Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

Hyvinvointiteknologia

2018

Juhani Koskinen & Torsti Paavilainen

OHJELMAKIRJASTO KÄYTTÄJÄN OPASTAMISEEN VERKKOPALVELUSSA

– sähköistä asiakaspalvelua itsepalveluna

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2018 | 55 sivua

Juhani Koskinen & Torsti Paavilainen

OHJELMAKIRJASTO KÄYTTÄJÄN OPASTAMISEEN VERKKOPALVELUSSA

- sähköistä asiakaspalvelua itsepalveluna

Opinnäytetyön tavoitteena oli kehittää verkkosovelluksen käyttäjän opastamiseen käytettävän ohjelman prototyyppi. Kehitettyä prototyyppiä käytetään haluttujen elementtien korostamiseen verkkosovelluksessa. Prototyyppi toteutettiin yhteistyössä turkulaisen lääketeollisuuden ja terveydenhuollon toimijoiden käyttöön verkkosovelluksia tuottavan yrityksen Medtime Ohjelmistotalon kanssa.

Prototyyppi suunniteltiin toimimaan mahdollisimman yleisellä tasolla minkä tahansa verkkosovelluksen tai -sivun kanssa. Kehitystyössä käytettiin ES6 spesifikaation mukaista JavaScriptiä.

Prototyypin toiminnallisuuden ja rakenteellisen laadun varmistamiseksi työssä tarkasteltiin modulaarisen ohjelmarakenteen hyötyjä ja ohjelmakirjaston ominaisuuksia. Lisäksi selvitettiin JavaScriptillä kirjoitetun kirjaston erityispiirteitä. Esimerkkinä käytettiin jQueryä, joka on täysin JavaScript-ohjelmointikielellä toteutettu modulaarinen ohjelmakirjasto.

Sisällön ja käytettävyyden laadun vaatimuksia pyrittiin selvittämään asiakaspalvelun ja itsepalvelun suhdetta tarkastelemalla. Työssä tutkittiin asiakaspalvelun ja itsepalvelun piirteitä, sekä sitä miten ne vaikuttavat verkkosovelluksen käyttäjän opastamiseen käytettävän ohjelman toteuttamiseen.

Valmis prototyyppi sovitettiin toimimaan Medtime Ohjelmistotalon kehittämän työvuorosuunnitteluun käytettävän ohjelmiston, Medtime Pro Työvuorot 2:n kanssa. Koska prototyyppiä käytetään työvuorosuunnitteluohjelmiston toiminnallisuuksien esittelyyn, työtä varten kartoitettiin kaikki ohjelmiston toiminnallisuudet. Jokaiselle toiminnolle tehtiin myös lyhyt kuvaus.

Medtime Ohjelmistotalo tulee jatkamaan ohjelman kehitystä perustaen työnsä tehtyyn prototyyppiin.

ASIASANAT:

ohjelmakirjasto, moduulit, verkko-ohjelmointi, JavaScript, käytettävyys, asiakaspalvelu, itsepalvelu

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and communication technology

2018 | 55 pages

Juhani Koskinen & Torsti Paavilainen

SOFTWARE LIBRARY FOR USER GUIDANCE WITHIN A WEB SERVICE

- electronic customer service as a self-service

The aim of the thesis was to develop a prototype of a program used for user guidance within a web application. The developed prototype is used to highlight elements in a web application. The prototype was implemented in collaboration with Medtime Software House, a Turku-based company that produces software for the pharmaceutical and health care sectors.

The prototype was planned to generally work with any web application or web site. The development was implemented with JavaScript in accordance to the ES6 specification.

The benefits of modular design and characteristics of a software library were examined to secure the functionality and structure of the prototype. The special characteristics of a JavaScript written software library were examined through jQuery, a modular software library implemented fully in JavaScript.

The relation between customer service and self-service was examined in an attempt to ensure the quality in the content and the functionality of the prototype. The characteristics of customer service and self-service were studied to find out how they would influence the development of a software used for user guidance within a web application.

The finished prototype was connected to work with Medtime Pro Shifts 2, a software used to plan work shifts being developed by Medtime Software House. Because the prototype is used as a tour for the different functionalities within the shift planning application, a mapping of all the applications functionalities was carried out as part of this thesis. In addition, a short description was compiled for each functionality.

Medtime Software House will continue the development of the user guidance program basing their work on the prototype.

KEYWORDS:

software library, modules, web programming, JavaScript, usability, customer service, self-service

SISÄLTÖ

KUVAT

KUVIOT

1 JOHDANTO

Ohjelmiston käytettävyyden kannalta on tärkeää, että ohjelmistotuotannossa on huomioitu mahdollisimman käyttäjälähtöiset toimintatavat. Ohjelmistojen kehittäjiä toisinaan syytetään liian teknologiakeskeisistä toimintatavoista, joiden sanotaan tekevän valmiista tuotteista vaikeakäyttöisiä ja hankalasti lähestyttäviä. Näihin haasteisiin pystyy vastaamaan kiinnittämällä ohjelmiston suunnittelu- ja toteutusvaiheissa erityistä huomiota käyttöliittymäsuunnittelun laatuun sekä riittävään käyttäjän opastamiseen ohjelmiston sisällä.

Asiakaspalvelu on perinteisimmillään kahden ihmisen, asiakkaan ja asiakaspalvelijan, välistä kanssakäymistä, jossa asiakaspalvelija asiantuntemuksellaan pyrkii vaikuttamaan asiakkaan ostopäätökseen tai tyytyväisyyteen jo ostamaansa palveluun tai tuotteeseen. Sosiaalinen ulottuvuus on tärkeä osa asiakaspalvelua. Ammattitaitoinen asiakaspalvelija osaa jakaa miellyttäviä kokemuksia asiakkailleen sekä antaa halutunlaisia mielikuvia myymästään tuotteesta tai palvelusta.

Verkossa tapahtuva sähköinen asiakaspalvelukin on useimmiten Chat-palveluiden, sähköpostin tai nousevassa määrin sosiaalisen median välityksellä tapahtuvaa kahden ihmisen välistä kanssakäymistä. Tarjolla on kuitenkin myös malleja, joissa asiakas toteuttaa asiakaspalvelua itsepalveluna. Tällöin asiakaspalvelutapahtuman ainoa ihmistoimija on asiakas itse. Tällaisia palveluita ovat esimerkiksi automatisoidut puhelinvaihteet tai verkkosivuille sijoitettavat FAQ:t, chatbotit tai interaktiiviset vianetsintä- tai opastusjärjestelmät. Myös asiakkaidenväliseen vertaistukeen perustuvat keskustelufoorumit ovat tapa vahvistaa yrityksen asiakaspalvelua.

Tämän opinnäytetyön tekninen osio on toteutettu yhteistyössä turkulaisen lääketeollisuuden ja terveydenhuollon toimijoiden käyttöön ohjelmistoja tuottavan Medtime Ohjelmistotalon kanssa. Opinnäytetyön lopputuotoksena on yhden yrityksen ohjelmiston toimintojen kartoitus sekä yrityksen eri ohjelmistojen kanssa käytettävän, tässä työssä määrittämiemme rajaehtojen mukaan toteutetun käyttäjää opastavan ohjelman prototyyppi. Työssä liitteenä oleva prototyypin lähdekoodi sekä prototyypin toimintaa koskeva luku on salattu toimeksiantajan pyynnöstä.

Opinnäytetyössä käydään yleisellä tasolla läpi mitä käyttäjää opastavan, useamman eri web-applikaation kanssa käyttöön soveltuvan itsenäisen ohjelman toteuttaminen vaatii. Opinnäytetyön teoriaosuudessa käsitellään laadukkaan, modulaarisuuteen perustuvan

ohjelmakirjaston ominaisuuksia ja perusrakenteita, sekä näiden tuomia etuja ja haasteita. Työssä käsitellään ohjelmakirjastoja yleisellä tasolla, sekä perehdytään tarkemmin JavaScript-ohjelmointikielellä toteutettavan ja selaimella käytettävän web-sovelluksen kanssa käytettäväksi tarkoitetun ohjelmakirjaston erityispiirteisiin.

Lisäksi työssä selvitetään asiakaspalvelun peruskäsitteitä, mitä on sähköinen asiakaspalvelu, sekä miten sitä voisi toteuttaa onnistuneesti itsepalveluna verkossa. Työhön on eri tutkimusten pohjalta koottu huomioita ihmisten asenteista ja odotuksista sekä itsepalveluun että sähköiseen asiointiin liittyen.

