LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Selaimessa toimivan verkkosovelluksen arkkitehtuuri

Kettunen, Timo

2017 Laurea

Laurea-ammattikorkeakoulu

Selaimessa toimivan verkkosovelluksen arkkitehtuuri

Timo Kettunen
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Tammikuu, 2017

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelma
Tradenomi (AMK)

Timo Kettunen

Selaimessa toimivan verkkosovelluksen arkkitehtuuri

Vuosi 2017 Sivumäärä 30

Tämän toiminnallisen opinnäytetyön tarkoituksena oli kuvata pienen Internet-selaimessa toimivan kyselysovelluksen kehitysprojektia sekä sen arkkitehtuurisia ratkaisuja. Lisäksi opinnäytetyössä selvitettiin yleisellä tasolla työn tukemiseksi, mitä ohjelmistoarkkitehtuuri on ja miten sitä voidaan soveltaa sovelluksien kehittämisessä. Perimmäisenä syynä projektille oli halu perehtyä syvällisemmin sovelluskehitykseen ja sen tekniikoihin.

Kyselysovelluksen kehitystyön teoriatietoa kerättiin kirjoista, e-kirjoista sekä Internetistä. Teoriaosuuden avulla selvisi yleisesti ohjelmistoarkkitehtuurin, erityisesti ohjelmiston laadullisista vaatimuksista syntyvän ”vision”, merkitys kehitystyössä. Sen onnistunut suunnittelu korostuu läpi ohjelmiston elinkaaren. Se mahdollistaa kommunikaation sidosryhmien kesken antaen tuottavan ja hallittavan käsityksen ohjelman tarkoituksesta ja rakenteista.

Projektissa määriteltiin, suunniteltiin, toteutettiin sekä lopputestattiin Internet-selaimessa toimiva kyselysovellus. Sovelluksessa käytettyjä pääasiallisia arkkitehtuureja olivat asiakas-palvelin-, MVC- sekä kerrosarkkitehtuuri. Tuotoksena syntyi toimiva verkkosovellus, jonka avulla kyselijä pystyy toteuttamaan kyselyitä, muokkaamaan niitä tarpeidensa mukaisiksi sekä tarkastelemaan saatuja vastauksia. Kyselyn vastaaja antaa vastauksensa anonyymisti avoimiin- tai vaihtoehtokysymyksiin.

Asiasanat: ohjelmistoarkkitehtuuri, sovellus, verkkosovellus

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's Thesis

Timo Kettunen

The architecture of a web application

Year 2017 Pages 30

The main objectives of this action research thesis were to introduce a development project and the architectural decisions for a small web browser based survey application. To support the main objectives, there is also a section in the thesis to explain on a general level, what software architecture is and how it can be applied to application development. The main purpose for the project was to study application development and its techniques in a greater depth.

The theoretical material to support the application development was gathered from the books, e-books and from the Internet. The software architecture was researched on a general level. In particular, the importance of "vision" in development was discussed that especially arises from the quality attributes of the software. The successful decisions made in the architectural planning manifest themselves in the life span of the software. The architectural planning also provides the basis for communication between the stakeholders. At the same time, it provides a productive and manageable picture of the purpose and the structures of the program.

A functional web browser based survey application was determined, planned and implemented in the project. The main architectures used in the application were the client-server, MVC and layered architectures. The application enables the implementation of surveys by editing open and cloze questions. The surveyor may also view the anonymous answers given by respondents.

Keywords: software architecture, application, web application

Sisällys

1 Johdanto 6
2 Lähtökohdat ja rajaukset 6
3 Käsitteitä 7
4 Ohjelmistoarkkitehtuurista 7
4.1 Ohjelmistoarkkitehtuurin tehtävät 8
4.2 Ohjelmistoarkkitehtuurityylit 10
4.3 Ohjelmistoarkkitehtuurin suunnittelun vaiheet 11
5 Kyselysovelluksessa käytetyt tekniikat 12
6 Kyselysovelluksen esittely 15
6.1 Vaatimusmäärittely 15
6.2 Suunnittelu ja toteutus 18
6.2.1 Sovelluskehys 20
6.2.2 Käyttöliittymätaso 20
6.2.3 Liiketoimintataso 24
6.2.4 Tietovarastotaso 25
6.3 Sovelluksen testaus 25
7 Yhteenveto ja oman oppimisen arviointi 25
Lähteet 27
Kuviot 29
Taulukot 30

1 Johdanto

Opinnäytetyössäni kuvaan pienen ohjelmistoprojektin, jossa toteutetaan selaimessa toimiva kyselysovellus käyttäen PHP-, HTML5- ja CSS3-kieliä. Sovelluksessa käytettävän tiedon tallennukseen käytetään MySQL-tietokantaa. Yksi suurimmista työhön kohdistuvista ongelmista on kokemattomuus sovelluksien kehityksessä. Tästä syystä, työn tarkoituksena on auttaa itseäni perehtymään yleisesti ohjelmointiin. Erityisesti haluan ymmärtää, mitä ohjelmistoarkkitehtuuri on käytännössä ja miten sovellusten sekä suurempien ohjelmistojen rakenteellisia päätöksiä tehdään.

Opinnäytteeni aluksi selvitän yleisellä tasolla, mitä on ohjelmistoarkkitehtuuri. Tämän jälkeen esittelen tekemäni sovelluksen. Esittelyssä käydään läpi sovelluksen arkkitehtuurillisia ratkaisuja sekä syitä, miten näihin ratkaisuihin on päästy.

2 Lähtökohdat ja rajaukset

Opinnäytetyöni on toiminnallinen projektityö. Tavoitteenani on määritellä, suunnitella ja toteuttaa Internet-selaimessa verkon kautta toimiva kyselysovellus sekä esitellä sen arkkitehtuurisia ratkaisuja raportin avulla. Lisäksi teoriaosuudessa selvitän kirjojen, elektronisten e-kirjojen ja Internetistä saatavan aineiston avulla, mitä ohjelmistoarkkitehtuuri on yleisesti. Toiminnallisessa opinnäytteessä ei aina tarvitse olla esiteltynä tutkimusongelmaa tai tutkimuskysymyksiä (Vilkka & Airaksinen 2003, 30). Olen kuitenkin projektia selkeyttääkseni muotoillut kysymykset ”mitä on ohjelmistoarkkitehtuuri?” sekä ”miten ohjelmistoarkkitehtuuria sovelletaan käytännössä sovelluksien kehittämisessä?”.

Kyselysovellus on pieni ja suhteellisen yksinkertainen sovellus, jonka tarkoituksena on mahdollistaa erilaisten kyselyiden toteutuksen selaimessa. Idea lähti halustani kehittää ohjelmointitaitojani. Samalla voisin tutustua paremmin ohjelmien kehitykseen. Taitojen ja ymmärryksen harjaannuttamiseksi yritin välttää käyttämästä valmiiksi tehtyjä ohjelmistoratkaisuja kuten ohjelmistokehyksiä.

Olen rajannut työtä siten, että siinä keskitytään kyselysovelluksen esittelyyn arkkitehtuurisella tasolla. Ulkopuolelle jää esimerkiksi varsinaisen koodauksen esittely. Työssä ei paneuduta varsinaisesti kyselyiden sisältöön eikä niiden laatimiseen.

3 Käsitteitä

Abstraktio Tässä työssä sillä tarkoitetaan yksinkertaistettua käsitettä ja epäolennaisen piilottamista.

HTTP (*Hypertext Transfer Protocol*) tiedonsiirtoprotokolla esimerkiksi Internet-selainten ja palvelimien välille.

HTTP GET Metodin avulla tietoa pyydetään palvelimelta.

HTTP POST Metodin avulla esimerkiksi lomakkeiden tiedot voidaan lähettää palvelimelle.

Komponentti Tässä työssä komponentilla tarkoitetaan yleisesti sovelluksen osiota.

4 Ohjelmistoarkkitehtuurista

Jokainen, niin työpöydällä kuin Internetissä toimiva sovellus ja ohjelmisto, sisältää jonkinlaisen arkkitehtuurin. Onnistuessaan arkkitehtuurin suunnittelu mahdollistaa, varsinkin suuremmissa ja monimutkaisissa järjestelmissä, sen kehittäjien tuottavan yhteistyön. Tästä rakentuu usein onnistunut kokonaisuus. (Clements ym. 2011, 1 – 2.) Huonosti suunniteltu ohjelmisto-arkkitehtuuri voi aiheuttaa ongelmia läpi ohjelmiston elinkaaren. Ongelmia, kuten ylläpidon, muokattavuuden, uudelleenkäytön ja testauksen vaikeutuminen tai kustannusten nousu, voi ilmetä. Pahimmassa tapauksessa huono arkkitehtuuri voi aiheuttaa jopa kehitystyön keskeytymisen. (Gorton 2011, 11; Koskimies & Mikkonen 2005, 16 - 17.)

