Haaga-Helia
ammattikorkeakoulu Oy

Pienen mittakaavan budjetointiohjelma kotitalouskäyttöön

Nina Kovaljeff

20.11.2020

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Kovaljeff, Nina

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma (monimuoto)

Raportin/Opinnäytetyön nimi
Pienen mittakaavan budjetointiohjelma kotitalouskäyttöön

Sivu- ja liitesivumäärä
20 + 2

Tämän opinnäytetyön lopputuloksena syntyy Javascript-ohjelma, johon ohjelman käyttäjä pystyy merkitsemään tulonsa ja menonsa, pitämään yllä ostoslistaa ja koostamaan merkitsemistään tiedoista muutamia perustason raportteja. Ohjelmaan on toteutettu sekä back end että front end; ohjelmalla on SQL-tietokanta, REST-rajapinta ja CSS-muotoillut HTML-käyttöliittymät, joiden kautta käyttäjä pystyy sekä lisäämään tietoja tietokantaan että poistamaan niitä. Kyseessä on toiminnallinen opinnäytetyö, jolla ei ole toimeksiantajaa, mutta kaikki projektin osa-alueet on toteutettu ohjelmistoprojektien yleisia käytänteitä mukaillen.

Ohjelman lisäksi opinnäytetyön tuloksena syntyy raportti, jossa määritellään keskeiset käsitteet ja esitellään ohjelmalle asetetut vaatimukset, käydään läpi ohjelman toteutusta varten valitut työskentelytavat ja teknologiat ja esitellään valmiin ohjelman toimintaa. Raportti on toteutettu niin kutsutulla vetoketjumallilla, jossa jokaisessa luvussa esiteltävää aihetta tukee ensin lähteiden pohjalta kirjoitettu tietoperusta, ja tämän jälkeen käydään läpi käytännön toteutus opinnäytetyössä.

Asiasanat
CSS, Javascript, HTML, käyttöliittymäsuunnittelu, ohjelmointi, projektityö, rautalankaprototyypit, relaatiotietokannat

Sisällys

1 Johdanto

Omien menojen ja tulojen seuraaminen on monelle suorastaan elinehto, jotta talous pysyy tasapainossa ja etenkin menot kurissa tuloihin nähden. Hämmentävän usein henkilökohtainen menojen ja tulojen seuranta jää kuitenkin edelleen vanhanaikaisilta tuntuvien Excel-taulukoiden tai paperin ja kynän varaan. Budjetointiin tarkoitetut ohjelmat ovat järjestään lähinnä yrityskäyttöön suunnattuja, eikä henkilökohtaisia ratkaisuja taas ole tarjolla juuri muussa muodossa kuin mobiilisovelluksina.Tämän opinnäytetyön tarkoituksena onkin luoda omien menojen ja tulojen seurantaohjelma henkilökohtaiseen tarkoitukseen - eräänlainen paranneltu Excel siis, korvike niin hankalille taulukoille ja paperille ja kynällekin.

Työn tuloksena syntyy toisin sanoen yksinkertainen Javascript-ohjelma, johon ohjelman käyttäjä pystyy merkitsemään tulonsa ja menonsa, pitämään yllä ostoslistaa (lisäämään listalle kohteita ja poistamaan ne), sekä koostamaan joitakin perustason raportteja ohjelmaan aiemmin merkitsemistään tiedoista. Lisäksi työn tuloksena syntyy opinnäytetyöraportti, jossa käyn läpi ohjelman osien teoriaa sekä niiden käytännön toteutuksen ohjelmassa, ja esittelen ohjelman toimintaa kirjallisessa muodossa, edeten käyttöliittymästä tietokannan kautta ohjelmointityöhön valittujen teknologioiden esittelyyn. Opinnäytetyöraportin liitteitä tulevat olemaan esimerkiksi käyttöliittymäprototyypin suunnitelmat.

Opinnäytetyön tavoite on tuottaa toimiva lopputuote eli ohjelma, jossa

- on tiettyjä kehyksiä apuna käyttäen itse toteutettu käyttöliittymä, tietokantarakenne ja ohjelmakoodi (tarkat tiedot valituista teknologioista ja välineistä raportin luvussa 5)
- tieto kulkee käyttöliittymästä tietokantaan ja takaisin, ts. ohjelmaan on rakennettu tietokantayhteys ja rajapinta pyyntöjen toteuttamista varten
- on toteutettu käyttäjän sisään- ja uloskirjautumistoiminnot
- toteutuvat ohjelman vaatimusmäärittelyn mukaiset toiminnot (ks. luku 2)

Lisäksi opinnäytetyön tavoite on tuottaa eheä, opinnäytetyölle asetettuja vaatimuksia vastaava raportti, josta käy ilmi, miten ja millainen ohjelma on toteutettu, millaisia työkaluja ja teknologioita sen toteutukseen on käytetty, ja miksi olen valinnut ko. toteutustavat.

Tämän opinnäytetyön pääpaino on ennen kaikkea toiminnallisuudessa ja toimivan ohjelman tuottamisessa, joten työn rajauksen ulkopuolelle jäävät esimerkiksi sellaiset ohjelmointityöhön sinänsä läheisesti liittyvät asiat kuin testaaminen, tietoturva ja versionhallinta. Ohjelman aktiivinen testaaminen ohjelmointityön aikana on toki ollut välttämätöntä virheiden korjaamiseksi ja ohjelman toiminnan varmistamiseksi, mutta testaaminen ei itsessään ole yksi tämän työn painopisteistä, ja ohjelman ollessa hyvin

yksinkertainen ei kovin laajamittainen testaus ole muutenkaan kovin tarpeellista. Sama koskee myös tietoturvaa, sillä vaikka ohjelma vaatii sisäänkirjautumisen toimiakseen, on sillä tämän opinnäytetyön kannalta lähinnä demonstraatioarvoa ohjelman ollessa epäkaupallinen ja tällä hetkellä pelkästään opinnäytetyötarkoitusta palveleva.

Raportissa esiintyviä keskeisiä lyhenteitä ovat lyhyesti määriteltyinä seuraavat:
CSS (Cascading Style Sheet): HTML-sivujen ulkoasun muotoiluun käytetty kieli (HTML.com 2020).