2 OHJELMAKIRJASTO

2.1 Ohjelmakirjaston määritelmä

Ohjelmakirjasto on sovelluksista ja paketeista koostuva kokoelma, joka on valjastettu yleiseen käyttöön ohjelmiston kehitysympäristössä. Ohjelmakirjastot saattavat sisältää kehitystä helpottavia työkaluja, ohjelmistopaketteja tai ohjelmointikielen kääntäjän. Yleensä ohjelmakirjastosta tarvitaan ainoastaan viite, jotta kehittäjä saa sen käyttöön omaan kehitysympäristöönsä. (Encyclopedia.com 2004.)

Techopedia (2018) määrittää ohjelmakirjaston sarjaksi dataa ja koodia, joita käytetään sovellusten ja ohjelmistojen tekoon. Ohjelmakirjastot on suunniteltu auttamaan niin ohjelmistokehittäjää kuin myös ohjelmistokielen kääntäjää ohjelmiston rakentamisessa ja suorittamisessa. Yleensä kehittäjä lisää kirjaston ohjelmistoonsa saadakseen enemmän toiminnallisuutta tai automatisoidakseen prosessin kirjoittamatta koodia, joka olisi vastuussa siitä. Kaikkia ohjelmakirjaston funktioita pystyy kutsumaan ohjelmiston rungosta ilman funktioiden erillistä määrittämistä. (Techopedia 2018.)

FOLDOC (1998) tietojenkäsittelyn ilmaisen sanakirjan mukaan ohjelmakirjastot ovat yksi varhaisimmista koodin uudelleenkäytön muodoista. Ohjelmakirjastoja tarjoaa yleensä käyttöjärjestelmät, kehitysympäristöt, sekä ohjelmistokehittäjät ja niiden käyttökohteita ovat monet eri ohjelmistot. (FOLDOC 1998.)

Rob Tougherin (2002) mukaan hyvin kirjoitettu ohjelmakirjasto on helppokäyttöinen, toimii virheettömästi ja tuottaa yksityiskohtaisia virheilmoituksia. Ohjelmakirjaston helppokäyttöisyyteen vaikuttaa ohjelmiston monimutkaisuus, ohjelmoinnin johdonmukaisuus ja ohjelmiston käytön intuitiivisuus. Virheettömyys voidaan taata laajalla testaamisella, johon auttaa myös itse kirjoitetut testausohjelmat. On myös tärkeää, että ohjelmakirjasto kertoo käyttäjällensä aina, kun se ei pysty suorittamaan tehtäviään oikein. (Tougher 2002.)

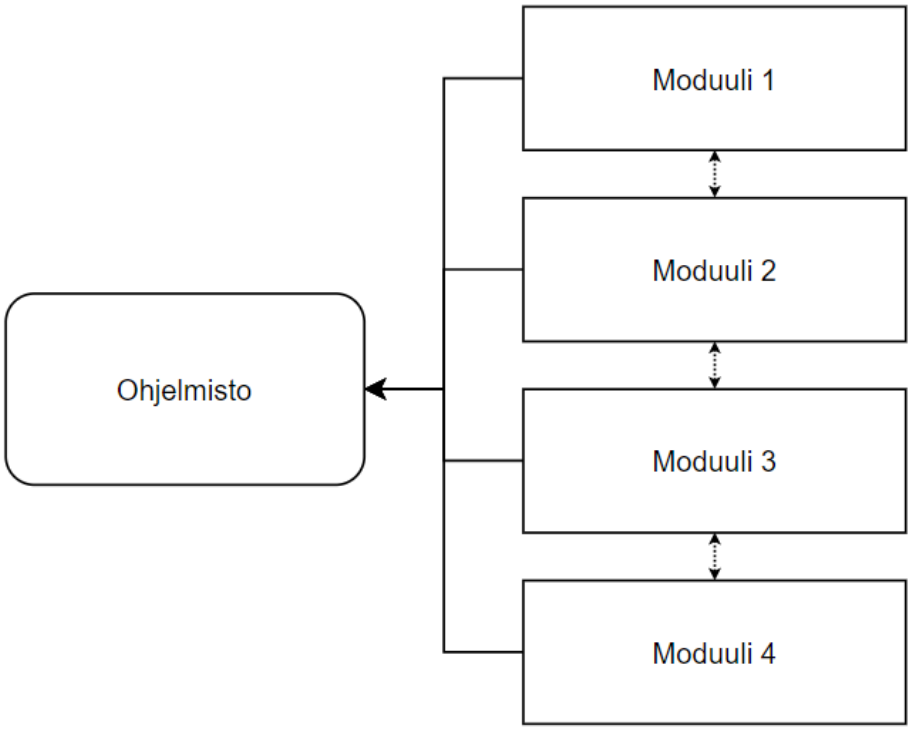
Kun kehittäjä käyttää perinteistä ohjelmakirjastoa, hallitsee hän itse sen toimintoja. Kehittäjä kutsuu itse ohjelmakirjastoa ja on vastuussa sen tuomista seuraamuksista. Ero perinteisen ohjelmakirjaston ja ohjelmistokehyksen välillä on, että kehittäjän sijaan ohjelmistokehys hallitsee ohjelmistoa ja hoitaa tarpeelliset toiminnallisuudet, mutta antaa

kehittäjälle kuitenkin mahdollisuuden välittää takaisinkutsufunktioita, joita kutsutaan prosessien päätteeksi. (Edwin 2014, 677.)

2.2 Modulaarisuus

Modulaarinen ohjelmisto on jaettu toiminnallisiin yksiköihin, joita kutsutaan moduuleiksi. Näiden moduulien ominaispiirre on, että ne voidaan yhdistää suurempaan ohjelmistokokonaisuuteen. Jokainen moduuli tarjoaa osan ohjelmiston kokonaistoiminnallisuudesta ja edustaa joukkoa siihen liittyviä huolenaiheita. Selain-pohjaisissa ohjelmistoissa moduuli voi esimerkiksi lisätä tai poistaa käyttöliittymän elementtejä, tai parantaa käyttöliittymän toiminnallisuutta tai yleistä käyttökokemusta.

Kehittäjät pystyvät rakentamaan ohjelmistoon uusia moduuleja, joilla ei ole riippuvuuksia jo olemassa oleviin moduuleihin. Ohjelmiston moduulit ovat itsenäisiä toisistaan, mutta ne pystyvät kuitenkin viestimään toistensa kanssa löyhästi yhdistettyinä (Kuvio 1) (Brumfield ym. 2011, 77).



Kuvio 1. Esimerkki modulaarisen ohjelmiston suhteista

2.2.1 Modulaarisuuden hyötyjä

Monimutkaisimpia ohjelmistoja ei yleensä kirjoiteta yhden henkilön toimesta, vaan ryhmässä. Ohjelmiston jakaminen moduuleihin mahdollistaa ohjelmiston eri osien kehityksen samanaikaisesti ryhmän jäsenten kesken. Tämä nopeuttaa huomattavasti ohjelmiston kehitystä.

Modernit ohjelmistot usein viivyttävät ohjelmiston komponenttien asetusten määrittämistä, kunnes ohjelmisto on otettu käyttöön. Modulaarinen lähestymistapa antaa mahdollisuuden toteuttaa tapoja, joilla ohjelmiston komponenttien asetukset voidaan määrittää ulkoisesti. Ilman ulkoista asetusten määrittämistä, tulee määrittämiseen käyttää kovakoodattuja komponenttien riippuvuuksia, joka johtaa koodiin, jota on vaikea testata ja ylläpitää. (Brumfield ym. 2011, 79.)

Modulaariset ohjelmistot voivat helpottaa ohjelmiston kehitystä, testausta, käyttöönottoa, sekä ohjelmiston laajentamista. Modulaarinen suunnittelu sisältää myös hyvin tunnettuja hyötyjä, jotka helpottavat ohjelmiston yksikkötestausta ja ylläpitoa ajan myötä. (Brumfield ym. 2011, 77.)

2.2.2 Modulaarisuuden vaatimukset

Bertrand Meyer (1997) määrittelee, että suunnittelumenetelmää voidaan kutsua modulaariseksi vasta kun se täyttää viisi olennaista vaatimusta. Nämä vaatimukset ovat hajotettavuus, koottavuus, ymmärrettävyys, jatkettavuus ja suojaus.

Hajotettavuudella tarkoitetaan sitä, että järjestelmä tulisi pystyä jakamaan pienempiin alijärjestelmiin ja näiden alijärjestelmien luominen tulisi pystyä jakamaan projektin työntekijöiden kesken. Tämä on vaikea tavoite, sillä se rajoittaa ohjelmiston riippuvuuksia alijärjestelmien kesken. Yhden alijärjestelmän kehityksen ei tulisi hidastaa toisen alijärjestelmän kehitystä.