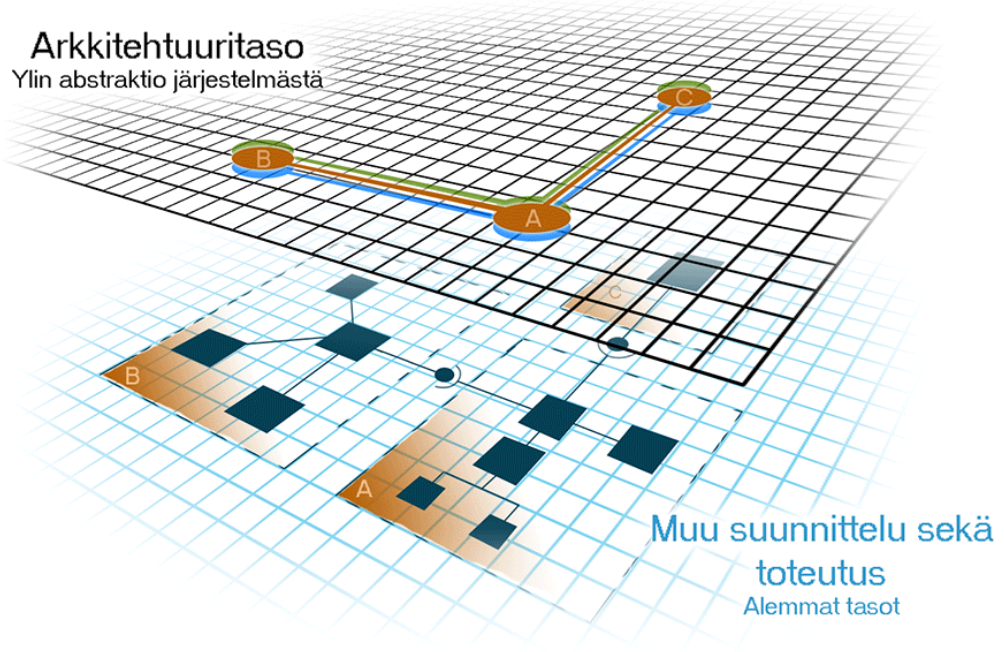
Ohjelmistoarkkitehtuurilla on monia erilaisia määritelmiä. Fowlerin (2003, 1) mukaan arkkitehtuuri on yleisesti mielletty järjestelmän korkeimman tason osioinniksi. Koskimiehen & Mikkosen (2005, 18) mukaan ”arkkitehtuuri on järjestelmän perustuslaki”. Fowler (2003, 1) mainitsee, että arkkitehtuurin ajatellaan usein kattavan myös ohjelman vaikeasti muutettavat suunnittelupäätökset. Ne juontavat juurensa liiketoiminnan tärkeimmistä tavoitteista sekä toiminnallisista ja laadullisista vaatimuksista. Se voi antaa suoranaisia rajoituksia muun muassa käyttävien standardien ja rajapintojen suhteen. Ajan kuluessa arkkitehtuuri voi muuttaa muotoaan ohjelmiston arkkitehtien ja kehittäjien toimesta esimerkiksi ohjelmiston vaatimuksien muututtua. Arkkitehtuuri vaikuttaa usein myös kehitystyön tekevän organisaation rakenteeseen ja työnjakoon. (Clements ym. 2011, 6 – 7; Koskimies & Mikkonen 2005, 21, 224.)

Brown (2016, 3) kuvailee arkkitehtuuria visioksi ohjelmasta. Kuvauksellaan hän tarkoittaa ohjelman vaatimuksien muuntamista kommunikoitavissa olevaksi käsitykseksi ohjelmasta, sen tarkoituksesta ja rakenteesta. Arkkitehtuurinen visio mahdollistaa järjestelmän helpomman

kehittämisen ja ylläpidon. Se muodostaa järjestelmän vaatimuksista rajat joiden sisällä kehitys tapahtuu. Olipa projekti vaatimuksiltaan vakaa, suunnitelman mukaan läpivietävä tai vaatimuksiltaan epävakaampi iteratiivisesti sykleissä etenevä, on visio tarpeellinen. (Booch, Maksimchuk, Engle, Young, Conallen & Houston 2007, 248 – 249, 254 – 255; Brown 2016, 3, 6 – 7, 84 – 87.)

Ohjelmistoarkkitehtuuri on pienempi osa laajempaa ohjelmistosuunnittelua. Ohjelmistosuunnittelun tekniikoiden tarkoituksena on monimutkaisuuden hallinta. Suurempi ongelma jaetaan ihmismielelle helpommin käsiteltäviin yksinkertaisempiin osioihin. Tämä jako edesauttaa haasteiden eriyttämisen (*Separation of Concerns*) -periaatetta. Näin osioiden vaikutukset toisiinsa pienenevät. (Booch ym. 2007, 11; Clements ym. 2011, 6 – 9; McConnell 2004, 78 – 79.)

Arkkitehtuuri on ohjelmiston abstraktiotasoista ylin taso (kuvio 1), jonka avulla järjestelmän osioista ja näiden suhteista saadaan esitettyä yksinkertaisempia näkökulmia. Se piilottaa näkyvistä yksityiskohdat kuten osioiden sisäiset epäolennaiset tiedot joilla ei ole sen ulkopuolella merkitystä. (Bass, Clements & Kazman 2013; Koskimies & Mikkonen 2005, 17.)



Kuvio 1: Arkkitehtuuritaso on ylin abstraktio järjestelmästä (Kettunen 2017)

4.1 Ohjelmistoarkkitehtuurin tehtävät

Ohjelmistoarkkitehtuurin perimmäisenä tehtävänä on täyttää järjestelmän sidosryhmien tarpeet. Se tekee sidosryhmien odotukset ja vaatimukset täyttävät päätökset mahdollisimman

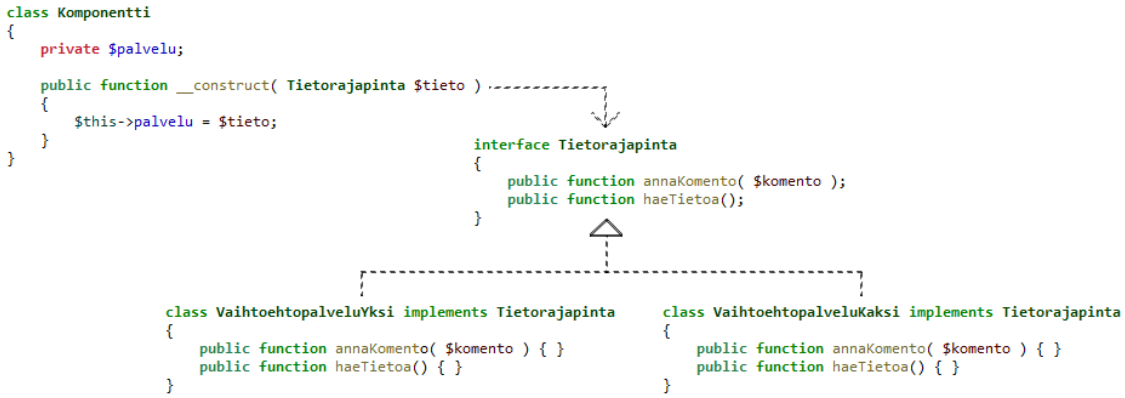
hyvin ja perustelee ne. Tällaisia päätöksiä ovat rakenteen osiot ja niiden vuorovaikutukset. Lisäksi päätökset ohjaavat ohjelmaa toimimaan organisaation liiketoiminnallisten tavoitteiden mukaisesti. Nämä päätökset voivat määrittää ohjelmistossa käytettävät teknologiat sekä tarvitun tiedon hankinta- ja tallennusmetodit. Arkkitehtuuri voi myös tehdä päätökset muun muassa turvallisuuteen, virheiden hallintaan ja vikatiloihin varautumiseen liittyen. Sidosryhmien vaatimukset voivat olla ristiriitaisia keskenään, jolloin arkkitehtuurin täytyy tehdä päätöksissä kompromisseja. (Clements ym. 2011, 6 – 7; Koskimies & Mikkonen 2005, 18 – 19, 24; McConnell 2004, 46 - 49.)

Kommunikaatio sidosryhmien kanssa tapahtuu ohjelmiston arkkitehdin tekemän arkkitehtuurikuvauksen avulla. Se kuvaa täsmällisesti, yksiselitteisesti ja perustellen arkkitehtuurin ja sen sallivuudet ratkaisuissa. Ohjelmiston kehittäjien, analyytikkojen, ylläpitäjien, testaajien, käyttäjien sekä muiden ohjelmistoon sidoksissa olevien täytyy saada selvä ymmärrys aikaansaaduista päätöksistä. Heidän pitäisi pystyä tekemään työnsä oikein ja ilman, että he joutuvat tekemään omia tulkintojaan arkkitehtuurista. Esimerkiksi analyytikoille kuvaus antaa tarvittavat tiedot järjestelmän analyysille, jossa tutkitaan vaatimuksien täyttymistä. Arkkitehtuurikuvaus elää arkkitehtuurin mukana. Se toimii ohjelmistokehityksen tukena ja apuvälineenä myös varsinaisen rakennusvaiheen jälkeen, jos se on päivitetty asianmukaisesti. (Clements ym. 2011, 9 – 10, 12, 14; Koskimies & Mikkonen 2005, 24, 29 - 30.)

Jokaisella arkkitehtuurin sidosryhmän jäsenellä on omat huolenaiheensa järjestelmästä ja siten oma näkökulmansa tuotettavaan ohjelmistoon. Arkkitehtuurikuvaus koostuu useista sidosryhmien näkymistä, joista jokainen vastaa omalla tavallaan ohjelmalle tehtyihin vaatimuksiin. Nämä vaatimukset määrittävät tarvittavat näkymätyypit, jotka esittävät vain sille ominaisesti tarpeelliset tiedot ja peittävät näkymälle tarpeettomat tiedot ohjelmiston elementeistä ja vuorovaikutuksista. (Clements ym. 2011, 22 - 23; Koskimies & Mikkonen 2005, 31.)

Ohjelmistoarkkitehtuurin perusajatus on pilkkoa ohjelmiston rakenne mahdollisimman vähän toisistaan riippuvaisiin osioihin. Näillä osioilla on jokaisella oma tehtävänsä järjestelmän toiminnassa. Osiointi mahdollistaa niiden paremman muunneltavuuden sekä nopeamman itsenäisen kehittämisen eri kehittäjien toimesta samanaikaisesti. Arkkitehtuuri voi vaatimuksien täyttämiseksi ottaa kantaa näiden osioiden suhteisiin ja niiden väliseen kommunikaatioon, kehittymiseen ja ohjelman ajonaikaiseen käyttäytymiseen. Arkkitehtuuri ei ota kantaa osioiden sisäiseen rakenteeseen, ellei kyseessä ole suuri ja merkittävä osio ohjelmistossa. Tällöin tälle osiolle voidaan tarvittaessa luoda omakohtainen arkkitehtuurikuvaus. (Clements ym. 2011, 8; Gorton 2011, 3 - 4; Koskimies & Mikkonen 2005, 18, 55). McConnell (2004, 45) määrittää, että osiot voivat olla pelkkiä yksittäisiä luokkia, luokkaryppäitä tai kokonaisia omia arkkitehtuurimaisia rakenteita.