HTML (Hyper Text Markup Language): Internet-sivujen ohjelmointiin käytetty kieli, joka mahdollistaa sisällöllisten ja toiminnallisten sivujen kirjoittamisen (esimerkiksi linkit toisille sivuille) (HTML.com 2020).

JSON (JavaScript Object Notation): Formaatti, jolla tietoa tuodaan ja viedään tietokannasta rajapinnan kautta ohjelmaan ja takaisin (ks. tämän opinnäytetyön tapauksessa kuva 6 luvusta 5) (MDN Web Docs 2020)

PHP (Hypertext Preprocessor): Skriptien eli komentosarjojen kirjoittamiseen käytetty kieli, jota käytetään yleisimmin erityisesti HTML-tiedostoihin upotettuna suorittamaan back end -toimintoja (esimerkiksi tietojen vienti ja tuonti) (PHP.net 2020.)

SQL (Structured Query Language): Tietokantojen luomiseen, ylläpitoon ja muokkaamiseen sekä tietokantakyselyihin käytetty kieli (Chapple 2020.)

2 Ohjelmointityön aloitus ja vaatimusmäärittely

Minkä tahansa tietojärjestelmän tai ohjelman suunnittelu ja toteutus - voisi sanoa, että koko ohjelmistoprojekti - lähtee liikkeelle tarpeesta: ilmenee asia tai ongelma, jonka hoitamiseen tai ratkaisuun koetaan hyväksi kehittää tietotekninen ratkaisu eli tässä tapauksessa tietojärjestelmä. Projekti alkaa tietenkin suunnitteluvaiheella, jossa nimetään projektiryhmä ja määritellään sen tarvitsemat resurssit, mutta ennen kuin koodia voidaan alkaa kirjoittaa, on myös tärkeää tietää, mitä projektiryhmän tarkalleen ottaen pitäisi saada aikaiseksi. Mitä projektiryhmä lähtee tekemään, millainen ohjelma projektin lopputuotoksena syntyy?

Näihin kysymyksiin vastaa ohjelman vaatimusmäärittely, joka on käytännössä lähes jokaisen ohjelmistoprojektin ensimmäinen askel (Chandramouli 2015). Toteutettavalle ohjelmalle asetetaan aivan aluksi sen käyttötarpeiden perusteella tiettyjä vaatimuksia, eli hyvin yksinkertaisesti ilmaistuna määritellään, mitä ohjelman tulee kyetä tekemään. Käyttäjäkunnasta ja sen laajuudesta riippuen erilaisia vaatimuksia on vähintään yhtä paljon kuin loppukäyttäjiäkin, ja toisaalta saattaa olla, että eri käyttäjien tulee kyetä tekemään samalla ohjelmalla eri asioita. Järjestelmänvalvoja tai muu ohjelmasta vastaava saattaa lisäksi tarvita ohjelmaan laajemmat käyttöoikeudet kuin ”rivikäyttäjä”. Hyvin usein vaatimusten lista (Scrum-projekteissa käytetään termiä ”kehitysjono”) elää projektin edetessä, mutta projekti tarvitsee lähtökohdakseen vähintään ” alustavasti parhaiten tunnetut vaatimukset” (Schwaber & Sutherland 2017), jotta se saadaan alkuun.

2.1 Vaatimusten esittäminen ja havainnollistaminen

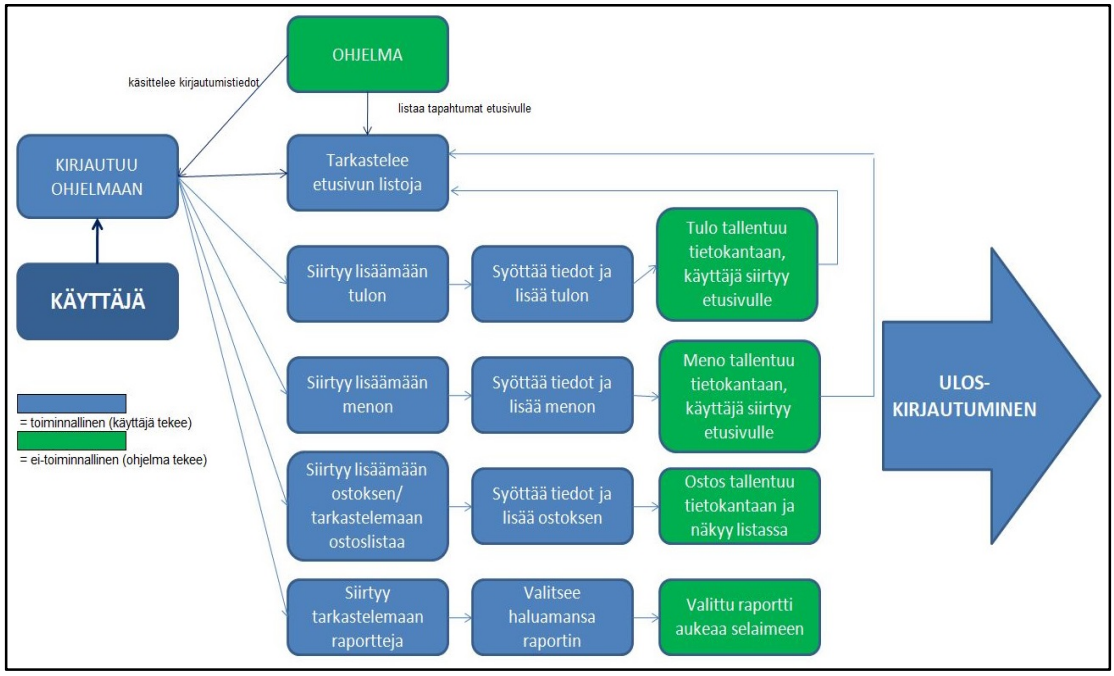
Vaatimukset esitetään ketterissä ohjelmistoprojekteissa kaikista tyypillisimmin käyttäjätarinoina, joista Mario Cardinal (2013) tarjoaa teoksessaan seuraavan, hieman HTML-lauseketta muistuttavan esimerkin: “As a **<role>,** I want **<desire>** so that **<benefit>**.” Suomennettuna ja toisin ilmaistuna kyseinen lainaus tarkoittaa, että vaatimuksia lähdetään tarkastelemaan ohjelman loppukäyttäjien ja heidän rooliensa kautta: mitä minun tulee pystyä roolissani järjestelmällä tekemään, ja miksi? Otan esimerkin oman opinnäytetyöni ulkopuolelta vaikkapa YouTube-videopalvelusta: ”tavallisena käyttäjänä minun on kyettävä merkitsemään video kirjanmerkkeihini, jotta pystyn katsomaan sen myöhemmin uudestaan”. ”Ylläpitäjänä minun on pystyttävä poistamaan loukkaava sisältö, jotta siitä ei koituisi harmia tai mielipahaa palvelun käyttäjille”. On tärkeää tunnistaa, millaisia käyttäjiä ohjelmalla tulee olemaan, ja mitä eri asioita heidän täytyy ohjelmalla pystyä tekemään (Cardinal 2013).