Koottavuus käsittelee järjestelmän komponenttien uudelleenkäytettävyyttä. Ohjelmiston osien tulisi olla tarpeeksi autonomisia, jotta niitä pystyisi käyttämään ohjelmiston eri osissa ja eri yhteyksissä. Tämän vaatimuksen perimmäinen tarkoitus on muuttaa ohjelmiston suunnitteluprosessia niin, että ohjelmiston rakentaminen tapahtuu yhdistelemällä valmiiksi tehtyjä komponentteja.

Ymmärrettävä suunnittelumenetelmä auttaa tuottamaan ohjelmiston jonka koodia luettaessa lukija pystyy ymmärtämään jokaisen moduulin itsenäisenä kokonaisuutena ilman, että hänen tulee tietää muista moduuleista mitään. Tämän lähestymistavan käyttäminen helpottaa paljon ohjelmiston ylläpitoa, sillä tällöin kehittäjän tarvitsee keskittyä ainoastaan yhteen moduuliin kerralla.

Jatkettavuuden vaatimus on suoraan yhteydessä ohjelmiston skaalautuvuuteen. Sillä tarkoitetaan, että pienten muutosten tulisi vaikuttaa ainoastaan yksittäisiin moduuleihin, ei koko ohjelmiston rakenteeseen.

Suojauksen vaatimus täyttyy, jos ajonaikaisten poikkeavuuksien vaikutukset saadaan eristettyä yhteen tai mahdollisimman pieneen määrään moduuleja. Vaatimus ei kosketa virheiden välttämistä, vaan ainoastaan sen modulaarista puolta eli leviämistä muualle ohjelmistoon. (Meyer 1997, 39–46.)

2.3 Ohjelmakirjaston ominaisuuksia

2.3.1 Uudelleenkäytettävyys

FOLDOC (1985) määrittelee uudelleenkäytön olevan ohjelmistolle kehitetyn koodin käyttämistä toisessa ohjelmistossa. Tavanomaisesti tämä saavutetaan käyttämällä ohjelmakirjastoja. (FOLDOC, 1985.)

Tietojenkäsittelyn olennaiset tietorakenteet, (joukot, listat, puut, pinot...) ja niihin liittyvät algoritmit (lajittelu, hakeminen, läpikäyminen...) ovat läsnä kaikkialla ohjelmistokehityksessä. Tavanomaisissa ohjelmistokehityksen lähestymistavoissa, ohjelmistokehittäjät tekevät usein näistä omia toteutuksiaan. Tämä voi kuitenkin olla vahingollista ohjelmiston laadulle. On epätodennäköistä, että tietorakenteen toteuttanut yksittäinen kehittäjä saavuttaisi optimaalisen toimintavarmuuden ja tehokkuuden.

Tästä syystä olioperustaiset kehitysympäristöt tarjoavat yleensä kattavia uudelleenkäytettäviä ohjelmakirjastoja, sekä myös tapoja, joilla kirjastoja pystyy luomaan lisää. Näiden kirjastojen uudelleenkäytettävien luokkien tulisi ainakin kattaa yleisimmin käytetyt algoritmit ja tietorakenteet. (Meyer 1997, 33.)

2.3.2 Skaalautuvuus

Tietotekniikassa skaalautuvuus voi tarkoittaa kahta asiaa. Se voi tarkoittaa ohjelmiston tai tuotteen kykyä säilyttää toimintakykynsä, kun ohjelmiston kokoa muutetaan vastaamaan uusiin tarpeisiin. Tavallisesti nämä muutokset ovat ohjelmistoa tai tuotetta laajentavia. Koon muutos voi tarkoittaa joko itse tuotteen muutoksia (laitteistomuutokset) tai sen siirtämistä uuteen asiayhteyteen (esim. uusi käyttöjärjestelmä).

Skaalautuvuus voi tarkoittaa hyvän toiminnallisuuden lisäksi myös sitä, miten tehokkaasti tuote tai ohjelmisto pystyy hyödyntämään uuden laajennetun toimintaympäristön. Ohjelmisto on skaalautuva esimerkiksi silloin, kun se pystyy uuden käyttöjärjestelmän asennuksen myötä käyttämään uuden ja laajemman käyttöjärjestelmän tuomat edut hyväkseen (SearchDataCenter 2006).

Skaalautuva ohjelmisto viittaa yleensä yritysten ohjelmistoihin, jotka pystyvät mukautumaan kasvaviin käyttäjä- ja tietomääriin. Esimerkiksi skaalautuvan tietokannan hallintajärjestelmän tulee pystyä tehokkaasti laajentumaan, kun tietoa lisätään tietokantaan. Ohjelmiston tulee siis pystyä laajentumaan sitä mukaa, kun ohjelmiston käyttö kasvaa. Tämä tarkoittaa, että skaalautuva ohjelmisto vie vähän resursseja pienissä käyttötapauksissa, mutta käytön kasvaessa se pystyy mukautumaan muuttuviin tarpeisiin (TechTerms 2011).

2.3.3 Staattiset ja jaetut kirjastot

Staattinen ohjelmakirjasto on linkitetty suoraan ohjelmiston lopulliseen ajotiedostoon. Tällaiset kirjastot pysyvät muuttumattomina, kunnes ohjelmisto käännetään uudestaan. Tämä on suorin tapa käyttää kirjastoa, koska lopullinen ajotiedosto ei sisällä riippuvuuksia ulkoisiin ohjelmakirjaston tiedostoihin.

Staattisten ohjelmakirjastojen huonoja puolia on kuitenkin sen päivittämisen hankaluus ja se, että se sisällytetään ohjelmistojen ajotiedostoihin. Staattisen kirjaston päivittäminen vaatii koko ohjelmiston uudelleenkääntämisen. Tämän lisäksi jokainen ohjelmakirjastoa käyttävä ohjelmisto sisältää oman kopionsa kirjastosta ajotiedostossaan.

Toinen kirjastotyyppi on jaetut kirjastot. Nämä kirjastot ladataan dynaamisesti ajonaikana ohjelmistoille, jotka niitä tarvitsevat. Ohjelmistot käyttävät osoittimia, jotka kertovat ohjelmistolle mitä kirjastoja missäkin osassa ohjelmistoa tarvitaan. Kirjasto ladataan muistiin ja suoritetaan vasta kun sitä tarvitaan. Jos kirjasto on ladattu jo toisessa ohjelmistossa, pystyvät ohjelmistot jakamaan kirjaston koodin. Näin pystytään säästämään paljon resursseja, varsinkin jos kyseessä on usein käytetty kirjasto (Wienand 2004).

2.4 Ohjelmakirjaston esimerkkinä jQuery

2.4.1 Ominaisuudet

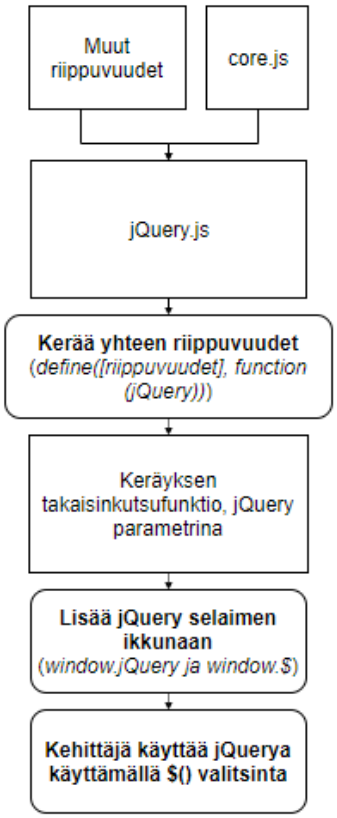
JQuery on avoimen lähdekoodin JavaScript-ohjelmakirjasto, joka huomioi DOM:n kanssa työskentelemisen haasteet eri selaimilla. Sillä on yksinkertainen ohjelmistorajapinta, joka voidaan jakaa kahteen eri kutsumiskäytäntöön: funktioihin ja metodeihin. Funktioita kutsutaan suoraan jQuery-olion kautta, kuten esimerkiksi ”$.extend(olio1, olio2)”. Metodeja kutsutaan paketoidusta joukosta, joka on jQueryn valitsimen kyselyn tulos. Tätä valitsinta käytetään elementtien valitsemiseen DOM:sta. JQuery tarjoaa myös ominaisuuksia tapahtumien herättämiselle ja niiden käsittelemiselle, sekä Ajax-kutsujen tekemiselle ja niiden palauttaman tiedon prosessoinnille. (Brumfield ym. 2011, 6.)