Koskimies & Mikkonen (2005, 53 - 55) käyttävät ohjelmiston osiosta nimitystä komponentti. Komponentit vaativat tai tarjoavat palveluita julkisten rajapintojen kautta. Komponenttien palveluiden toteutus on näiden komponenttien yksityinen oma asia. Tämä mahdollistaa palvelua käyttävän osapuolen olevan riippuvainen vain palvelun abstraktiosta eli rajapinnasta. Tästä seuraa, että palvelun varsinainen toteutus on muutettavissa tai vaihdettavissa tarvittaessa toiseen (kuvio 2). (Bass ym. 2013; Koskimies & Mikkonen 2005, 57 - 59.)



Kuvio 2: "Komponentti”-luokka on riippuvainen ”Tietorajapinta”-rajapinnasta, ei rajapinnan toteuttavista vaihtoehtoisista luokista

4.2 Ohjelmistoarkkitehtuurityylit

Arkkitehtuuriset suunnittelupäätökset ovat usein vaikeita varmistaa ja testata ohjelmiston suunnitteluvaiheessa. Siksi ohjelmiston arkkitehdit turvautuvat usein riskien minimoimiseksi testattuihin ja hyväksi koettuihin ratkaisuihin eli tyyleihin ohjelmiston rakenteessa. (Gorton 2011, 9 – 10.)

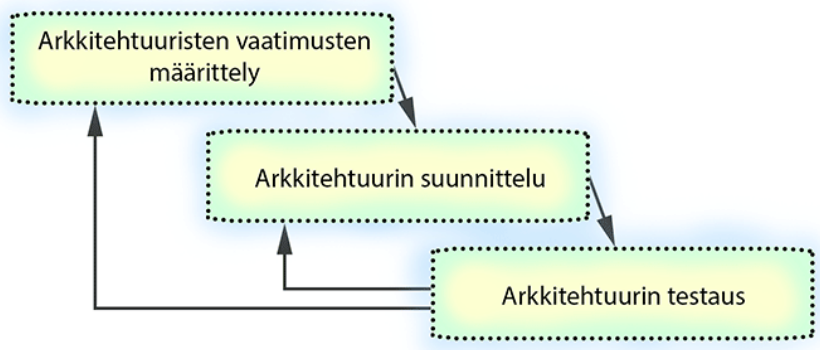
Clementsin ym. (2011, 32, 34 - 36) mukaan *arkkitehtuurityylit* sekä samankaltaiset, kuitenkin yksityiskohtaisemmat *arkkitehtuurimallit*, ovat joukko valmiiksi koottuja, yleisluontoisia ja jalostamista vaativia suunnittelupäätöksiä. Ne voivat sisältää esimerkiksi tietoa tarvituista komponenteista, niiden välisistä vuorovaikutuksista sekä rajoitteista. Yhdessä ohjelmistossa voidaan sitä rakentaessa käyttää montaa erilaista arkkitehtuurityyliä ja -mallia. (Clements ym. 2011, 36.)

Koskimies & Mikkonen (2005, 125) käyttävät koko järjestelmän muotoa koskevista suunnittelupäätöksistä yleisesti nimitystä arkkitehtuurityylit. Ne muistuttavat läheisesti *suunnittelumalleja*, jotka puolestaan ratkaisevat järjestelmän suunnittelun alemman tason ongelmatapauksia (Koskimies & Mikkonen 2005, 102, 125).

Yleisesti suunnittelumallit koostuvat ongelmasta, ongelmayhteydestä ja ratkaisusta. Ongelma on yleisesti toistuva ja ohjelmointikielestä riippumaton. Se koostuu ratkaisulle annetuista vaatimuksista, rajoituksista sekä toivotuista ominaisuuksista. Ongelmayhteys määrittää, milloin ongelma tavallisesti esiintyy ja koska ratkaisua on siihen pätevä käyttää. Se voi esimerkiksi suosittaa joissain tilanteissa käyttämään tiettyjä suunnittelumalleja yhdessä. Ratkaisu määrittää tarvittavat muuttumattomat rakenteet sekä näiden rakenteiden vuorovaikutukset toisiinsa nähden. (Buschmann, Meunier, Rohnert, Sommerlad & Stal 1996, 8 - 10; Koskimies & Mikkonen 2005, 101, 105.)

4.3 Ohjelmistoarkkitehtuurin suunnittelun vaiheet

Arkkitehtuurin suunnittelu on iteratiivinen prosessi joka koostuu arkkitehtuuristen vaatimuksien määrittelystä, arkkitehtuurin rakenteen suunnittelusta sekä aikaansaadun arkkitehtuurin testauksesta (kuvio 3). Testauksessa saadaan käsitys siitä, onko arkkitehtuuri vaatimuksien mukainen. Arkkitehtuuria parannetaan, kunnes se täyttää vaatimukset. (Gorton 2011, 97 – 99.) Arkkitehtuurin muuttaminen jälkikäteen voi tulla kalliiksi (Koskimies & Mikkonen 2005, 17).



Kuvio 3: Arkkitehtuurin suunnittelun vaiheet (mukailtu Gorton 2011, 98)

Arkkitehtuuriset vaatimukset koostuvat toiminnallisista ja laadullisista vaatimuksista sekä rajoituksista, kuten käytetyt teknologiat tai kehitykseen varattu budjetti. Vaatimukset voivat koskea lähdekoodin periaatteita tai siihen kohdistuvia testejä. Olennaista on ymmärtää myös ohjelmistoon kohdistuvat riskit, kuten mahdolliset häiriötilat. (Brown 2016, 56 – 57.)

Toiminnalliset vaatimukset ovat syy ohjelmiston rakentamiselle. Ne kertovat ohjelmiston toiminnan tavoitteet, eli mitä ohjelmiston tulisi tehdä. Laadulliset vaatimukset vastaavat *millä tavalla* ohjelmistolle annettuihin toiminnallisiin vaatimuksiin suhtaudutaan. Laadullisia vaatimuksia ovat esimerkiksi skaalautuvuus, suorityskyky ja turvallisuus. Ne voivat liittyä liiketoi-

minnallisiin tavoitteisiin antaen sille etulyöntiaseman kilpaileviin ohjelmistoihin nähden. Vaatimukset voivat tulla myös organisaation ulkopuolelta esimerkiksi maassa noudatettavien lakien tai säädöksien muodossa. Usein vaatimukset muuttuvat kehitystyön edetessä. Näin voi käydä muun muassa silloin, kun käyttäjien toiveet ovat olleet aluksi liian epämääräisiä. Tämä aiheuttaa huomattavia haasteita ohjelmistorakenteiden muunneltavuuden kannalta. Rakenteet tulisi suunnitella tarvittaessa niin joustaviksi, että muutoksiin on mahdollisimman helppo varautua. Arkkitehtuurin täytyisi siis pystyä osoittamaan, että ohjelmistoon on mahdollista tehdä tarpeen vaatiessa muutoksia, jotka ovat helppoja toteuttaa. Myös liiallinen joustavuus ohjelmiston rakenteissa voi aiheuttaa ongelmia. Siitä voi esimerkiksi aiheutua suunnittelun ja toteuttamisen vaikeutumista sekä ohjelmiston rakenteiden turhaa monimutkaistumista. (Bass ym. 2013; Booch ym. 2007, 8 – 9; Brown 2016, 58 - 59, 61; Gorton 2011, 23, 31; McConnell 2004, 52.)

Ohjelmistolle annetut vaatimukset ovat usein ristiriitaisia. Tällöin joudutaan tekemään myönnytyksiä. Esimerkiksi ohjelmiston vaatiessa suurta käyttövalmiutta ja luotettavuutta, sen suorituskyky voi huonontua. (Gorton 2011, 37, 100.)

Arkkitehtuurin suunnittelu koostuu arkkitehtuurin strategian sekä ohjelmiston komponenttien ja niiden vastuiden määrittelystä. Arkkitehtuurin strategia ja sen sisältämät tyylit valitaan ohjelmalle annettujen vaatimuksien avulla. Jokaisella arkkitehtuurityylillä on omat tapansa käsitellä näitä vaatimuksia strategiassa. Suunnittelussa esiintyvien ongelmien ratkaisussa voidaan turvautua alan tuntevien asiantuntijoiden (*domain expert*) apuun. (Booch ym. 2007, 134; Gorton 2011, 101 - 102.)

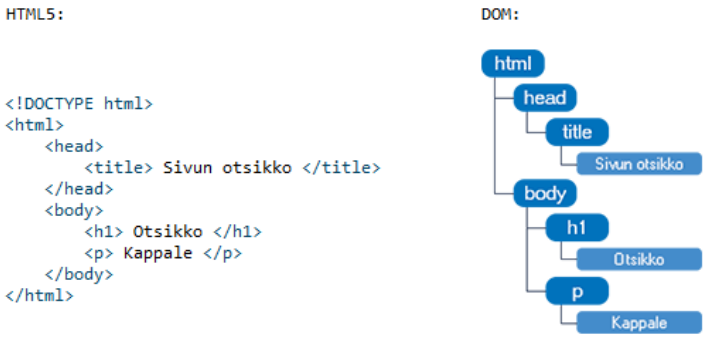
Arkkitehtuurin testauksessa eli arvioinnissa koetellaan erityisesti arkkitehtuurille annettujen laadullisten vaatimuksien täyttymistä ohjelmiston rakenteissa. Tämän vuoksi annettujen laadullisten vaatimuksien tulisi olla mitattavissa selkeillä kriteereillä. Tällainen kriteeri on esimerkiksi vasteaika eli aikamääre, jossa ohjelmiston halutaan antavan vastaus käyttäjälle. Vaatimuksissa on oltava määritettynä, minkä asian suhteen vaatimus on annettu. Arvioinnissa voidaan myös testata toiminnallisia vaatimuksia. Tällöin testataan kykenevätkö suunnitellut komponentit tuottamaan vaaditut toiminnallisuudet. (Gorton 2011, 23, 25; Koskimies & Mikkonen 2005, 221 - 223.)

5 Kyselysovelluksessa käytetyt tekniikat

Kyselysovelluksessa käytettyjä tekniikoita ovat käyttöliittymäpuolella (*front-end*) HTML5 ja CSS3 sekä näiden mahdollistama responsiivinen suunnittelu. Sovelluksen ulospäin näkymättömällä puolella (*back-end*) käytettiin PHP-kieltä.