Koska tietojärjestelmälle asetettavien vaatimusten lista on usein varsinkin aivan järjestelmän toteutusprosessin aloitusvaiheessa pitkä ja monimuotoinen, on vaatimusmäärittely eli vaatimusten esittäminen ja tunnistaminen sekä priorisointi ja karsiminen tärkeää. Vaatimusten asettaminen tärkeysjärjestykseen helpottaa ohjelmointityön aloittamista ja etenemistä työjärjestyksen ollessa selvä, ja mahdollisimman tarkkaan jo alkuvaiheessa tehty vaatimusmäärittely vähentää viivästyksiä ja epäselvyyksiä ohjelman tarkoituksesta ja halutusta lopputuloksesta. Tämä on etenkin kaupallisissa ohjelmistoprojekteissa oleellista niin budjetin kuin kaikkia osapuolia tyydyttävään lopputulokseen pääsemisen kannalta.

Cardinal (2013) huomauttaa teoksessaan osuvasti kuitenkin myös, että ohjelmalle ja projektille alunperin asetetut vaatimukset mitä suurimmalla todennäköisyydellä muuttuvat projektin edetessä, ja hylkää hyvinkin suorasanaisesti ajatuksen ns. perinteisestä insinöörityöstä, jossa hyvin suoraviivaisesti ”vain” kehitetään ongelmaan tekninen ratkaisu (Cardinal 2013). Ohjelmistokehityksessä insinöörityö poikkeaa perinteisestä suurestikin: ohjelmointityö on luonteeltaan ennalta-arvaamatonta, ohjelman tilaaja ei useinkaan tiedä itsekään, mitä haluaa tai mitä todella tarvitsee, vaatimuksia saatetaan esittää tai muutella pitkin projektin kaarta, ja toisaalta myös esimerkiksi tekniset haasteet voivat pakottaa projektin muuttamaan kurssiaan. Ketterän menetelmän projektien vahvuus - ja myös toisaalta minkä tahansa ohjelmistoprojektin elinehto - onkin se, että se sopeutuu muuttuviin vaatimuksiin ja ikään kuin hyväksyy sen, että vaatimukset muuttuvat ja lopputuote saattaa vastata joihinkin hyvinkin erilaisiin vaatimuksiin kuin mitä sille alun perin asetettiin (ks. esim Schwaber & Sutherland 2017.)

2.2 Opinnäytetyön käyttötapaukset

Käyttötapauksella tarkoitetaan ohjelmistokehityksen kontekstissa kuvausta siitä, miten ohjelman käyttäjä suorittaa ohjelmassa tietyn prosessin (Visual Paradigm s.a.) Käyttötapauksen kuvaamisella (tekstinä tai visuaalisesti) pyritään selvittämään, mitä vaiheita prosessin suorittamiseen sisältyy, ja näin vastaamaan prosessin asettamiin vaatimuksiin ja välttämään mahdollisia virheitä. Vaatimuksia ja käyttäjätarinoita visualisoidaan usein myös käyttötapauskaavioina esimerkiksi kuvan 1 tapaan, mukaillen Bruce Powell Douglassin (2014) kaavioita:



Kuva 1. Käyttötapauskaavio

Käyttötapaus kuvataan askel askeleelta, mutta liiallisia detaljeja pyritään välttämään selkeyden vuoksi (Visual Paradigm s.a.) Esimerkiksi ylläolevaan kaavioon ei tarvitse erikseen sisällyttää jokaista napin painallusta, vaan käyttötapaukset kuvataan yleisellä tasolla.

Tämän opinnäytetyön lopputuloksena syntyvä ohjelma ei ole kaupallinen projekti eikä toteutushetkellä mitenkään laajempaan levitykseen tarkoitettu. Ohjelmalle asetettujen vaatimusten lista ei siis ole laaja ja keskittyy yksinkertaisiin perustoiminnallisuuksiin. Koska kyseessä on pitkälti henkilökohtaiseen käyttöön tarkoitettu järjestelmä, en myöskään tämän opinnäytetyön yhteydessä nähnyt tarpeelliseksi luoda järjestelmään erillistä admin- eli järjestelmävalvojan roolia, vaan keskityn tässä ainoastaan peruskäyttäjälle tarpeellisiin toimintoihin. Mikäli järjestelmänvalvojan roolille olisi tarvetta, voisi hän esimerkiksi hallinnoida muiden käyttäjien käyttäjätietoja (vrt. salasanojen resetointi tai tunnuksen poisto) ja oikeuksia, mutta kuten todettua, tällaiset laajennukset jäävät tämän opinnäytetyön rajauksen ulkopuolelle. Käyttäjätietoja ei myöskään tämän opinnäytetyön puitteissa ole erikseen lisätty tietokantaan, vaan ohjelmaan on testikäyttöä varten ”kovakoodattu” yksi käyttäjä, jolla on automaattisesti kaikki katselu- ja muokkausoikeudet.