JQueryn parhaimpia erityspiirteitä on sen laajennusmahdollisuudet. Suosituin tapa laajentaa jQuerya on sen liitännäiset. Monet jQueryn nykyisistä ominaisuuksista ovat alkuperältään kirjaston ulkopuolisia liitännäisiä, jotka myöhemmin sulautettiin osaksi ohjelmakirjastoa. Koska liitännäiset toimivat samalla tavalla kuin muutkin jQueryn metodit, voi niitä kutsua eri elementeissä käyttäen jQueryn valitsinta. (Brumfield ym. 2011, 90.)

2.4.2 Toiminnallisuus

JQueryn lähdekoodiin tutustumalla on huomattavissa, että se ei ole kehitetty käyttäen vain yhtä tiedostoa, vaan se on jaettu moniin eri moduuleihin käyttäen AMD-moduuliformaattia. Jokainen moduuli sisältää listan riippuvuuksista, mikä helpottaa koodiviittauksien alkuperien löytämistä. Nämä moduulit käännetään yhdeksi JavaScript tiedostoksi käyttäen requireJS-moduuli- ja tiedostolataajaa.

Ohjelmakirjaston rakenne on nähtävissä kaikkien riippuvuuksien yhdistäjänä toimivasta ohjelmakirjaston aloitustiedostosta. Ohjelmakirjaston ydintiedosto ladataan ensin selaimen ikkunaan. Samalla luodaan jQueryn nimiavaruus, jota muut riippuvuudet muokkaavat tarpeen mukaan (Kuvio 2). Ydintiedosto on vastuussa myös jokaisen jQuery-olion prototyypin luomisesta.



Kuvio 2. jQueryn alustaminen.

Ydintiedostosta on nähtävissä myös, että jQuery käyttää paljon pieniä apumetodeja koodissaan. Näitä apumetodeja ei ole yhdistettynä yhteen tiettyyn olioon, vaan jokaiselle apumetodille on oma tiedostonsa. Apumetodia tarvittaessa joutuu kyseisen apumetodin tiedoston lisäämään riippuvuudeksi (McCormik 2015).

2.4.3 Liitännäiset

JQueryn liitännäiset ovat pohjimmiltaan funktioita, joita voidaan lisätä jQuery-olion prototyyppiin. Kun prototyyppiä laajennetaan, kaikki siitä tehdyt jQuery-oliot perivät prototyyppiin lisätyt funktiot. Liitännäisten ideana on, että niillä voidaan tarpeen mukaisesti lisätä toiminnallisuutta DOM-elementtien manipulointiin (jQuery 2015).

Rakhitha Nimesh (2013) määrittelee jQueryn liitännäisen itsenäisenä moduulina, joka lisää mukautettua toiminnallisuutta ohjelmistoon. Sen käyttöönottaminen tai käytöstä poistaminen ei vaikuta ohjelmiston ydintoimintoihin tai muihin liitännäisiin, mikä helpottaa ohjelmiston ylläpitoa. Liitännäiset myös vähentävät konflikteja funktioiden välillä, sillä kaikki liitännäiseen liittyvät funktiot luodaan oman nimiavaruuden sisälle.

JQuery-liitännäiset automatisoivat parametrien ja funktioiden laajentamisen liitännäisen sisällä. Liitännäiselle voi asettaa oletusparametreja ja -funktioita, jotka kehittäjä pystyy korvaamaan ajonaikana käyttäen laajennettuja arvoja ja funktioita. Tämä vähentää oletusparametrien ja pakollisten parametrien käsittelyyn menevää työtä.

Liitännäisten tehokkain ominaisuus on kuitenkin ketjuttaminen. Puhtaassa JavaScriptissä ketjuttaminen tapahtuisi funktion kutsumisella ja sen tulosten syöttämisellä toiseen funktiokutsuun. JQueryn funktion tuloksesta pystyy kuitenkin kutsumaan suoraan toista jQuery funktiota, mikä helpottaa toiminnallisuuden lisäämistä tai poistamista dynaamisesti (Nimesh 2013).

3 ASIAKASPALVELU SÄHKÖISENÄ ITSEPALVELUNA

3.1 Asiakaspalvelu

Asiakaspalvelu mielletään eri yritysten tai yhteisöjen tarjoamiksi palveluiksi. Ensisijaisesti on aina kyse nimenomaan palvelusta, ei niinkään tuotteesta. Palvelut yleensä määritelläänkin juuri suhteessa konkreettisiin tuotteisiin. Tässä mielessä palvelulle voidaan listata seuraavat neljä erityispiirrettä:

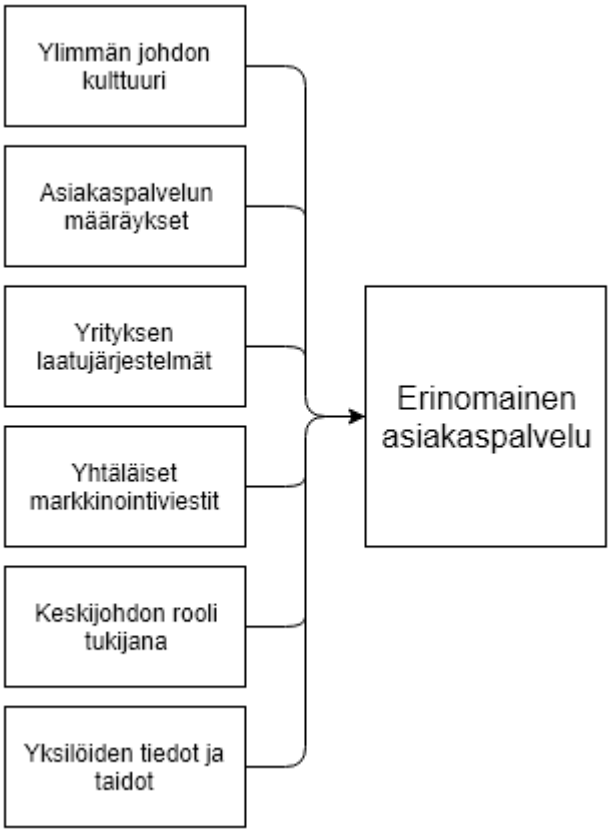
1. Palvelut ovat ainakin osaksi aineettomia.
2. Palvelut ovat prosesseja tai toimintasarjoja.
3. Palvelut kulutetaan – tai koetaan – samanaikaisesti, kun niitä tuotetaan.
4. Käyttäessään palveluja asiakas itse osallistuu palvelutapahtuman tuottamiseen.

Näiden erityispiirteiden lisäksi palvelu voidaan nähdä liiketoiminnan edistäjänä. Edistäminen on mahdollista, kun positiivisen toimintatavan ja laadun kautta saavutetaan positiivisia asiakaskokemuksia. Ollakseen laadukasta ja synnyttääkseen positiivisia kokemuksia, on palveluiden täytettävä tai ylitettävä palvelun käyttäjän odotukset. (Valvio 2010, 45–46.)

Puhuttaessa asiakaspalvelusta laadun lisäksi korostuu luotettavuus, nopeus ja yksilöllisyys. On erityisen arvokasta, että palvelu on yksilöllistä ja henkilökohtaista. Asiakaspalvelun tehtävä on saada asiakas tuntemaan, että hän saa juuri hänen ongelmiinsa tai tarpeisiinsa paneutuvaa palvelua. Koska asiakkaan tyytyväisyys on tärkeä asiakassuhteen ylläpitämiseksi, on empatia asiakaspalvelijan tärkein taito. (Lehtonen ym. 2002, 59–63.)

Newby (2000) tiivistää erinomaisen asiakaspalvelun olevan yrityskulttuurin ja asiakaspalvelijoiden tietojen ja taitojen yhdistelmä (Kuvio 3). Asiakaspalvelijalla tulee olla perinpohjainen tieto tarjottavista palveluista ja tuotteista, sekä asiakkaista joille nämä palvelut ja tuotteet ovat tarkoitettu. Hänen tulee tuntea organisaationsa toimintatavat ja sidosryhmät. Tarvittavia taitoja Newby luettelee olevan virheettömyys, kuuntelutaidot, rakentavuus, paineensietokyky, selkeä viestintä sekä oman työympäristönsä yhteisöllisyyden ylläpitämiseen tarvittavat taidot. Yksilön tietojen ja taitojen lisäksi hyvä asiakaspalvelu

heijastelee koko yrityksen kulttuuria. Yrityksen arvot sekä tietynlaiset toimintatavat ja johtamistyyli ohjaavat palvelukulttuurin syntymistä ja varmistavat sen säilymisen. (Newby 2000, 20–23.)