HTML (*HyperText Mark-up Language*) eli Hypertekstin merkkauskieli on verkkosivujen esittämiseen kehitetty kieli, jota käytetään nykyään lukuisissa muissa kohteissa kuten esimerkiksi sähköposteissa sekä elektronisissa kirjoissa. HTML-kielen ideoi vuonna 1989 Sveitsissä tutkija Tim Berners-Lee Euroopan hiukkasfysiikan tutkimuskeskuksessa CERN:ssä. Idean takana oli halu kehittää keskitetty tapa esittää ja jakaa tieteellisiä dokumentteja tutkijoiden keskuudessa ympäri maailman. (Korpela 2014, 26; Raggett, Lam, Alexander & Kmiec 1998.)

HTML-kieli koostuu elementeistä, jotka jakavat sivun erilaisiin rakenteellisiin osioihin kuten otsikoihin ja tekstikappaleisiin. Elementti koostuu aloitustagista, elementin sisällöstä kuten tekstistä tai toisista elementeistä sekä elementin päättävästä sulkemistagista. HTML-dokumentti voidaan kuvata selventävänä DOM-puuna (*Document Object Model*, *Dokumenttioliomalli*)(kuvio 4). (Korpela 2005; Korpela 2014, 62.)



Kuvio 4: Esimerkki HTML-sivun DOM-puusta

HTML5:n pääperusperiaatteita ovat sen yhteensopivuus aiempiin HTML-versioihin. Vanhat sivut toimivat uusissakin selaimissa kuten ne ovat alun perin suunniteltu ja niitä voidaan tarvittaessa päivittää vastaamaan nykystandardeja. HTML5 standardin on myös pystyttävä selittämään ja kuvaamaan tarpeeksi selaimien toimintaa, jotta eri selainvalmistajat osaisivat tehdä yhdenmukaisia selaimia sekä sivujen kehittäjät tietäisivät miten selain käyttäytyy tietyissä tilanteissa. (Korpela 2014, 35 – 36; MacDonald 2014, 7.)

Periaatteellisesti kielen tulee standardoida virallisiksi ominaisuuksiksi epäviralliset sivujen tekijöiden parissa paljon käytetyt tavat. Kieleen tehtävien muutoksien tulee olla vain käytännöllisiä. Esimerkiksi HTML5:een on käytännön syistä sisällytetty itseensä videon toisto, joka tekee osittain turhiksi selainten videon toistoon tarkoitetut lisäosat kuten Adobe Flash -soittimen. (MacDonald 2014, 7 – 9.)

CSS3 (*Cascading Style Sheet*) on HTML- ja XML-dokumenttien ulkonäön määrittelyyn kehitetty muotoilukieli. Kielen avulla voidaan muuttaa dokumentin elementtien ominaisuuksia tyyliohjeiden (style sheet) avulla. Tyyliohjeet voidaan antaa dokumentille kirjoittamalla se suoraan

dokumentin lähdekoodiin tai antamalla dokumentin käyttöön erillinen CSS –tiedosto. (Korpela 2013, 32 – 33, 43 - 44; McFarland 2013, 1.)

Yksittäinen CSS-tyyliohje koostuu selektorista ja deklaraatiosta. Selektorilla määritetään, mihin muotoiltavan dokumentin elementtiin tai elementteihin ohje kohdistuu. Deklaraatio puolestaan määrittää halutun arvon elementin ominaisuudelle. (Korpela 2003; Korpela 2013, 48; McFarland 2013, 37.)

PHP (*Hypertext Preprocessor*) on kehittäjänsä Rasmus Lerdorfin 1995 esittelemä, erityisen PHP-tulkin kautta ajettava kieli. Kieli on alun perin tarkoitettu dynaamisen Internet-sisällön tuottamiseen. PHP:tä voidaan käyttää myös suoraan käyttöjärjestelmän komentotulkin kautta tai itsenäisenä graafisena ohjelmana PHP-GTK –lisäosan avulla. (Heinisuo & Rauta 2007, 12; Tatroe, MacIntyre & Lerdorf 2013, 1 - 2.)

Internet-sivuilla PHP-kieltä käytetään kirjoittamalla sivun lähdekoodin HTML-kielen sekaan PHP -komentoja (kuvio 5). Palvelimelle asennettu tulkki ajaa nämä komennot aina ennen kuin palvelin lähettää sivun sitä tarkastelevalle selaimelle (Heinisuo & Rauta 2007, 13).

```
<!DOCTYPE html>
<html>
    <head>
        <title> Sivun otsikko </title>
    </head>
    <body>
        <h1> HTML-otsikko </h1>
        <p> <?php echo "PHP koodia."; ?> </p>
    </body>
</html>
```

Kuvio 5: Esimerkki PHP-koodista HTML-sivulla

PHP tukee olio-ohjelmointia, jossa sovellus koostuu yhteistyössä toimivista, sovelluksen maailmaan liittyvistä olioista. Nämä oliot ovat luokkien (kuvio 6) ajonaikaisia ilmentymiä. Olioilla on oma identiteetti, tila, käyttäytyminen ja olion sisäisen informaation piilottava julkinen käyttäytymisrajapinta. Luokka sisältää jonkin tietyn oliotyypin ominaispiirteet ja se voi olla periytynyt toisesta luokasta. Tällöin se voi käyttää uudelleen tämän toisen luokan rakenteita ja käyttäytymistä. (Booch ym. 2007, 41, 51, 77, 93 - 94, 100; Koskimies 2000, 24; Tatroe, MacIntyre & Lerdorf 2013, 147.)

```php
class Auto
{
    private $merkki;
    private $istuintenLukumäärä;
    private $matkustajat;
    private $matkustajienLukumäärä;

    public function __construct( $merkki, $istuintenLkm )
    {
        $this->merkki = $merkki;
        $this->istuintenLukumäärä = $istuintenLkm;
        $this->matkustajat = [];
        $this->matkustajienLukumäärä = 0;
    }

    public function lisääMatkustaja( Matkustaja $matkustaja )
    {
        if ( $this->matkustajienLukumäärä < $this->istuintenLukumäärä )
        {
            $this->matkustajat[] = $matkustaja;
            $this->matkustajienLukumäärä++;
        }
        else
        {
            return FALSE;
        }
    }
}
```

```php
class Matkustaja
{
    private $nimi;

    public function __construct( $nimi )
    {
        $this->nimi = $nimi;
    }

    public function nimi()
    {
        return $this->nimi;
    }
}
```

Kuvio 6: ”Auto” –luokka, jonka sisälle voidaan tuoda ”Matkustaja” –luokka

PHP sisältää PDO (*PHP Data Objects*) kirjaston, jonka avulla pystytään ottamaan yhteys moniin erilaisiin tietokantoihin kuten MySQL (Tatroe, MacIntyre & Lerdorf 2013, 203). MySQL on relaatiotietokanta, joka ymmärtää SQL (*Structured Query Language*) –kieltä tiedon hakemiseen ja tallentamiseen (Welling & Thomson 2009, 243 - 244).

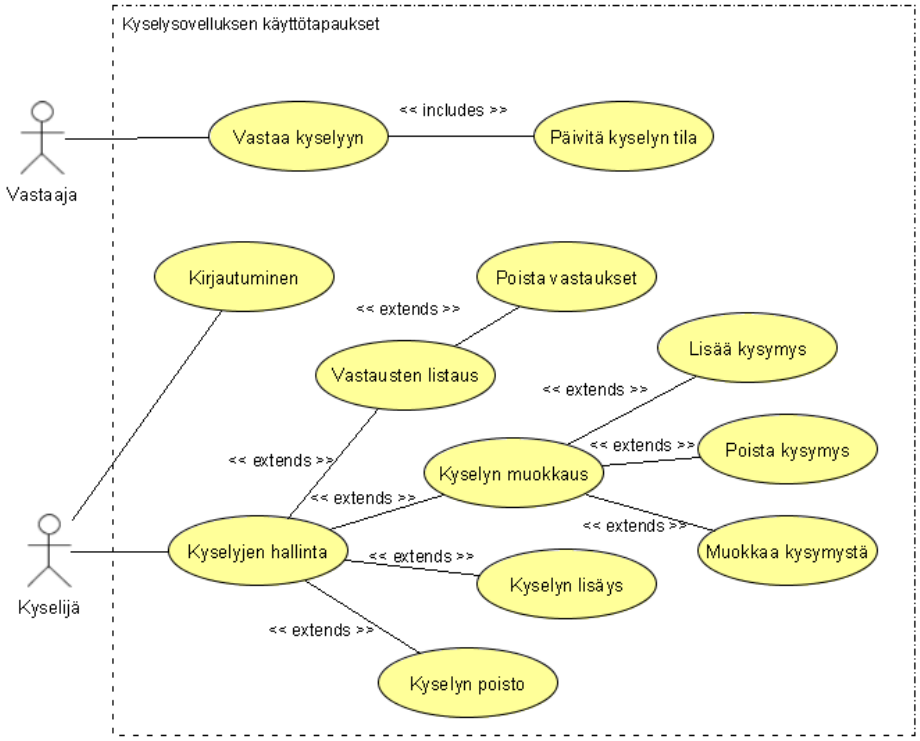
6 Kyselysovelluksen esittely

Kyselysovelluksella voidaan tehdä Internet-selaimessa toimivia lomakekyselyitä. Kyselyn toteuttaja voi muokata luomiaan kyselyitä tarkoituksiensa mukaisiksi sovelluksen sisältämien hallintasivujen avulla. Kyselyissä käytettäviä kysymysvaihtoehtoja ovat teksti- ja vaihtoehtokysymykset. Kysely näyttää yhden kysymyksen kerrallaan ja vastaukset välittyvät kyselijälle anonyymisti. Kyselijä pystyy tarkastelemaan saatuja vastauksia kyselyidensä hallintasivuilta.

Sovelluksen kehitys tapahtui Eclipse-kehitystyökalun sekä XAMPP–palvelinohjelmiston avustuksella, joista molemmat ovat ilmaisia. Ohjelman työstäminen tapahtui tekemällä erilaisia prototyyppejä, joita paransin tiedon ja kokemuksen kasvaessa.