Aloittaessani opinnäytetyöprojektia koin, että ohjelman (perus)käyttäjän on pystyttävä:

- merkitsemään menonsa ohjelmaan
- merkitsemään tulonsa ohjelmaan

- näkemään ajantasainen rahatilanteensa (ohjelmaan merkittyjen tietojen perusteella)
- koostamaan joitakin perustason raportteja menoistaan tai tuloistaan
- merkitsemään muistiin mahdollisia tulevia menoja, vrt. ostoslista tai säästötavoitteet
- poistamaan ostoslistan kohteita hankintojen jälkeen

Kyse on hyvin yksinkertaisesta ohjelmasta, eikä järjestelmän kautta ole tarkoitus pystyä esimerkiksi maksamaan laskuja tai tekemään tilisiirtoja. Mikäli tällaista rahavirtojen seurantajärjestelmää haluaisi kehittää edelleen eteenpäin, voisi käyttötapauksiksi ajatella myös sellaisia asioita kuin raporttien tulostaminen ja juurikin laskujen maksu, mutta tällaiset ohjelman itsensä ulkopuolelle ulottuvat toiminnot jäävät myös tämän opinnäytetyön ulkopuolelle.

2.3 Käyttäjätarinat ja niiden priorisointi

Sen kannalta, että *budjetointi*ohjelma palvelee tarkoitustaan, ovat ohjelman kannalta tärkeimmät vaatimukset seuraavat, prioriteettinsa mukaisessa järjestyksessä ja käyttäjätarinoiksi auki kirjoitettuina:

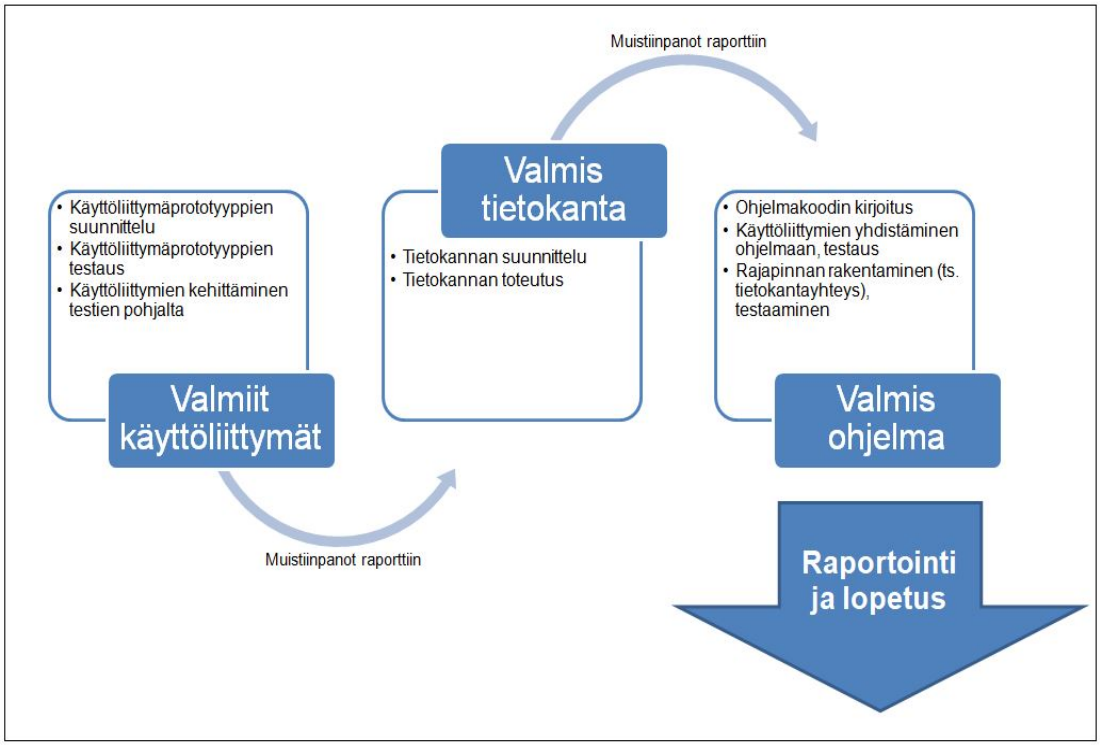
- Ohjelman käyttäjän on pystyttävä merkitsemään menonsa ohjelmaan, jotta hän pysyy menoistaan ajan tasalla ja pystyy seuraamaan niitä
- Ohjelman käyttäjän on pystyttävä merkitsemään tulonsa ohjelmaan, jotta hän pysyy ajan tasalla ansatuista tuloistaan
- Ohjelman käyttäjän on pystyttävä kokoamaan viikoittainen tai kuukausittainen raportti menoistaan, jotta hän pääsee paremmin selville rahankulutuksensa kokonaismääristä
- Ohjelman käyttäjän on pystyttävä kokoamaan vuosittainen raportti tuloistaan, jotta hän pääsee paremmin selville tuloistaan vuositasolla (mikä auttaa esimerkiksi veroprosentin määrittämisessä)
- Ohjelman käyttäjän on pystyttävä merkitsemään ohjelmaan muistiin mahdollisia tulevia hankintoja, eli pitämään yllä ostoslistaa, jotta hän pystyy säätelemään rahankulutustaan listan kohteiden hankkimiseksi ja asettamaan vaikkapa säästötavoitteita

Ohjelmaan lisäämäni ostoslistatoiminto on sinänsä ylimääräinen, ettei sillä ole suoraan mitään yhteyttä käyttäjän rahatilanteen seuraamiseen, joten tärkeimpien vaatimusten kärkipäähän ostoslistatoiminnallisuudet eivät yllä. Koen kuitenkin, että ostoslistan pitäminen voi kuitenkin auttaa potentiaalista käyttäjää ohjaamaan rahankäyttöään ja pitämään esimerkiksi säästötavoitteet mielessä, joten valmiiseen ohjelmaan ostoslistan pitämistoiminnallisuuskin sisältyy.