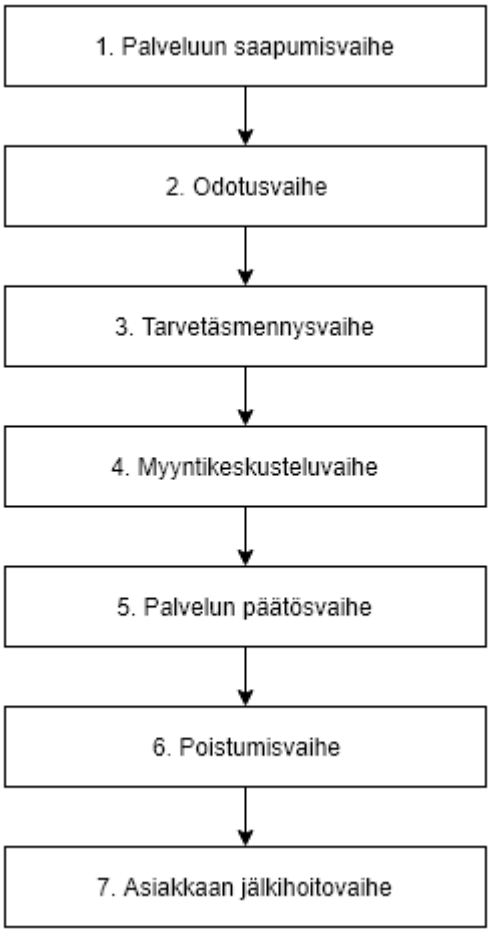


Kuvio 3. Erinomainen asiakaspalvelu.

Yksittäisen asiakaspalvelutapahtuman voi jakaa seitsemään vaiheeseen (Kuvio 4). Kokonaisuuden kannalta on tärkeää jokaisen vaiheen onnistuminen. Palveluorganisaation jokaisen asiakaspalvelijan tulee suoriutua hyvin jokaisessa palveluketjun vaiheessa sekä yksittäisten asiakaspalvelutapahtumien laadun tulee olla tasaisen korkea. (Lahtinen 1998, 69–70.)

Nykyaikaista, globalisaation ja korkealle kehittyneen viestintäteknologian kautta syntyneestä hyvinvoinnista hyötyvää kuluttajaa kutsutaan postmoderniksi kuluttajaksi. Tällaiselle kuluttajalle tuotteissa ja palveluissa materiaalista sisältöä tärkeämpiä ovat niiden luomat mielikuvat ja elämykset. Postmoderni kuluttaja ei myöskään halua sitoutua arvoihin, brändeihin tai yrityksiin, vaan on individualisti ja haluaa yksilöllisiä palveluita. (Lehtonen ym. 2002, 131.)

Edellä mainittujen lisäksi yksilöllisyyden tärkeyden nostaa esiin myös Lahtinen (2000, 48), Tuorila (2006, 10) sekä Legget (2016, 1; 2017, 1), joka nostaa yksilöllisyyden yhdeksi avainkohdaksi sekä vuoden 2016 että vuoden 2017 asiakaspalvelujen trendejä käsittelevissä raporteissaan.



Kuvio 4. Asiakaspalvelutapahtuman vaiheet.

3.2 Sähköinen asiakaspalvelu

Lähes jokaiselle yritykselle verkkosivut ovat tärkein asiakaspalvelun kanava (Lehtonen 2002, 127). Internetin merkitys asiakaspalvelun välineenä on noussut viime vuosina sitä mukaa kun kotitalouksien verkkoyhteydet ovat lisääntyneet. Nopea ja kattava mobiiliverkko ja mobiililaitteiden yleisyys kasvattavat sähköisten kanavien merkitystä entisestään. Tiedon hankkimista ei enää rajoita aukioloajat eikä tiedonhakijan tai tiedon fyysinen sijainti.

Palveluiden siirtyminen verkkoon luo kuitenkin myös uusia haasteita palveluntarjoajalle. Asiakkaat odottavat yhä nopeampaa vastausta kyselyihin. Turhia hidastelematon palveluprosessi ja nopea sekä jatkuvasti kehittyvä toiminta ovat asiakkaille yhä tärkeämpiä. Steven Van Belleghemin (2013) selvityksen mukaan palveluntarjoajaan sähköpostilla yhteydessä olevista kuluttajista 39 % odottaa saavansa vastauksen seuraavan neljän tunnin sisällä. Sosiaalisen median kautta yhteyden ottavista kuluttajista vastausta neljän tunnin sisällä odottaa saavansa jopa 55 %.

Erityisesti palvellessaan asiakkaita verkossa, tulisi yrityksen pohtia miten se pystyy luomaan ja ylläpitämään palvelumielikuvaa toiminnassaan. Palveluketju ei saa koskaan katketa itsestään, vaan aina joko asiakkaan tai palveluntarjoajan tulisi päättää koska palvelutapahtuma on ohi. (Valvio 2010, 23-24.)

Sähköisen asiakaspalvelun käytettävyyteen vaikuttaa myös palvelua käyttävän asiakkaan ikä. Valvio (2010, 34) kertoo eräästä tavasta jakaa ihmiset teknologian kannalta kolmeen sukupolveen:

- Vanhimmalla sukupolvella on runsaasti kokemuksia ja tietoa, mutta vähän taitoa hyödyntää teknologiaa.
- Keskimmäisellä sukupolvella on ammatillista osaamista melko paljon, ja se osaa käyttää sähköisiä välineitä työssään.
- Kolmas ja tässä ryhmittelyssä nuorin sukupolvi elää jo valmiiksi Web 2.0 -maailmassa, sillä heille tiedon jakaminen ja yhteisöllisyys ovat arkipäivää.

Koko nuorin sukupolvi ja osa keskimmäisestä voidaan laskea kuuluvan näiden ryhmien lisäksi myös postmoderni kuluttaja -ryhmään. Tämän ryhmän ulkopuolelle jää toisenlaiset tietotekniset taidot ja odotukset omaava vanhempi sukupolvi. Tuorila (2006, 11) mainitsee ettei suuri osa yli 50-vuotiaista ole kiinnostuneita opettelemaan uusia tietoteknisiä taitoja, vaan kokevat elämänkokemuksen tuoman tiedon ja osaamisen tärkeämmiksi pärjäämiselleen. Tutkimus, johon Tuorila viittaa, on kuitenkin vuodelta 2004. Tällä hetkellä tietotekniikan opettelusta kiinnostumattomien ikääntyvien ryhmään kuuluvien määrän voi olettaa pienentyneen.

3.3 Itsepalvelu

Itsepalvelusta puhuttaessa yleisimpiä esimerkkejä ovat pankkien verkkopalvelut, eri matkustusmuodoissa käytettävät e-liput tai aivan viime vuosina joissakin kaupoissa käyttöön otetut itsepalvelukassat. Asiakkaan palvellessa itse itseään on mahdollista vapauttaa osa asiakaspalvelijoiden työpanoksesta muihin tehtäviin. Kun pankit alkoivat veloittaa asiakkaitaan siitä, etteivät he maksaneet laskujaan verkossa itse, johti se protestointiin. Nykymaailma näyttää kuitenkin hyvin erilaiselta. Van Belleghem (2013) sanoo, että yritys joka ei tarjoa itsepalveluratkaisuja joutuu itse kärsimään seuraukset. Tällä hän vihjaa, että tarjoamalla itsepalveluja yritys pystyy vastaamaan asiakkaidensa odotuksiin, sekä luomaan itselleen selvän edun kilpailijoihinsa.

Kaikki edellä luetellut esimerkit ovat Internetin ja teknologisen kehityksen mahdollistamia itsepalvelun muotoja. Itsepalvelua toteutetaan kuitenkin myös verkon ulkopuolella. Auto tankataan nykyisin lähes poikkeuksetta itsepalveluna. Huoltoaseman työntekijän suorittamaa tankkausta tarjotaan enää vain harvoilla asemilla maksullisena lisäpalveluna. Itsepalvelutankkaaminenkin on kuitenkin Internetpalveluiden ja teknologian kehityksen myötä kehittynyt edelleen. Kun vielä joitakin vuosia sitten tankkauksen jälkeinen maksutapahtuma suoritettiin perinteisenä asiakaspalvelutapahtumana kahden ihmisen välillä, on nykyisin useimmissa tankkauspisteissä itsepalvelumaksuautomaatti. Automaattien myötä nykyään myös maksutapahtuma suoritetaan yhä useammin itsepalveluna.

Asiakaspalveluun liittyvä itsepalvelu on verkossa nykyisellään monimuotoista. Lehtonen ym. (2002, 129) mainitsevat 20/80-säännön: 20 prosentilla kaikesta tuotteeseen tai palveluun liittyvästä tiedosta pystyy ratkaisemaan 80 prosenttia tätä tuotetta tai palvelua koskevista ongelmista. Tämä huomioiden itsepalvelu todella on tehokas tapa vastata asiakkaiden tarpeisiin. Yksinkertaisimmillaan toteutus voi olla FAQ-osion toteuttaminen osaksi verkkopalvelua. Tähän osioon on parhaimmillaan koottuna vastaukset kaikkiin mainittuun 20 prosenttiin sisältyviin asiakasta mietityttäviin kysymyksiin.