6.1 Vaatimusmäärittely

Kyselysovelluksen varsinainen työstö alkoi määrittelemällä, miksi sovellus rakennetaan ja mitä sen on tarkoitus tehdä. Näistä määrityksistä syntyivät sovelluksen käyttötapaukset (kuvio 7). Käyttötapauksista syntyivät edelleen toiminnalliset vaatimukset. Oleellisimpana vaatimuksena sovellukselle oli sen käyttöliittymän toimiminen Internet-selaimessa helpon käytön mahdollistamiseksi.



Kuvio 7: Kyselysovelluksen käyttötapaukset

Käyttäjiä sovelluksessa ovat kyselijä ja vastaajat. Samalla palvelimella voi sijaita monia erilaisia kyselyjä. Sovelluksessa ei ole toistaiseksi kuin yksi kyselijä, mutta siihen voidaan kuitenkin jatkokehityksessä tarvittaessa lisätä mahdollisuus monille kyselijöille.

Kysely alkaa vastaajan mennessä hänelle annettuun Internet-osoitteeseen. Tämä osoite voi sisältää GET-kyselytunnuksen, jonka perusteella sovellus etsii sitä vastaavan kyselyn tietokannasta. Sovellus näyttää kyselyn löytyessä ”tervetuloa”-viestin ruudulla. Jos hakukoodi ei ole määritettynä osoitteeseen tai tietokannasta ei löydy tuolle koodille vastinetta, vastaajalle näytetään sivu johon hän voi kirjoittaa haettavan kyselyn hakukoodin. Vastaaja voi aina painaa ”keskeytä”-nappia, jolloin hänet ohjataan kyselystä pois tallentamatta vastauksia. Painamalla ”seuraava”-nappia, kysely tallentaa sessioon mahdollisen vastauksen ja hakee uuden kysymyksen sekä siihen mahdollisesti jo annetun vastauksen. Vastaaja voi siis halutessaan selata kysymyksiä eteen- ja taaksepäin. Kysymysten loppuessa näytetään ”kiitos”-viesti ja vastaukset viedään tietokantaan. Vastaajan painaessa selaimen ”päivitä ruutu”-nappia, kysely keskeytyy ja palaa kyselyiden hakusivulle. Tämä estää kesken jääneen kyselyn vastausten tarkastelun, jos kyselyyn palataan myöhemmin toiselta sivustolta.

Kyselyn hallintasivut näyttävät kyselijän laatimat kyselyt ja saadut vastaukset. Ne antavat hänelle myös mahdollisuuden lisätä, muokata ja poistaa kyselyitä, kysymyksiä ja kysymysten vaihtoehtoja.

Toiminnallisten vaatimuksien lisäksi määritettiin laadullisia vaatimuksia, jotka ohjasivat kyselysovelluksen arkkitehtuurisia ratkaisuja. Laadullisista vaatimuksista (taulukko 1) kyselysovelluksessa tavoiteltiin erityisesti muunneltavuutta, jotta tarvittaessa tietokanta voitaisiin korvata toisella saatavissa olevalla tietokantatyypillä. Tarvittaessa se voidaan korvata pelkkään tekstitiedostoon nojautuvalla ratkaisulla. Myös käyttöliittymän haluttiin olevan mahdollisesti vaihdettavissa. Käyttöliittymä haluttiin olevan ulkoasultaan responsiivinen, eli sen tuli mukautua toimimaan sekä tietokoneiden että matkapuhelimien selaimissa.

Kyselysovelluksen rajoituksia ja laadullisia vaatimuksia	Ratkaisu
Rajoituksia (toimii selaimessa ja ulkoasultaan responsiivinen)	- HTML5 ja CSS3 - responsiivinen suunnittelu
Muunneltavuus (tietokanta ja käyttöliittymä vaihdettavissa / muunneltavissa)	- Kerrosarkkitehtuuri (vaihdettavat tasot) - MVC-arkkitehtuuri
Turvallisuus (käyttäjän tunnistus ja valtuutukset, syötteiden tarkistus)	- Kyselyjen hallintasivuilla kirjautuminen - Käyttäjien syötteiden tarkistus ennen niiden vientiä tietokantaan

Taulukko 1: Kyselysovelluksen rajoituksia ja laadullisia vaatimuksia

Turvallisuuteen on panostettu erityisesti kyselyjen hallintasivuilla. Kyselijän täytyy olla tunnistettu ja valtuutettu päästäkseen omille sivuillensa. Kyselysovelluksen yleisenä turvallisuusnäkökohtana on ollut vastauslomakkeiden sisällön tarkastus esimerkiksi vaarallisten SQL-kyselyiden varalta. Puutteellisesti suojatun kaavakkeen kenttiä voitaisiin käyttää vihamielisesti hyväkseen esimerkiksi lähettämällä vahingollista tietoa tietokantaan.

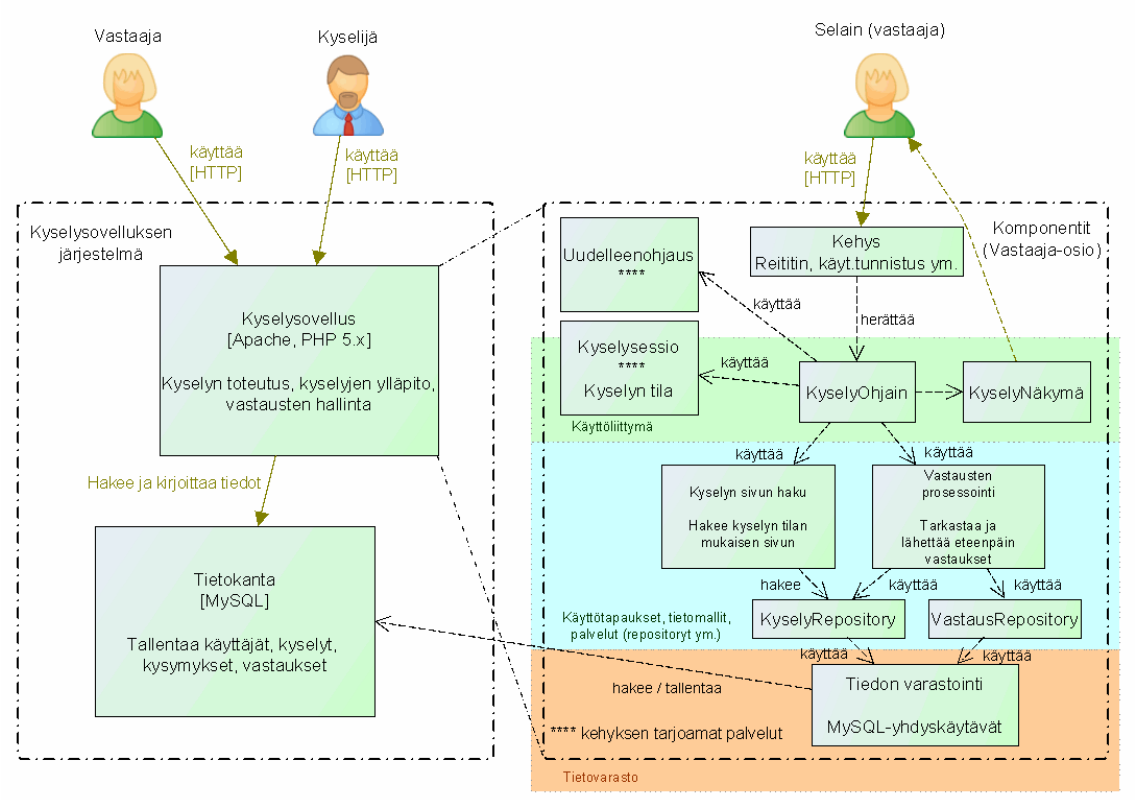
Kehitystyötä rajoitettiin siten, että se tapahtui osittain jo tutuilla kielillä kuten PHP, HTML5 ja CSS3. Näistä HTML5 ja CSS3 mahdollistavat responsiivisuuden, jonka avulla sivuston ulkoasu muokkautuu esimerkiksi selaimen koon mukaisesti. Näin tietokoneelle ja puhelimelle ei tarvitse tehdä omia sivujaan. Sama sivusto käy molemmille alustoille.

Sovelluksessa noudatettaviksi periaatteiksi määritettiin sen rakenteen jakaminen olio-ohjelmoinnin mukaisesti omat vastuualueensa omaaviin komponentteihin. Helpon muunnelta-

vuuden aikaansaamiseksi näiden komponenttien väliset riippuvuudet oli syytä toteuttaa löyhästi erityisesti liiketoimintalogiikan ja tiedonvarastoinnin välisten komponenttien osalta.

6.2 Suunnittelu ja toteutus

Kyselysovelluksen suunnittelu tapahtui vaatimusmäärittelyiden jälkeen. Suunnittelun tuotoksena syntyivät käyttötapauksien toiminnallisuuden ja kokonaisuuden mahdollistavat komponentit (kuvio 8). Komponenttien määrä ja rakenne muuttui sovelluksen työstön aikana, sillä halutut toiminnot lisääntyivät ajan myötä. Lisäksi tekniikoiden tullessa tutummiksi, koodia voitiin parantaa sekä selkeyttää ymmärrettävyyden aikaansaamiseksi. Kyselysovelluksen vaatimuksiin kuuluu interaktiivinen selaimessa käyttäjän kanssa kommunikoiva käyttöliittymä.



Kuvio 8: Sovelluksen rakenne sovelluksen vastaaja-osion osalta. Mukailtu Brownin (2016, 38) C4-arkkitehtuurimallista

Kyselysovellus on Internet-selaimessa toimiva verkkosovellus. Se käyttää toiminnassaan pohjimmiltaan asiakas-palvelin-arkkitehtuurityyliä. Siinä asiakas-komponentti ja palvelin-komponentti ovat vuorovaikutuksessa pyyntö-vastaus-vuorovaikutustekniikan mukaisesti. Tyypillisesti asiakas herättää palvelimen tarjoaman palvelun ja jää odottamaan kunnes palvelin lähettää vastauksensa. Esimerkiksi HTML-dokumenttien selaaminen Internetissä toimii asia-

kas-palvelin-periaatteella. Tässä tapauksessa asiakas on selain, joka on HTTP-protokollan avulla Internet-palvelimeen yhteydessä. (Clements ym. 2011, 124, 163 - 165.)