2.4 Projektin suunnittelu ja projektikaavio

Koska tämäkin opinnäytetyö on omanlaisensa ohjelmistoprojekti, oli sen etenemisjärjestys myös suunniteltava aivan sen alkuvaiheissa ennen kuin varsinaista ohjelmointityötä pääsi

tekemään pidemmälle. Perinteistä vesiputousmallia mukaileva kaavio omasta projektityöskentelystäni voisi näyttää seuraavanlaiselta:



Kuva 2. Opinnäytetyön ohjelmoinnin eteneminen

Huomionarvoista on, että käyttöliittymät on toteutettu ennen tietokantaa. Vaikka loogiselta tuntuisikin edetä ikään kuin pohjalta ylöspäin (tietokannasta ohjelmakoodin kautta käyttöliittymään), on käyttöliittymien suunnittelusta ennen tietokantaa se etu, että tämä auttaa hahmottamaan, mitä tietokannalta tarkkaan ottaen vaaditaan ja mitä se varastoi. Käyttöliittymän ja tietokannan ollessa olemassa on helpompaa rakentaa toimivaa ohjelmaa, ja toimintoja pääsee myös testaamaan paljon helpommin ja välittömämmin käyttöliittymän kautta kuin esimerkiksi JSON-lausekkeilla eli ”raakamateriaalilla”.

3 Käyttöliittymän suunnittelu

Käyttöliittymän tarkoitus on - kuten nimikin tavallaan kertoo - tehdä ohjelmasta käytettävä ja tuoda ohjelma käyttäjän ulottuville. Itse ohjelmakoodi ja sen toiminnallisuudet toimivat periaatteessa ilman käyttöliittymääkin, mutta ohjelman käyttäjälle olisi varsin tuskallista ja hidasta joutua lisäämään esimerkiksi tietoa tietokantaan kirjoittamalla raaka JSON-pyyntö tai SQL-lauseke joka kerta käsin. Käyttöliittymä on siis eräänlainen linkki käyttäjän ja varsinaisen ohjelman välillä. Käyttäjä välittää pyyntönsä ohjelmalle esimerkiksi täyttämällä lomakkeen ja painamalla lomakkeen lähetyspainiketta, ja näiden käyttäjälle näkyvien toimintojen taakse puolestaan kätkeytyy varsinainen ohjelmakoodi, joka käyttäjän pyynnön toteuttaa (ks. tämän ohjelman tarkempia teknisiä tietoja varten luku 5).

3.1 Käytettävyys vs. käyttäjäkokemus

Modernissa käyttöliittymäsuunnittelussa käytetään paljon termejä UI (User Interface) ja UX (User Experience), ja usein iloisesti sekaisin. Käyttöliittymä ja sen käytettävyys eivät ole sama asia kuin käyttäjäkokemus; voisi ajatella, että käyttöliittymä ja käyttäjäkokemus ovat ikään kuin pokeriautomaatti. Pokeriautomaatti laitteena on fyysinen käyttöliittymä, ja sen käyttäjäkokemukseen kuuluu puolestaan ominaisuus, jolla automaatti ehdottaa pelaajalle automaattisesti lukittavia kortteja mahdollisimman hyvän käden saamiseksi. Käyttöliittymä pyrkii ohjaamaan käyttäjää mahdollisimman paljon tiettyyn paikkaan ja säilyttämään käytettävyyden ”flown”, vaikka viime kädessä käyttäjä itse valitsee, mitä nappia painaa. Käyttäjäkokemuksella on toki muitakin ulottuvuuksia, mutta ytimekkäästi sen voi tiivistää ajatukseen siitä, miten helposti käyttäjä saa haluamansa asiat sivustolla tai järjestelmässä suoritettua.

Tämän opinnäytetyön puitteissa keskitytään ns. pelkästään peliautomaattiin ja sen rakentamiseen eikä varsinaisesti siihen, ehdottaako peliautomaatti käyttäjälle kortteja, joten syvemmälle UX-suunnitteluun en tämän raportin puitteissa pureudu. Westley Knight (2018) huomauttaa teoksessaan kuitenkin myös, että visuaalinen ulkoasu on itse asiassa tärkeä osa käyttäjäkokemusta: ”Good visual design enhances a user’s experience and builds their trust in the brand by focusing on aesthetics. It aligns the typography, colors, images, and other visual elements to help convey the content or function of the product”. Vapaasti suomennettuna palvelun miellyttävä visuaalinen ulkoasu siis paitsi parantaa palvelun käyttökokemusta ja vankistaa käyttäjän luottamusta siihen, myös parhaimmillaan auttaa selkiyttämään käyttäjälle tarjotun tuotteen tai palvelun konseptin ja tarkoituksen (Knight 2018). Käyttöliittymän ulkoasullakin on väliä sen kannalta, millainen käyttökokemus on. Pienillä ulkoasuteknisillä asioilla, kuten esimerkiksi painikkeiden ja

tekstikenttien sijoittelulla, olen itsekin pyrkinyt siihen, että käyttäjä ohjautuisi mahdollisimman jouhevasti sivustolla oikeaan aikaan oikeaan paikkaan ja olisi jatkuvasti selvillä siitä, mikä minkäkin asian tarkoitus on.

3.2 Rautalankaprototyypit ja testaus koehenkilöillä

Käyttötapausten perusteella tunnistettujen vaatimusten perusteella suunnittelin käyttöliittymäprototyypit ja toteutin prototyyppien luonnokset Microsoft Office Powerpoint -ohjelmalla. Kuten todettu, vaikka pääpainon tuleekin käyttöliittymäsuunnittelussa olla tietenkin siinä, että palvelu on selkeä ja yksinkertainen käyttää, on visuaalisuus myös tärkeää. Pyrinkin jo prototyyppejä suunnitellessani - sekä tietenkin myös valmista käyttöliittymää ohjelmoidessani - siihen, että käyttöliittymä on selkeä ja helppokäyttöinen, muttei kuitenkaan laimean tai jopa tylsän näköinen. Jo ensivaikutelma mistä tahansa verkkopalvelusta luo tietynlaisen pohjavireen asiointitapahtumalle, ja pyrin siihen, että käyttöliittymä on aina sisäänkirjautumissivusta alkaen miellyttävän pirteä, selkeä ja yhtenäinen, ensimmäisestä sivusta viimeiseen. Kaikki prototyyppiluonnokset löytyvät tämän raportin liitteistä (liite 1), mutta prototyyppien rakenteen havainnollistamiseksi esimerkkikuva etusivun ensimmäisestä prototyypistä alla (kuva 3):