Edellä mainittuihin verrattuna interaktiivisempia asiakaspalveluun liittyvän itsepalvelun muotoja ovat esimerkiksi keskustelufoorumit, virtuaaliset assistentit ja chatbotit. Keskustelufoorumeilla asiakaspalvelijan lisäksi käyttövinkkejä sekä vastauksia asiakkaiden kysymyksiin tuottavat asiakkaat itse. Virtuaalista assistenttia tai chatbottia varten tarvitaan

asiakkaan tuottamaa puhetta tai tekstiä tulkitseva ohjelma. Tällaisten ohjelmien on mahdollista toimia täysin itsenäisesti kahdenkeskisessä vuorovaikutuksessa suoraan asiakkaan kanssa.

Tuorila nostaa esille palvelujen yksilöllisyyden tarpeen sekä ikäluokkien toisistaan eroavat käyttötottumukset verkossa tapahtuvaan itsepalveluun liittyen. Ikääntyvät hoitavat asiansa mieluiten henkilökohtaisesti asiakaspalvelijan kanssa. Ikäryhmä kokee kasvokkain paikan päällä tehdyn asioinnin puhelinasiointiakin miellyttävämmäksi asiointitavaksi. Henkilökohtaisen ja yksilöllisen palvelun tarpeesta huolimatta itsepalvelut eivät suurten ikäluokkien ikääntyessä tule vähenemään. Tulevaisuuden ikääntyvät ovat enenemissä määrin jo oppineet ja tottuneet käyttämään Internetiä ja erilaisia automaatteja asioidensa hoitamiseen. (Tuorila 2006, 8.)

Myös Legget toteaa itsepalvelun olevan yksi isoista asiakaspalvelun nousevista trendeistä. Yritykset tulevat jatkossakin keräämään enemmän ja jäsentämään tarkemmin asiakkaisiin ja palvelutapahtumiin liittyvää tietoa. Tiedon pohjalta pyritään luomaan automaatioon perustuvia, asiakkaan tarpeet mahdollisimman tarkasti ennakoivia palvelujärjestelmiä. (Legget 2017, 3.)

Van Belleghemin (2013) raporttia varten tehtyyn kyselyyn osallistuneista kuluttajista 70 % odottaa palveluntarjoajaltaan jonkinlaista itsepalvelusovellusta kyselyiden ja valitusten käsittelyyn. Itsepalvelu on ainoa pitkän tähtäimen vaihtoehto, jolla yritys pystyy vastaamaan asiakkaidensa odotuksiin. Belleghem mainitsee kuitenkin henkilökohtaisen palvelun vaihtoehdon olevan ehdoton välttämättömyys, sillä asiakas ei joka tilanteessa pysty itsepalveluna saavuttamaan haluamaansa.

3.4 Asiakaspalvelutapahtuma itsepalveluna verkossa

Tarkastellaan verkossa toteutettua itsepalvelutapahtumaa Lahtisen ym. (1998) mukaan kuvioon 4 jaoteltujen palvelutapahtuman vaiheiden mukaisesti. Käytetään esimerkkinä verkkopalvelua, jossa käyttötukityyppinen asiakaspalvelu on toteutettu itsepalveluna interaktiivinen assistentti -moduulin avulla. Jatkossa tästä moduulista käytetään nimeä asiakaspalveluohjelma.

Ensimmäinen palvelutapahtuman vaihe on saapumisvaihe. Tässä vaiheessa tärkeää on hyvän ensivaikutelman luominen, palvelun käytön varmistaminen sekä palveluista tiedottaminen. Hyvän ensivaikutelman luomisen kannalta tärkeintä on palvelun selkeä ja

miellyttävä ulkoasu. Palvelussa käytetyn kielen tulee vastata käyttäjän odotuksia. Käyttäjällä tulee olla aina palvelun joka näkymässä selkeä pääsy asiakaspalveluohjelmaan. Hyvä käyttöliittymäsuunnittelu on myös käytön varmistamisen kannalta oleellista. Asiakaspalveluohjelman olemassaolosta tulee tiedottaa verkkopalvelun sisällä, etenkin jos verkkopalvelu on käyttäjälle vielä uusi. Verkkopalveluun palaavan, tai usein asiakaspalveluohjelmaa käyttävän käyttäjän voidaan kuitenkin olettaa jo olevan asiakaspalveluohjelmasta tietoinen. Näiden käyttäjien tapauksissa kannattanee tiedottamista tehdä harkitummin. Liiallinen mainostaminen saattaa herättää käyttäjässä negatiivisia tuntemuksia, sekä tehdä käyttöliittymästä epäselvemmän.

Saapumisvaiheen jälkeen seuraa odotusvaihe. Perinteisessä palvelutapahtumassa odotusvaiheen olemassaolo johtuu asiakaspalvelijoiden määrää suuremmasta asiakkaiden määrästä. Sähköisessä ympäristössä tämä odotusvaihe on huomattavasti pienempi, kun asiakaspalveluohjelman instansseja on käytännössä käynnissä yhtä monta kappaletta kuin verkkopalvelulla on käyttäjiä. Palvelun tai sen osien latautumista käyttäjä saattaa kuitenkin joutua odottamaan. Asiakaspalveluohjelman käyttökokemuksen tuleekin olla sujuva ja nopea.

Perinteisesti kolmannessa vaiheessa, tarvetäsmennysvaiheessa, asiakaspalvelijan tulee selvittää asiakkaansa tarpeet. Esimerkin tapauksessa tärkeää on, että asiakaspalveluohjelma ymmärtää käyttäjän antaman syötteen. Tietokoneen tulkitsema syöte voi olla tekstiä, ääntä tai vuorovaikutusta ohjelman käyttöliittymän kanssa. Toimiva tekstin- tai äänentunnistaminen kuitenkin vaatii palvelun taustalle kehittyneen ohjelmiston. Tarvittaessa tarvetäsmennystä voi tehdä myös esittämällä käyttäjälle tarvevaihtoehtoja joista käyttäjä voi valita oman tarpeensa. Vaihtoehtoja voi tarjota esimerkiksi verkkopalvelussa aktiivisena olevan näkymän ja mahdollisesti käytössä olevan käyttäjäprofiilin mukaan. Tällaisessa tilanteessa joudutaan aina tekemään oletuksia, joten se, että on olemassa riittävän laaja ja selkeä kuva käyttäjän tarpeista on palvelutapahtuman onnistumisen kannalta erityisen tärkeää.

Koska myyntikeskusteluvaihe on onnistuneeseen kauppaan pyrkivään ja puhtaasti myynnilliseen asiakaspalvelutapahtumaan liittyvä vaihe, voidaan se tämän esimerkkitapauksen kohdalla ohittaa ja siirtyä palvelun päätösvaiheeseen. Vaikka Lahtinen ym. (1998) käsittelevät myös päätösvaihetta lähinnä myynnillisestä näkökulmasta, on tämä vaihe kuitenkin hyvä käsitellä myös esimerkkitapauksen kannalta. Samoin kuin asiakaspalvelijan on hyväksyttävä tilanne jossa asiakas myyntitilanteessa päättää olla osta-

matta, on käyttötukea tarjoavan asiakaspalveluohjelman tunnistettava tilanne jossa käyttäjä ei itsepalveluna onnistu saamaan vastausta kysymykseensä. Asiakkaan rankaiseminen halusta henkilökohtaiseen apuun on kohtuutonta, eikä pakottamalla luoda positiivista imagoa tai aitoa kiinnostusta (Tuorila 2006, 13). Van Belleghem (2013) jakaa kyselyynsä vastanneet asiakkaiden yhteydenotot kolmeen ryhmään: ennen myyntitilannetta, sen aikana, sekä myyntitilanteen jälkeen tapahtuvaksi yhteydenpidoksi. Myynnin jälkeisessä vaiheessa palveluntarjoajaan yhteyttä ottaessaan kyselyyn vastanneista 48 % valitsee mieluiten itsepalvelun ja 35 % valitsee mieluiten puhelimen. Käyttäjälle on aina itsepalvelun ohella tarjottava mahdollisuus ottaa yhteys perinteiseen käyttötukeen puhelimitse.