Ohjelmakoodin ymmärrettävyydeksi ja ohjelman helpomman jälkeenpäin muokattavuuden aikaansaamiseksi erityisesti tietokannan mahdollisen vaihtamisen edesauttamiseksi, kyselysovellus käyttää kerrosarkkitehtuurityyliä (kuvio 9).

Kuvio 9: Kyselysovelluksen tasot

Kerrosarkkitehtuurityylin ajatuksena on jakaa ohjelmisto päällekkäin oleviin, eri vastuualueisiin erikoistuneisiin tasoihin. Tyypillisesti ylempi taso käyttää suoraan sen alapuolella olevaa tasoa. Nämä tasot sisältävät palveluita tai komponentteja jotka toteuttavat rajapintoja ja käyttävät alempien tasojen rajapintoja. Tietyissä tilanteissa on sallittua ohittaa tasoja esimerkiksi jos suoraan alempi taso ei pysty tarjoamaan pyydettyä palvelua. Alempi taso ei saisi käyttää ylemmän tason tarjoamia palveluita tai komponentteja jos tästä aiheutuu alemman tason riippuvuus ylemmästä tasosta. (Clements ym. 2011, 88 – 91; Koskimies & Mikkonen 2005, 126.)

Palveluiden ja komponenttien sijoittaminen rajapintojen taakse edesauttaa erityisesti muunneltavuutta, siirrettävyyttä sekä uudelleenkäytettävyyttä. Tällöin tasoja voidaan muuntaa tai vaihtaa ilman vaikutusta ylempiin tasoihin. Perusteltuihin tasoihin jakaminen myös auttaa

tasojen toteuttamista, sillä näin tasoille voidaan antaa toteuttajikseen niihin erikoistuneet kehittäjät. (Clements ym. 2011, 89 – 91.)

Sovelluksen komponenttien riippuvaisuuksien löysentämiseksi ja muunneltavuuden helpottamiseksi sovelluksessa käytetään riippuvaisuusinjektiota (*Dependency Injection*). Se perustuu hallinnan muutossuunnan vaihdon (*Inversion of Control*) suunnitteluperiaatteeseen. Siinä objektin tarvitsemien palveluiden luonti ja syöttö on annettu siihen erikoistuneelle auktoritatiivisemmalle taholle eli injektorille. Tällöin noudatetaan samalla yhden vastuualueen periaatetta (*Single Responsibility Principle*). (Martin 2009, 157.)

Kyselysovelluksen riippuvaisuusinjektio on annettu kehyksen toimenkuvaksi. Se käyttää Aurynriippuvaisuusinjektoria palveluiden luomiseen ohjelman erilaisten komponenttien ja luokkien välillä. Riippuvaisuusinjektori voi esimerkiksi tutkia luokan konstruktorin ja syöttää sinne tämän luokan vaatimat riippuvuudet. Riippuvuuden ollessa rajapinta, injektorille annetaan erikseen käsky rajapinnan toteuttajasta.

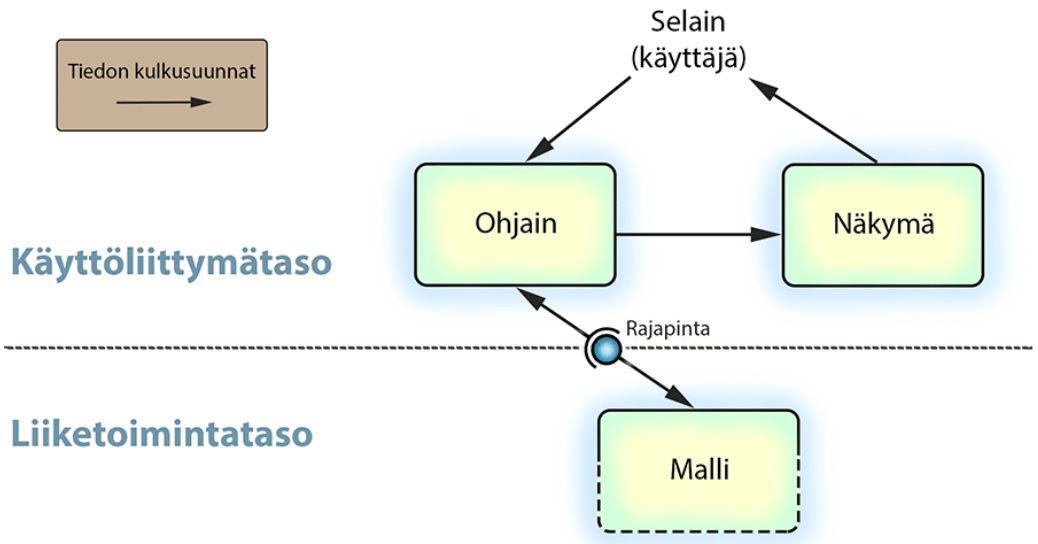
6.2.1 Sovelluskehys

Sovelluskehys on uudelleenkäytettävä runko johon koodia lisäämällä saadaan itsenäisesti toimiva sovellus (Koskimies & Mikkonen 2005, 188). Kyselysovelluksen kehyksen keskeisimpiin tehtäviin kuuluu siepata selaimelta HTTP-protokollan avulla tulevat GET- ja POST-kyselyt. Tämän jälkeen kehys reitittää sovelluksen suorituksen näiden kyselyiden mukaisiin MVC (*model-view-controller, malli-näkymä-ohjain*) -ohjaimiin (*controller*). Nämä se luo riippuvuusinjektorin avulla. Kehyksen vastuualueeseen kuuluu sovelluksessa myös PHP-istuntojen hallinta sekä käyttäjien tunnistus ja valtuutukset siirtyä pyydettyyn ohjaimeen.

6.2.2 Käyttöliittymätaso

Kyselysovelluksen vaatimuksiin kuuluu interaktiivinen selaimessa käyttäjän kanssa kommunikoiva käyttöliittymä. Kyselysovelluksen käyttöliittymänä toimii mukaelma malli-näkymä-ohjain -arkkitehtuurimallista. Ideana MVC-arkkitehtuurissa on erottaa käyttöliittymä eli käyttäjän syötteisiin reagoiva ohjain ja mallin (*model*) tietojen tulostamisesta vastaava näkymä (*view*) tietorakenteista eli mallista. Tämä erottaminen edesauttaa vastuualueiden jakoa. Mallin ja käyttöliittymän kehittäjillä ovat erilaiset syyt kehittää omaa osa-aluettansa. Käyttöliittymä voidaan esimerkiksi muuttaa tai vaihtaa kokonaan toisenlaiseen esimerkiksi eri sidosryhmille näiden vaatimuksiensa mukaan. MVC-arkkitehtuurissa ainoastaan käyttöliittymän pitäisi olla riippuvainen mallista eikä malli käyttöliittymästä. Tämä helpottaa esimerkiksi uuden käyttöliittymän liittämistä mallin sekä yleisesti mallin työstämistä ja testaamista. (Buschmann ym. 1996, 125 - 126; Fowler 2003, 330 - 331; Koskimies & Mikkonen 2005, 142.)

Kyselysovelluksen MVC-mukaelmassa (kuvio 10) ohjaimet toimivat liittimenä tietorakenteiden ja näkymän välillä. Kyselysovelluksen ohjaimien pääasiallinen velvollisuus on käyttäjän syötteiden sekä esimerkiksi kyselyn tilatiedon lähettäminen tietorakenteisiin. Se myös välittää tietorakenteiden lähettämät vastaukset näkymälle.

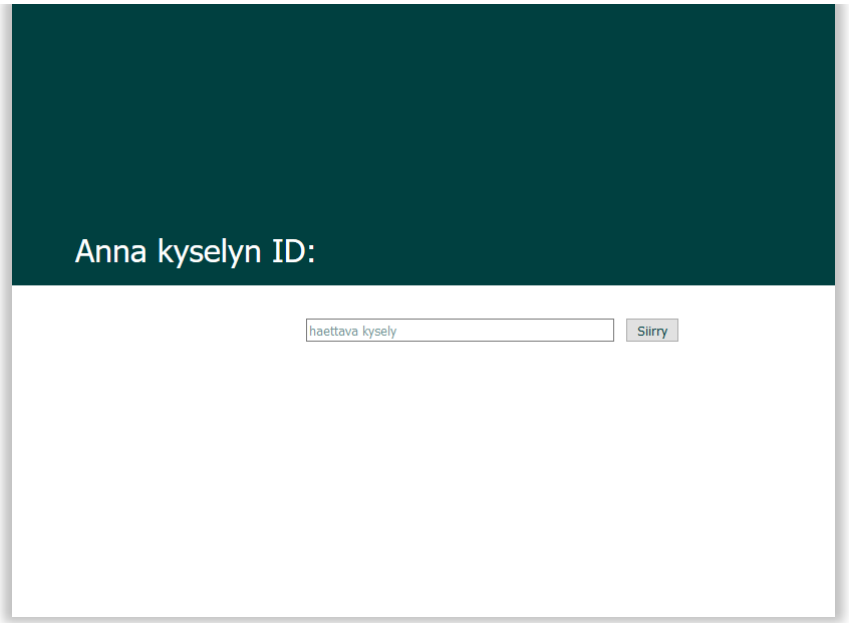


Kuvio 10: Kyselysovelluksessa käytetty MVC-arkkitehtuuri (Kettunen 2017)

Ohjaimen toimenkuvaan kuuluu myös käyttää kehyksen tarjoamaa uudelleenohjauspalvelua. Tämä palvelu toimii PRG (*POST-REDIRECT-GET*) -mallia käyttämällä. Tässä mallissa sovellus havaitsee esimerkiksi lähetetyn kaavakkeen ja tekee tarvitut toimenpiteet kuten tämän kaavakkeen tietojen tarkistuksen sekä tallennuksen tietokantaan. Lopuksi se ohjaa selaimen toiselle sivulle. Tämä estää kaavakkeen uudelleenlähetyksen ja ratkaisee ongelman, jossa kaavakkeen lähetyksen jälkeen selaimen ”päivitä ruutu”- tai ”takaisin”-napin painallus lähettäisi kaavakkeen tiedot uudelleen tietorakenteisiin ja edelleen uudelleen tietokantaan. Uudelleenohjaus toteutettiin käyttämällä ohjaimissa olevia GET- ja POST-kyselyihin erikoistuneita metodeja.

