevi.xyz

Diagnostiikkatyökalut

Dashboard

Trendikuvaaja

PC-inspektori

Sovellukset

Kokonaiskäyttöaikaraportti

Käyttöaste kellonajoltain

Näppäimistö

Yhteenvedo

Kuomituskuvaaja

Hiiri

Käyttäjän perustiedot

Käyttäjä tunnus

masa

Etinimi

Matti

Sukunimi

Meikäläinen

Sähköpostiosoite


matti.meikalainen@hillevi.xyz

Toiminnot

PERUUTA

TALLENNA

Hillevi

hillevi@hillevi.xyz

Diagnostiikkatyökalut

Dashboard

Trendikuvaaja

PC-inspektori

Sovellukset

Kokonaiskäyttöaikaraportti

Käyttöaste kellonajoltain

Näppäimistö

Yhteenvedo

Kuomituskuvaaja

Hiiri

Hallintatyökalut

Rajapinnat

Datan hallinta

Lisää, poista ja muokkaa dataa

Järjestelmän asetukset

Käyttäjät

Käyttöoikeudet

Valitut käyttäjät

Maija Meikäläinen , Matti Meikäläinen , Atte Tarvainen

HAE

Käyttöoikeus	Atte Tarvainen	Matti Meikäläinen	Maija Meikäläinen
admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
appsetting.permissions.read	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
appsetting.permissions.write	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
appsetting.users.delete	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
appsetting.users.read	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
appsetting.users.write	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
cron.read	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
cron.write	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
dashboard	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
import.execute	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Page: 1

Rows per page: 10

1 - 10 of 19

TALLENNA