Poistumisvaiheessa esimerkkitapauksen kannalta oleellisinta on varmistaa, että käyttäjälle on jokaisessa palveluvaiheessa selvää, miten asiakaspalveluohjelmasta poistutaan. Käyttäjän tulee pystyä päättämään palvelutapahtuma heti kun tämä on löytänyt tarvitsemansa. Koska asiakkaat odottavat palveluilta jatkuvaa kehittymistä tulisi myös asiakaspalveluohjelman kehityksen olla jatkuvaa. Poistumisvaiheessa kerätty käyttäjäpalaute on ohjelman laadun kannalta oleellista.

Jälkihoitovaihe voidaan esimerkkitapauksessa liittää poistumisvaiheeseen. Lahtisen ym. (1998) mukaan tämä vaiheen tarkoitus on varmistaa palvelutapahtuman jälkeen tapahtuvien asioiden onnistuminen. Myyntitilanteessa tällaisia ovat esimerkiksi laskutus, tavaran toimitus tai valitusten vastaanottaminen. Asiakaspalveluohjelman palautteen antamisen yhteydessä on hyvä kerätä käyttäjiltä myös vapaasti muotoiltua, avointa palautetta. Käyttäjälle tulee antaa mahdollisuus esittää mahdollinen kritiikki. Tässä vaiheessa voi tarjota myös mahdollisuutta yhteydenottopyynnön jättämiseen. Tärkeää on, että palvelutapahtuman päättää joko asiakaspalvelija tai asiakas.

Sähköisessä ympäristössä tapahtuvan asiakaspalvelutapahtuman suurin haaste on henkilökohtaisuuden ja yksilöllisen palvelutapahtuman saavuttaminen. Lehtonen ym. (116, 2002) mainitsee asiakaspalvelijan kyvyn kuunnella asiakasta tärkeänä osana näiden kahden välistä vuorovaikutusta. Legget (2017) taas muistuttaa tunnesiteen luomisen tärkeydestä sekä siitä, että asiakas odottaa aina saavansa yksilöllistä palvelua.

Palvelutilanteessa jossa ainoa aktiivinen ihmistoimija on asiakas, joudutaan asiakaspalvelijan puolesta väistämättäkin tekemään oletuksia asiakkaan tarpeista. Oletukset ovat vaarallisia, ja oletukset liiketoiminnan tai asiakkaiden suhteen voivat muodostua hyvän

palvelun esteiksi (Kannisto 56, 2008). Oletuksiin liittyvät haasteet ovat kuitenkin palveluntarjoajan itsensä luomia, joten niihin pystyy myös omalla toiminnallaan vaikuttamaan.

Legget (1, 2016) kannustaa raportissaan palveluntarjoajia monipuolisen asiakastiedon keräämiseen sekä palvelutapahtuman kehittämiseen tämän tiedon pohjalta. Palvelutapahtumista kerätyn tiedon voidaan nähdä sähköisessä ympäristössä tapahtuvan asiakaspalvelun yksilöllisyyden ainoana mahdollistajana. Aiempiin tapahtumiin perustuvalla tiedolla voidaan tehdä tarkempia oletuksia asiakkaan tarpeista.

4 PROTOTYYPIN TEKNINEN MÄÄRITYS

4.1 Käytettävä ohjelmointikieli

Verkkosovelluksen selaimessa tapahtuvien muutosten ohjelmointiin käytetään ensisijaisesti Javascriptiä. Ohjelmakirjaston lähdekoodi kirjoitetaan ECMAScript 2015:llä (ES2015), joka on JavaScriptin standardointia varten luodun määrittelyn vuonna 2015 julkaistu versio. Versio tunnettiin aiemmin myös nimellä ES6. Tämän version lisäämät muutokset tekevät JavaScriptistä huomattavasti monipuolisemman ja laajuudeltaan mittavampien ohjelmien kirjoittamiseen paremmin sopivan ohjelmointikielen. ES6-standardin mukaan kirjoitettu ohjelmakoodi on myös vanhempiin standardeihin verrattuna helppolukuisempaa.

Pääosa moderneiksi laskettavien selainten uusista versioista osaa tulkita ES6-standardin mukaista JavaScriptiä. Ongelmia tuottaa Internet Explorer 11 (IE11), selain jolla Netmarketshare.comin (2018) mukaan huhtikuussa 2018 oli maailmanlaajuisesti 12,3 % osuus kaikista Internetin käyttäjistä. Vaikka IE11 tukee ES6:n ominaisuuksista vain yhtätoista prosenttia (Kangax, 2018), standardin edelliselle versiolle, ES5:lle, IE11:llä on täysi tuki. Kääntämällä lähdekoodi ES5:n mukaiseksi on mahdollista saavuttaa tuki kiitettävälle osuudelle yleisimmin käytetyistä selaimista. Lähdekoodin kääntämiseen käytetään JavaScript-kääntäjää Babelia. Babelin avulla on mahdollista myös lähdekoodin minifiointi, lähdekoodin muuntaminen mahdollisimman vähämerkkiseen muotoon pitäen lähdekoodi edelleen selaimelle luettavassa muodossa.

Tavoitteena on luoda täysin itsenäinen ohjelmakirjasto. Lähdekoodi tulee kirjoittaa ainoastaan ES6-standardin mukaista Javascriptiä käyttäen, välttäen ylimääräisiä ohjelmistokehyksiä tai avustavia kirjastoja. Ohjelman toimintavarmuus on korkeampi, kun ohjelma on mahdollisimman vähän riippuvainen ohjelman ulkopuolisesta koodista. Puhtaan JavaScriptin käytöllä sekä minifioinnilla onnistutaan pitämään ohjelman koko mahdollisimman pienenä. Pienen tiedostokoon vuoksi myös ohjelman latausajat pysyvät lyhyempinä.

4.2 Toiminta

Käyttääkseen ohjelmakirjastoa, kutsuvan ohjelman tulee tehdä seuraavat kolme asiaa jokaisessa HTML-tiedostossa, jossa ohjelmaa halutaan olevan mahdollista käyttää.

1. Ohjelman lähdekoodin sisältävä tiedosto tulee sisällyttää script-tagin avulla.
2. Tiedostossa tulee olla HTML-elementti, jonka sisälle ohjelma piirretään.
3. Ohjelman tulee antaa ohjelmakirjastolle tarvittavat korostettaviin elementteihin liittyvät tiedot määritetyssä muodossa.

Ohjelma vastaanottaa JSON-muodossa olevan listan, joka sisältää jokaisen kutsuvassa ohjelmassa korostettavissa olevan elementin tiedot objektina. Jokaisessa objektissa on neljä avain-arvo-paria: korostustunniste, nimi, kuvaus sekä alaelementit. Kolmen ensimmäisen avaimen arvo on tyyppiä merkkijono ja alaelementit avaimen arvo on tyyppiä lista.

```
[
    {
        'korostustunniste': 'mainElem1',
        'nimi': 'Pääelementti 1',
        'kuvaus': 'Kuvaus pääelementille 1',
        'alaelementit': [
            {
                'korostustunniste': 'subElemA',
                'nimi': 'Alaelementti A',
                'kuvaus': 'Kuvaus alaelementille A'
            }
        ]
    },
    {
        'korostustunniste': 'mainElem2',
        'nimi': 'Pääelementti 2',
        'kuvaus': 'Kuvaus pääelementille 2'
    },
]
```

Kuva 1. Korostustietojen rakenne.

Korostustunnisteen arvo tulee olla sama kuin siihen sidottavan HTML-elementin data-aspa-attribuutin arvo. Nimen ja kuvauksen arvot ovat tekstisisältö joka halutaan sijoittaa korostetun elementin kohdalla näytettävän infolaatikkoon. Jos käyttäjä voi korostettavan elementin kautta siirtyä alinäkymään, jossa myös on korostettavia elementtejä, löytyvät näiden elementtien tiedot alaelementit-avaimen alta listana objekteja. Nämä objektit ovat samankaltaisia kuin ylemmällä tasolla olevat objektit. Esimerkki korostettavia elementtejä kahdella tasolla sisältävästä listasta kuvassa 1. Kuvassa on yksi pääelementti, jolla on yksi alaelementti, sekä yksi pääelementti jolla ei ole lainkaan alaelementtejä.