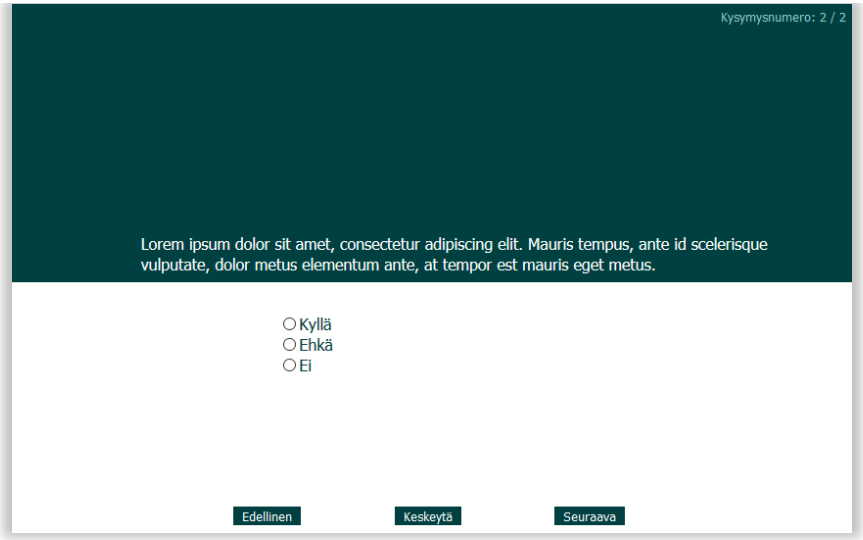
Ohjaimeen liitetyn näkymän pääasiallisena tehtävänä on valita mallin antamien tietojen perusteella oikea HTML-mallipohja, sijoittaa tiedot niille varatuille paikoille pohjassa ja lähettää tämä HTML-sivu selaimeen.

Kyselysovelluksen vastaajalle ensimmäinen näkyvä sivu on kyselyn hakusivu (kuvio 11). Tämä sivu joko näyttää kyselynhakulomakkeen tai ohjaa vastaajan suoraan kyselyn ”tervetuloa”-sivulle, jos sivun osoitekenttään on annettu järjestelmän tuntema GET-kyselytunnus.



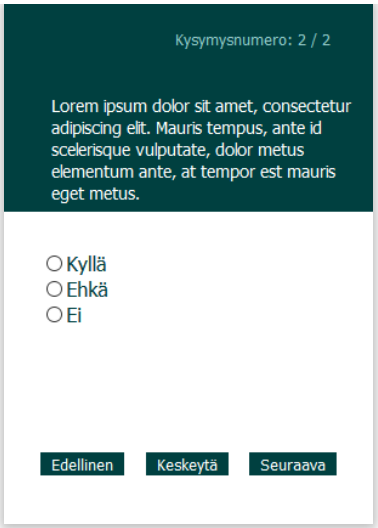
Kuvio 11: Kyselyn hakusivu

Erilaisia kyselysivuja sovelluksessa on kaksi: tekstikysymys sekä vaihtoehtokysymys (kuvio 12).



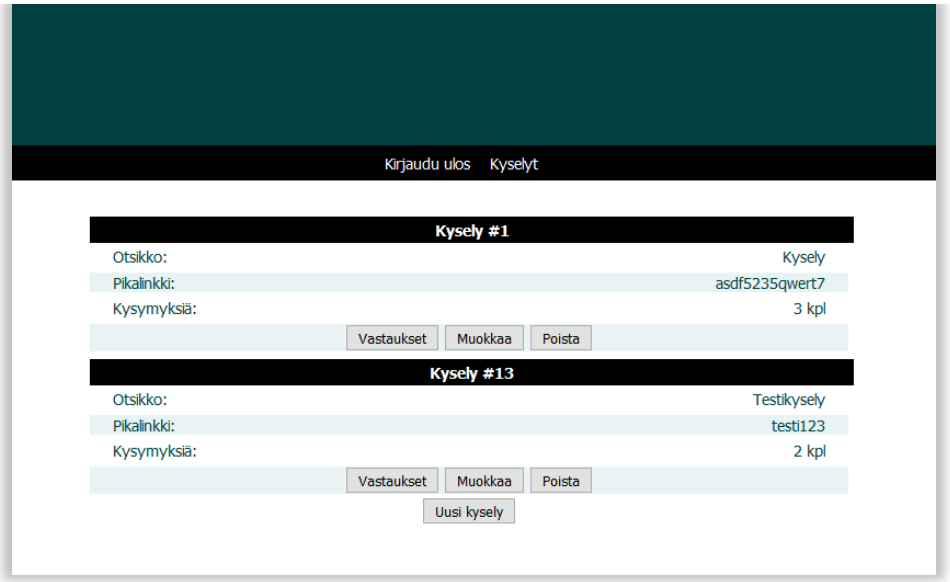
Kuvio 12: Kyselyn vaihtoehtokysymys -sivu

Kyselyn sivut muuttavat muotoaan responsiivisen sivusuunnittelun mukaisesti CSS-määritysten avulla käyttäjän selaimen ikkunan koon muuttuessa (kuvio 13).



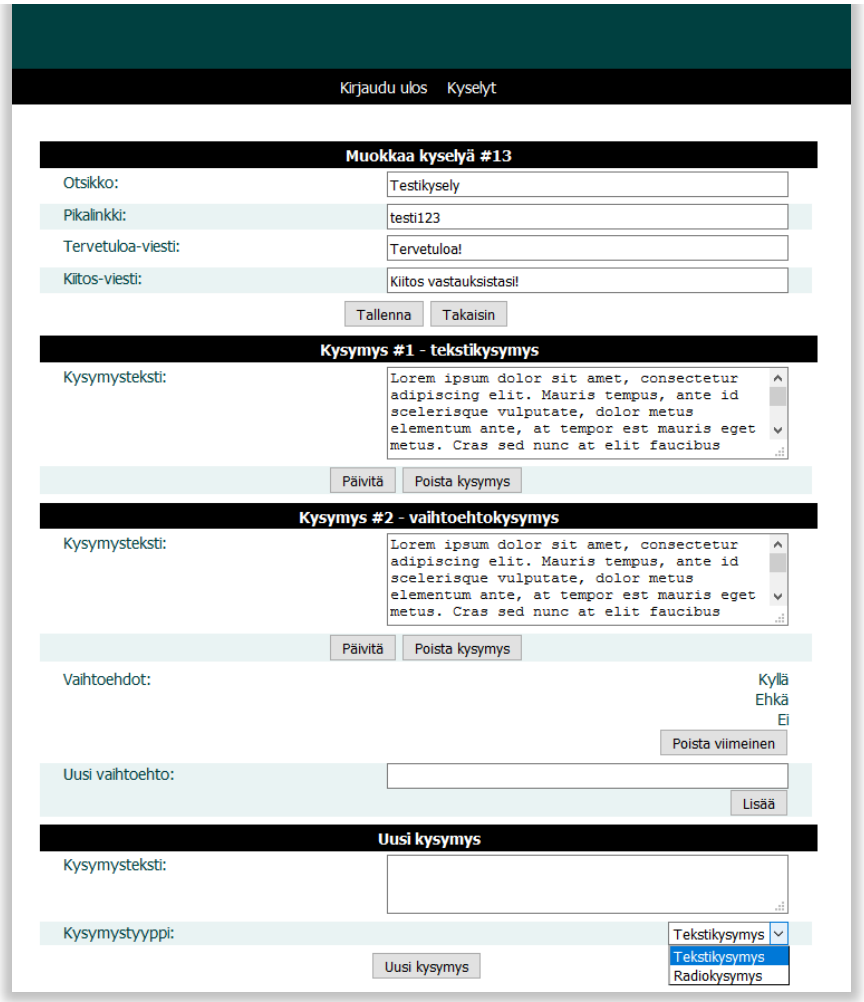
Kuvio 13: Kyselyn vaihtoehtokysymys pienemmässä selainikkunassa

Kyselyn lopuksi näytetään ”kiitos vastauksistasi” -sivu. Kyselijän luomia kyselyitä voidaan hallinnoida (kuvio 14) kirjautumista vaativien sivujen kautta.



Kuvio 14: Kyselyiden pääsivu, jossa on listaus käyttäjän luomista kyselyistä

Kyselyn muokkaus-sivulla (kuvio 15) voidaan muuttaa valitun kyselyn tietoja sekä lisätä, muokata ja poistaa kysymyksiä.

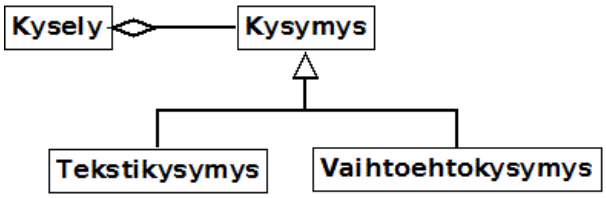


Kuvio 15: Kyselyn muokkaus -sivu

Kyselijän sivuilta löytyy myös mahdollisuus tarkastella vastaajien antamia vastauksia sekä poistaa kaikki annetut vastaukset tietokannasta.

6.2.3 Liiketoimintataso

Liiketoimintataso sisältää pääasiassa kyselyohjelman sisältämät käyttötapaukset eli niiden toiminnallisuuden, tietomallit sekä näihin tietomalleihin liittyvät palvelut. Esimerkkinä tietomallista on ”kysely” (kuvio 16) joka sisältää kyselyyn läheisesti liittyvät attribuutit ja metodit. Liiketoimintataso peittää tietovarastotason allensa ja käyttää tietovaraston tarjoamia rajapintoja tietomallien tietojen hakemiseen sekä viemiseen tietokantaan.



Kuvio 16: ”Kysely”-malli, joka voi sisältää erilaisia kysymyksiä

6.2.4 Tietovarastotaso

Tietovarastotaso sisältää logiikan tiedon tallentamiseksi ja hakemiseksi tietokannasta. Kyselysovelluksen tietokantana toimii MySQL, johon tietovarastotasolla sijaitsevat yhdyskäytävä-objektit pitävät yhteyttä PDO-rajapinnan avulla. Yhdyskäytävä-objektit toteuttavat rajapintoja, joita liiketoimintatason palvelut käyttävät.