Kuva 3. Etusivun rautalankaprototyyppi

Periaatteessa käyttöliittymän testaaminen koehenkilöillä ennen varsinaista julkaisua ei ole missään vaiheessa pakollista - etenkään tämän opinnäytetyön tapauksessa, sillä kyseessä ei ole laajaan levitykseen tarkoitettu ohjelma. Kuten Jessica Thornsby (2016) toteaa, käyttöliittymän testaaminen koeyleisöllä tai ”paperiprototypointi” ennen sen mahdollista laajempaa käyttöönottoa on kuitenkin yksinkertaisesti järkevää. Käyttöliittymää itse suunnitellessaan ja/tai tehdessään suunnittelija sekä sokeutuu helposti omille virheilleen että saattaa jäädä tarpeettoman paljon jumiin pelkästään omaan käsitykseensä siitä, millainen on käytettävyysasteeltaan hyvä ja myös hyvännäköinen käyttöliittymä. Koehenkilöillä testaaminen ja koehenkilöiden näkemysten tarkastelu auttaa huomaamaan sellaisia kehityskohteita ja jopa virheitä, jotka ilman testejä saattaisivat pahimmillaan tulla muuten ilmi vasta tuotantoon viennin jälkeen ja siten olla entistä hankalampia korjata (Thornsby 2016).

Tämän opinnäytetyönkin käyttöliittymää testautin muutamalla koehenkilöllä, ja näiden testien tarkoituksena oli kerätä kommentteja ja mahdollisia parannusehdotuksia lopullisen käyttöliittymän ohjelmointia ja parantelua varten. Yhdistetty kesäloma- ja korona-aika pitkine etäisyyksineen ja etätyöskentelyn vaatimuksineen aiheutti hieman hankaluuksia varsinaisten testitilanteiden järjestämiseen, mutta lopulta onnistuin järjestämään loka- ja marraskuulle kolme ns. live-testitilaisuutta ja keräämään hyvinkin välitöntä palautetta käyttöliittymän toimivuudesta testitilanteen edetessä. Koehenkilöt valikoituivat edellä mainittujen olosuhteiden vuoksi osin siksi, että he asuvat lähellä ja testaaminen ei edellyttänyt keneltäkään esimerkiksi matkustamista, mutta toisaalta myös koehenkilöiden kokemuspohja osui yksiin testauksen tarkoituksen kanssa. Ensimmäinen koehenkilöni työskentelee UX-suunnittelun ja itsepalveluportaalin käyttöliittymän kehittämisen parissa, toinen puolestaan on opiskellut graafista suunnittelua ja tehnyt myös käyttöliittymätoteutuksia, ja kolmas on liiketalouden tradenomi, joka tällä hetkellä suorittaa itsekin ohjelmistorobotiikasta korkeakouludiplomia.

Kaikilta kolmelta koehenkilöltä pyysin vastauksia seuraaviin kysymyksiin:

1. Millainen on yleisvaikutelma käyttöliittymästä ja sen ulkonäöstä?
2. Vaikuttaako käyttöliittymä helppokäyttöiseltä ja selkeältä? Ja kun liikut sivustolla, oletko jatkuvasti ns. kartalla siitä, mitä tapahtuu, vai jääkö tilanne hämäräksi?
3. Mikä käyttöliittymässä on hyvää, mikä huonoa? Muuttaisitko, lisäisitkö tai poistaisitko käyttöliittymästä/ohjelmasta jotakin?
4. Muita kommentteja käyttöliittymästä?

Kysymysten tarkoitus oli kerätä tietoa ja palautetta lähinnä käyttöliittymän visuaalisesta toteutuksesta ja siihen mahdollisesti tarvittavista muutoksista, koska sivustoni ei ole järin

monisyinen, eikä potentiaalisia ns. käyttäjäpolkuja ole tässä tapauksessa kovinkaan montaa. Halusin kuitenkin sisällyttää kysymykseen 2 myös UX-sävytteisen tarkennuksen siitä, onko ohjelma helppotajuinen ja onko käyttäjä ohjelmassa liikkuessaan jatkuvasti tilanteen tasalla, sillä tämä on ohjelman käytettävyyden kannalta oleellinen tieto ja saattaa toisaalta antaa aihetta myös käyttöliittymän muutoksiin (vrt. muutokset asetteluun, tekstien tai viestien tarkennukset).

3.3 Testien tulokset ja esitetyt parannusehdotukset

Pääosin käyttöliittymäprototyypit olivat testaajien mielestä selkeitä, eikä suuria muutoksen tarpeita tunnistettu. Prototyypissä alun perin käyttämieni värien (punertavan ruskea ja vihreä) todettiin kuitenkin jo aivan projektin alussa olevan väri- tai punavihersokeille henkilöille hankalasti hahmotettavia, ja lisäksi käyttöliittymän toteutuksen edetessä kävi nopeasti ilmi, ettei valitsemani fontti tai Google Fonts -kirjastossa tarjolla ollut samankaltainen fontti ollut tarpeeksi selkeä nopeaa hahmotusta varten. Kiinnitin itsekin nopeasti huomiota alkuperäisen käyttöliittymäprototyyppini lievään tympeyteen, ja hylkäsinkin sen nopeasti muuttaen käyttöliittymää prototyyppien teon ja ensimmäisten kommenttien jälkeen paljon vaaleampaan ja selkeämpään suuntaan.