Ohjelmalla on kolme eri tilaa: passiivinen, aktiivinen ja korostus. Passiivisessa tilassa ohjelmasta on näkyvissä ainoastaan valikon avaamiseen käytettävä painike. Kun käyttäjä avaa ohjelman painamalla tätä painiketta, siirtyy ohjelma aktiiviseen tilaan. Tällöin näkymään avataan valikko, johon listataan linkkeinä kaikkien vastaanotettujen korostettavissa olevien elementtien nimet. Käyttäjä voi tässä vaiheessa joko sulkea valikon ja siirtyä takaisin passiiviseen tilaan, tai valita korostettavan asian klikkaamalla haluamaansa linkkiä. Linkkiä painettaessa ohjelma siirtyy korostustilaan, jossa korostettuna on painettuun linkkiin yhdistettävä elementti kutsuvassa ohjelmassa. Kutsuvan ohjelman päälle piirretään neljä HTML-elementtiä korostetun elementin ympärille: ylös, oikealle, alas ja vasemmalle. Näiden peittävien elementtien taustaväri on opasiteetiltaan 60-prosenttinen musta ja ne nostetaan z-indeksillä kaikkien muiden elementtien yläpuolelle. Korostustilassa elementit tummentavat näkymästä kaiken muun paitsi korostetun elementin. Peittävien elementtien koordinaatit lasketaan korostettavan elementin sijainnin ja selainikkunan koon mukaan. Koordinaatit tulee laskea uudelleen aina kun selainikkunan kokoa muutetaan. Korostetun elementin läheisyyteen luodaan infolaatikko, jossa käyttäjälle näytetään elementin nimi ja kuvaus. Infolaatikon sijainti suhteessa korostettavaan elementtiin määräytyy korostettavan elementin sijainnin mukaan.

Infolaatikon sisältä on mahdollista siirtyä näkymän seuraavaan ja edelliseen korostettavaan elementtiin. Ohjelma pysyy tällöin korostustilassa, mutta korostettuna on listassa edellisenä tai seuraavana lueteltu elementti. Infolaatikko sisältä myös linkin, jonka kautta käyttäjä pystyy liikkumaan korostustasolla ylös ja alas, päätoimintojen ja alitoimintojen välillä. Kun käyttäjä siirtyy infolaatikon linkin avulla näkymien välillä, ohjelma osaa käynnistää itsensä korostustilassa linkin osoitteessa annetun parametrin mukaan.

5 LOPUKSI

Opinnäytetyön tavoitteena oli kehittää verkkosovellusten kanssa käytettäväksi tarkoitetun, mahdollisimman itsenäisen ohjelmakirjaston prototyyppi. Sen toiminnallisuuden ja rakenteellisen laadun varmistamiseksi selvitimme työssämme modulaarisuuden hyötyjä ja JavaScript-ohjelmakirjastojen ominaisuuksia. Sisällön ja käytettävyyden laadun vaatimuksia pyrimme selvittämään asiakaspalvelun ja itsepalvelun suhdetta tarkastelemalla.

Modulaarisen ohjelmistokehityksen periaatteiden pohjalta oli mahdollista aloittaa mahdollisimman yleispätevän apuohjelman kehittäminen. Työn selvitysvaiheessa esille nousseet huomiot auttoivat varmistamaan, että verkossa tapahtuva asiakaspalveluun panostaminen on oikea suunta. Itsepalvelun määrän ennustetaan jatkavan kasvuaan tulevaisuudessa. Iso osa ihmisistä odottaa palveluntarjoajaltaan itsepalvelumahdollisuuksia.

Laadukas asiakaspalvelu on ensisijaisesti tunnuttava asiakkaalle yksilölliseltä ja henkilökohtaiselta. Sähköisessä palvelussa, etenkin itsepalvelussa, laadukkaan asiakaspalvelun tarjoaminen voi olla hyvin haasteellista. Yksilöllistä ja henkilökohtaista palvelua pystyy tarjoamaan ainoastaan, tuntemalla asiakkaansa. Lähes joka yhteydessä nouseva trendi tuntuu olevan asiakastietojen kerääminen ja tehokas käyttäminen. Näin on myös asiakaspalvelussa, jossa asiakastietoja on mahdollista käyttää palvelun laadun varmistamiseen tarjoamalla yhä yksilöllisempää ja henkilökohtaisempaa palvelua myös sähköisesti.

Toiminnallisuuksiltaan prototyyppi vastaa pääosiltaan suunniteltua. Kehittämisen varaa kuitenkin jäi, ja jotkin määritetyistä toiminnallisuuksista ja ominaisuuksista jäivät puuttumaan prototyypistä. Näitä muutoksia tehdessä tulee kuitenkin huomioida pyrkimys pitää prototyyppi mahdollisimman yleispätevänä. Harkitsemattomat muutokset saattavat tulevaisuudessa vaikeuttaa prototyypin käyttöönottoa toisen ohjelman kanssa. Toiminnallisuuden ja käyttöliittymän kannalta jatkokehitysideoita on listattu tarkemmin luvussa, joka on salattu toimeksiantajan toiveesta ja näin ollen jätetty pois opinnäytetyön julkaistusta versiosta. Osa listatuista kehityskohteista on ohjelman toimivuuden kannalta pakollisia muutoksia ja osa toiminnallisuuden, rakenteen ja käyttöliittymän vapaaehtoista optimointia.

Toteutimme prototyypin testikäyttöönoton yhden toimeksiantajan ohjelman kanssa. Jatkossa prototyyppi voidaan ottaa käyttöön myös muiden ohjelmien kanssa. Ennen varsinaista käyttöönottoa yhdenkään ohjelman kanssa, tulee käyttöliittymän olla yhdenmukaisempi yrityksen ohjelmistojen ilmeen kanssa, sekä käytettävien ohjelmien toimintojen olla kartoitettuna. Myös mahdollisuus toimintojen hallintaan ja muokkaukseen käyttöliittymän kautta olisi suotavaa. Opinnäytetyö ja prototyyppi toimivat dokumentaationa ja hyvänä lähtökohtana tehtävälle jatkokehitykselle.

LÄHTEET

Brumfield, B.; Cox, G.; Delgado, N.; Puleio, M.; Shifflet, K.; Smith, D. & Taylor, D. 2011 Project Silk, Client-Side Web Development for Modern Browsers, Microsoft Developer Guidance.

Kangax 2018 ECMAScript compatibility table. Viitattu 29.5.2018. https://kangax.github.io/compat-table/es6/

Edwin, N.M. 2014. Software Frameworks, Architectural and Design Patterns, Scientific Research Publishing Inc.

Encyclopedia 2004. Viitattu 8.4.2018. https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/program-library.

FOLDOC 1998 Free On-Line Dictionary Of Computing. Viitattu 8.4.2018 http://foldoc.org/.

Isoviita A.; Lahtinen J. 1998 Asiakaspalvelu ja markkinointi. Jyväskylä: Avaintulos Oy.

jQuery 2015. Plugins. Viitattu 15.5.2018. https://learn.jquery.com/plugins/.

Kannisto, P.; Kannisto, S. 2008 Asiakaspalvelu. Tampere: Amk-Kustannus.

Legget, K. 2016 Trends 2016: The future of customer service. Cambridge: Forrester Research.

Legget, K. 2017 2017 Customer service trends: Operations become smarter and more strategic. Cambridge: Forrester Research

Lehtonen, J.; Pesonen H.; Toskala A. 2002 Asiakaspalvelu vuorovaikutuksena. Jyväskylä: PS-Kustannus.

McCormik, B. 2015 How jQuery Works – Introduction. Viitattu 15.5.2018. https://benmccormick.org/2015/06/08/how-jquery-works-an-introduction/.

Meyer, B. 1997 Object-Oriented Software Construction, second edition. New Jersey: Prentice Hall.

Netmarketshare.com 2018 Browser market share. Viitattu 29.5. 2018 https://www.netmarketshare.com/browser-market-share.aspx

Newby, T. 2000 Hyvään asiakaspalveluun. Helsinki: Inforviestintä.

Nimesh, R. 2013. Why We Develop jQuery Plugins. Viitattu 15.5.2018. https://www.sitepoint.com/why-we-develop-jquery-plugins/.

SearchDataCenter 2006. Viitattu 26.4.2018. https://searchdatacenter.techtarget.com/definition/scalability.

Techopedia 2018 Software Library. Viitattu 8.4.2018. https://www.techopedia.com/definition/3828/software-library.

TechTerms 2011 Viitattu 15.5.2018. https://techterms.com/definition/scalable.

Tougher, R. 2002 Creating Reusable Software Libraries. Viitattu 8.4.2018. http://www.tldp.org/LDP/LG/issue81/tougher.html.

Tuorila H. 2006 Itsepalvelun ikääntyvä ihmemaa? Helsinki: Kuluttajatutkimuskeskus.

Valvio, T. 2010 Palvelutapahtuma ja asiakkaan kohtaaminen. Hämeenlinna: Kauppakamari.

Van Belleghem, S. 2013 Viitattu 26.4.2018 http://stevenvanbelleghem.com/blog/new-report-the-self-service-economy.

Wienand, I. 2004 Computer Science from the Bottom Up, Viitattu 27.4.2018. https://www.bottomupcs.com/libraries.xhtml#d0e85