Mahdollisen tietokannan vaihdon yhteydessä yhdyskäytävä-objektit vaihdetaan uutta tietokantaa tukeviksi versioiksi. MySQL-tietokannan suunnitteluun ja taulujen rakentamiseen käytin selaimessa toimivaa phpMyAdmin-sovellusta.

6.3 Sovelluksen testaus

Kyselysovelluksen toiminnallisuuden testaus tapahtui sovelluskehityksen edetessä sekä työn päätteeksi. Testaus työn kuluessa perustui yritys-erehdys-tapaan. Sovelluksen kehityksessä ei siten turvauduttu testivetoiseen ohjelmointiin.

Projektin lopuksi suoritettiin lopputestaus, jossa jokainen sovelluksen sivu ja niiden toiminto käytiin läpi eri selaimissa. Siinä varmistettiin, että kaikki määritellyt vaatimukset olivat toteutettu asianmukaisesti. Lopputestauksessa käytettyjä selaimia olivat pöytäkoneissa Firefox-, Chrome- sekä Microsoft Internet Explorer 10- ja Edge-selaimet. Android-käyttöjärjestelmää käyttävässä matkapuhelimessa testaukseen käytettiin Firefox- sekä Chrome-selaimia.

7 Yhteenveto ja oman oppimisen arviointi

Tämän toiminnallisen opinnäytetyön tarkoituksena oli rakentaa Internet-selaimissa toimiva kyselysovellus sekä esitellä sen arkkitehtuurisia ratkaisuja. Työssä oli myös tarkoitus selvittää, mitä ohjelmistoarkkitehtuuri on yleisellä tasolla sekä miten sitä sovelletaan sovelluksien kehittämisessä.

Kyselysovelluksen kehitysprojekti piti sisällään alkumäärittelyn, näiden määrittelyjen perusteella suoritetun alkusuunnittelun, toteutuksen sekä lopputestauksen. Sovelluksen kehitys

tapahtui useissa sykleissä, joissa uuden vaatimuksen ilmaannuttua palattiin määrittelyn kautta suunnitteluun ja edelleen toteutukseen.

Projekti alkoi perehtymisellä alan kirjallisuuteen sekä Internetistä saatavaan informaatioon aiheesta. Tiedon kerääminen jatkui läpi projektin, joka aiheutti osaltaan projektin viivästymistä. Tiedon lisääntyessä ja aiheen käydessä tutummaksi, löydettiin uusia lähestymistapoja sovelluksen rakenteellisiin ratkaisuihin. Alun perin suunniteltu muutaman kuukauden aikataulu venyi useilla kuukausilla.

Projekti opetti minua paljon sovelluskehittämisestä ja avarsi ymmärrystäni erityisesti arkkitehtuurista. Ohjelman kehittämisessä on tärkeää hyvä kokonaiskuva siitä, mitä on rakentamassa työn sujuvuuden ja selkeyden takia. Jokaisen sidosryhmän odotukset ja vaatimukset on yritettävä ottaa huomioon ohjelman suunnittelussa. Myös kommunikaatio näiden sidosryhmien kanssa on oleellista, jotta nämä pystyisivät tekemään työnsä oikein. Ohjelman kehitystyössä täytyy pystyä varautumaan muutoksiin oikeissa paikoissa rakenteita, sillä hallitsemattomana muutokset aiheuttavat ongelmia niin kehitys- kuin ylläpitovaiheessa.

Suuri aika projektissa meni sovelluskehyksen rakenteellisten ja arkkitehtuuristen ratkaisujen tekemiseen. Oman kehyksen rakentamisen suurimmaksi hyödyksi koin sen muodostumisen juuri omiin tarpeisiin sopivaksi. Näin pystyin varmistumaan sen oikeanlaisesta toiminnasta pinnan alla. Usein kuulee oman kehyksen rakentamisen olevan ”pyörän uudelleen keksimistä”. Tämä onkin useimmissa tapauksissa mielestäni totta. Vaatii paljon aikaa ja panostusta rakentaa toimiva ja käytännöllinen ratkaisu. Valmiissa kehyksissä ja komponenteissa päätökset ovat muotoutuneet toimiviksi ratkaisuiksi ajan saatossa. Tällöin loppukäyttäjän ei tarvitse välttämättä miettiä näitä ratkaistavia asioita niin paljon kuin puhtaalta pöydältä aloitettaessa. Uutta projektia käynnistäessä en välttämättä rakentaisi omaa sovelluskehystä. Valmiin kehyksen käyttäminen nopeuttaa huomattavasti sovelluksien kehitystyötä. Ongelmana mielestäni valmiissa kehysratkaisussa on, ainakin jos sovelluksen suunnittelutyö on tehty huonosti, sovelluksen liiallinen riippuvuussuhde kehykseen ja käytettyihin teknologioihin.

Kyselysovelluksen jatkokehitysaiheita heräsi paitsi projektin jälkeen, myös sen aikana. Koditasolla varauduttiin muun muassa uusien käyttäjien lisäämiseen järjestelmään sekä uusien kysymystyyppien lisäämiseen. Jatkokehitysaiheeksi nousi myös mahdollisuus näyttää erilaisia tilastotietoja vastauksista.

Jälkikäteen projektia ja siitä saatuja kokemuksia pohtiessani, voin todeta sen auttaneen minua suurin harppauksin eteenpäin sovelluskehityksen alalla. Se on lisännyt tulevaisuutta ajatellen tietotaitoani ohjelmiston kehitykseen liittyvästä teoriasta ja tekniikoista, joiden avulla toimiva sekä hallittava kokonaisuus voidaan rakentaa.

Lähteet

Painetut lähteet

Booch, G., Maksimchuk, R., Engle, M., Young, B., Conallen, J., Houston, K. 2007. Object-Oriented Analysis and Design with Applications. 3. painos. Boston: Addison-Wesley.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. 1996. Pattern-Oriented Software Architecture. Volume 1: A System of Patterns. Chichester: John Wiley & Sons.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P,. Nord, R. & Stafford, J. 2011. Documenting Software Architectures. Views and Beyond. 2. painos. Boston: Addison-Wesley.

Fowler, M. 2003. Patterns of Enterprise Application Architecture. 17. painos. Boston: Addison-Wesley.

Gorton, I. 2011. Essential Software Architecture. 2. painos. Berlin: Springer.

Heinisuo, R. & Rauta, I. 2007. PHP ja MySQL. Tietokantapohjaiset verkkopalvelut. 4. painos. Helsinki: Talentum.

Korpela, J. 2013. CSS3. Uudet mahdollisuudet. Jyväskylä: Docendo.

Korpela, J. 2014. HTML5-käsikirja. Jyväskylä: Docendo.

Koskimies, K. 2000. Oliokirja. Helsinki: Satku.

Koskimies, K. & Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Helsinki: Talentum.

MacDonald, M. 2014. HTML5: The Missing Manual. 2. painos. Sebastopol: O'Reilly.

Martin, R. 2009. Clean Code. A Handbook of Agile Software Craftmanship. Boston: Pearson Education.

McConnell, S. 2004. Code Complete. 2. painos. Redmont: Microsoft Press.

McFarland, D. 2013. CSS3: The Missing Manual. 3. painos. Sebastopol: O'Reilly.

Tatroe, K., MacIntyre, P. & Lerdorf, R. 2013. Programming PHP. 3. painos. Sebastopol: O'Reilly.

Vilkka, H. & Airaksinen, T. 2003. Toiminnallinen opinnäytetyö. Helsinki: Tammi.

Welling, L. & Thomson L. 2009. PHP and MySQL Web Development. 4. painos. Boston: Addison-Wesley.

Sähköiset lähteet

Bass, L., Clements, P. & Kazman R. 2013. Software Architecture in Practice. 3. painos. Boston: Addison-Wesley. E-kirja.

Brown, S. 2016. Software Architecture for Developers. Volume 1. Technical leadership by coding, coaching, collaboration and just enough up front design. Versio 15.8.2016. Leanpub E-kirja.

Korpela, J. 2003. CSS-termejä suomeksi ja englanniksi. Viitattu 5.10.2016. https://www.cs.tut.fi/~jkorpela/styles/termit.html

Korpela, J. 2005. Hyvin lyhyt johdatus SGML:ään. Viitattu 4.10.2016. https://www.cs.tut.fi/~jkorpela/sgml.html

Raggett, D., Lam, J., Alexander, I. & Kmiec, M. 1998. A history of HTML. Viitattu 4.10.2016. http://www.w3.org/People/Raggett/book4/ch02.html

Kuviot

Kuvio 1: Arkkitehtuuritaso on ylin abstraktio järjestelmästä (Kettunen 2017) 8
Kuvio 2: "Komponentti"–luokka on riippuvainen "Tietorajapinta"–rajapinnasta, ei rajapinnan toteuttavista vaihtoehtoisista luokista 10
Kuvio 3: Arkkitehtuurin suunnittelun vaiheet (mukailtu Gorton 2011, 98) 11
Kuvio 4: Esimerkki HTML-sivun DOM-puusta 13
Kuvio 5: Esimerkki PHP-koodista HTML-sivulla 14
Kuvio 6: "Auto" –luokka, jonka sisälle voidaan tuoda "Matkustaja" –luokka 15
Kuvio 7: Kyselysovelluksen käyttötapaukset 16
Kuvio 8: Sovelluksen rakenne sovelluksen vastaaja-osion osalta. Mukailtu Brownin (2016, 38) C4-arkkitehtuurimallista 18
Kuvio 9: Kyselysovelluksen tasot 19
Kuvio 10: Kyselysovelluksessa käytetty MVC-arkkitehtuuri (Kettunen 2017) 21
Kuvio 11: Kyselyn hakusivu 22
Kuvio 12: Kyselyn vaihtoehtokysymys -sivu 22
Kuvio 13: Kyselyn vaihtoehtokysymys pienemmässä selainikkunassa 23
Kuvio 14: Kyselyiden pääsivu, jossa on listaus käyttäjän luomista kyselyistä 23
Kuvio 15: Kyselyn muokkaus -sivu 24
Kuvio 16: "Kysely"–malli, joka voi sisältää erilaisia kysymyksiä 25

Taulukot

Taulukko 1: Kyselysovelluksen rajoituksia ja laadullisia vaatimuksia 17