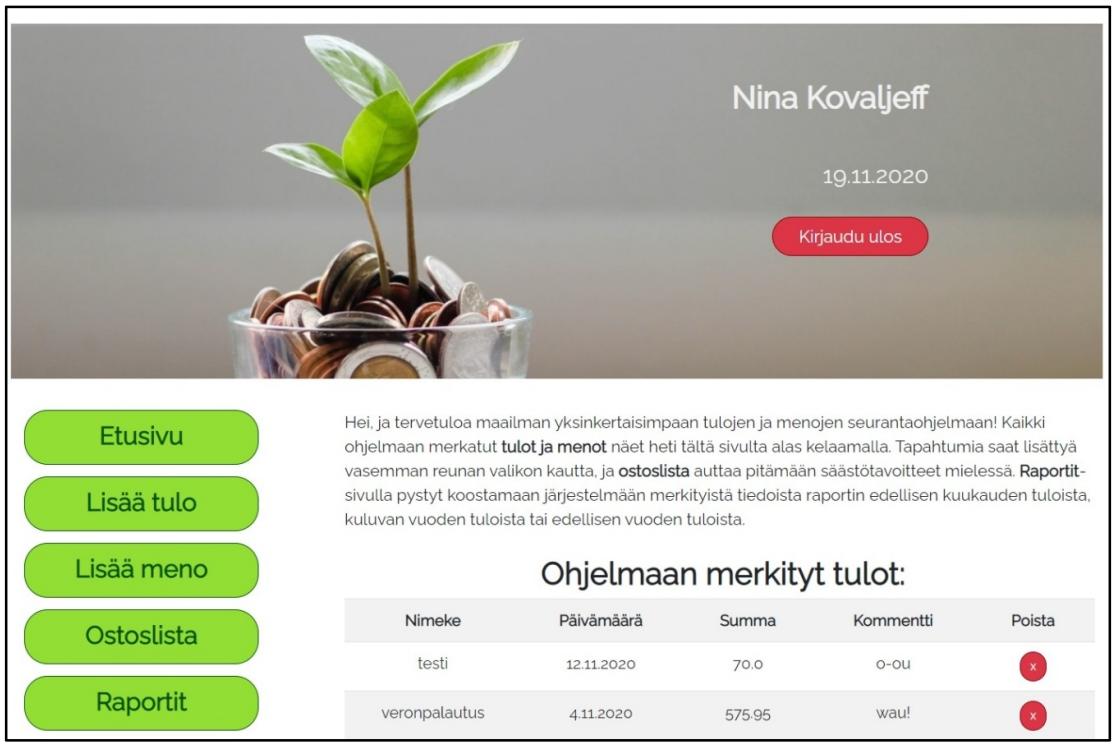
Lopullista käyttöliittymää (josta kuva 3 havainnollistaa etusivua, ks. myös kuvat 4, 5, 8 ja 9) koehenkilöt pitivät kaiken kaikkiaan raikkaan ja selkeän näköisenä. Suuria muutostarpeita nykyiseen käyttöliittymään ei juuri tunnistettu. Ainoastaan yksi koehenkilö ehdotti hyvinkin selkeitä asettelun muutoksia etenkin etusivulle, jossa hän olisi sijoittanut navigointipainikkeet bannerikuvan alle ja näin vapauttanut koko alla olevan tilan muulle sisällölle. Ensimmäinen koehenkilöistä totesi myöskin sivun olevan alkuperäisessä asussaan melko epäinformatiivinen - mikäli en olisi koetilanteen alussa selittänyt, mistä järjestelmässä on kyse, sen tarkoitusta olisi voinut joutua jonkin aikaa pohtimaan. Niinpä yritinkin seuraavia testejä varten lisätä esimerkiksi ohjetekstejä ja kuvauksia tehdäkseni niin sivusta kuin käyttötilanteestakin käyttäjälle mahdollisimman ymmärrettävän ja sujuvan; käyttöliittymätesteistä tuli siten myös Scrum-tyyliin aavistuksen iteratiivisia.

Eniten käyttöliittymätesteissä tuli esille sellaisia toiminnallisuuksien muutostarpeita tai kehitysehdotuksia, jotka liittyvät ennen kaikkea ohjelman mahdolliseen jatkokehitykseen tämän opinnäytetyön rajausta pidemmälle. Koehenkilöt pohtivat esimerkiksi mahdollisuuksia hakea tai luokitella merkintöjä (etenkin menoja), mobiilisovellusta ja mahdollisuutta merkitä ostoslistan kohteita suoraan menoiksi hankintojen jälkeen. Tällaiset toiminnallisuudet eivät liity niinkään käyttöliittymään kuin ohjelman itsensä toimintaan, ja jäävät tämän opinnäytetyön rajauksen ulkopuolelle, mutta erään

koehenkilön sanoin tämä opinnäytetyö tarjoaa toteutetussa muodossaan jo eräänlaisen rungon, joka olisi pienellä jatkokehityksellä hyvinkin tuotteistettavissa.

3.4 Valmis käyttöliittymä

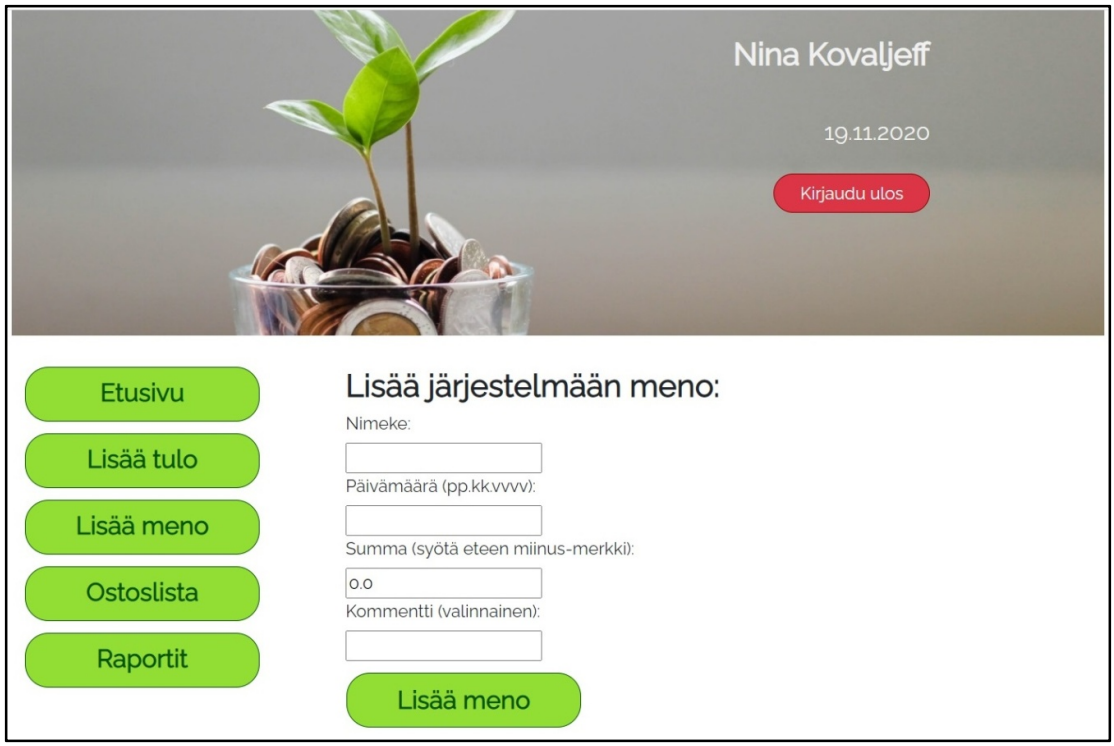
Tähän raporttiin en sisällytä kaikkia käyttöliittymäkuvia, mutta kuva 4 näyttää lopullisen version käyttöliittymän etusivusta (sisäänkirjautumisen jälkeen) ja kuva 5 havainnollistaa myöhemmin menon lisäyssivua, joka on tulon lisäyssivun kanssa miltei identtinen.



Kuva 4. Valmiin käyttöliittymän etusivu

Ohjelman valmis käyttöliittymä ei suurilta osin poikkea prototyypeistä, etenkään sisällöllisesti. Sisältö on aseteltu hyvin pitkälti samalla tavalla ja on samanlaista kuin prototyypissä. Suurimmat muutokset näkyvät lopulliseen versioon valituissa väreissä, fontissa sekä esimerkiksi painikkeiden ulkonäössä. Lopullisesta käyttöliittymästä jätin sivun vaaleanvihreän taustavärin kokonaan pois ja yritin näin edelleen rauhoittaa ja pelkistää sivun ulkoasua sekä välttää ns. tunkkaisuutta. Myöskin painikkeiden ja taulukoiden ulkoasut muuttuivat prototyypissä esitellystä. Osin tämä johtuu tietenkin siitä, että aivan täysin Microsoft Office -tyyliseen ratkaisuun ei ollut mieltä pyrkiä, mutta myös ohjelmointiteknisistä syistä. Esimerkiksi painikkeet löytyivät ns. valmisratkaisuna, jonka lähdekoodia muokkaamalla sain lopulta aikaan miellyttävän, käyttöliittymään hyvin sopivan lopputuloksen.

Niin käyttöliittymäprototyypeissä kuin valmiissa käyttöliittymissäkin pyrin alusta lähtien yhtenäisyyteen sekä jatkuvuuteen niin, että niin ohjelman ulkoasu kuin käyttökokemuskin olisi selkeä, sulava ja miellyttävä. Sivun yläreunan bannerirakenne kuvineen ja uloskirjautumispainikkeineen sekä vasemman reunan navigaatiopainikkeet toistuvat samanlaisina samoilla paikoilla kautta sivuston. Muun sisällön, kuten lomakkeet ja taulukot, olen pyrkinyt asettelemaan kullakin sivulla samaan kohtaan niin, etteivät näiden osien marginaalit banneriin ja navigaatiopalkkiin olisi keskenään erilaisia. Kyse on hyvin pienistä asioista, ja itse ohjelman käytön kanssa sisällön marginaaleilla tai painikkeiden muodolla ei ole mitään tekemistä. Käyttäjän käyttökokemuksen kannalta hyvinkin pienillä asioilla voi kuitenkin olla merkitystä - pienet asiat voivat tehdä ohjelmasta entistä paremmin käytettävän tai toisaalta nousta käyttäjälle syystä tai toisesta hyvinkin häiritseviksi.



Kuva 5. Menon lisäyssivu

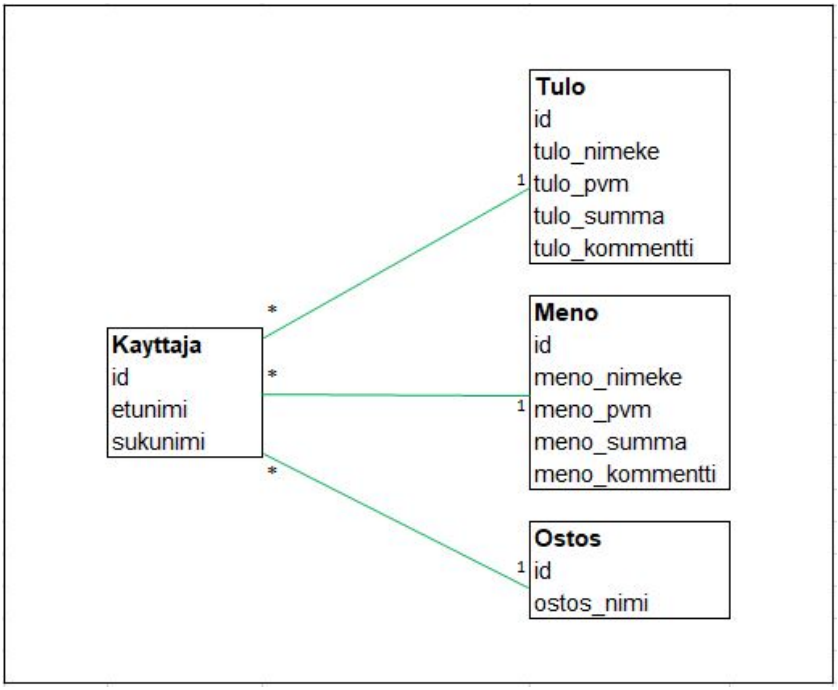
4 Tietokannan suunnittelu

Missä tahansa tietokoneohjelmistossa on hyvin karkeasti ajatellen kolme kerrosta: käyttöliittymä, jonka kautta ohjelman käyttäjä (useimmissa tapauksissa) ohjelmaa käyttää, varsinainen ohjelmakoodi, joka suorittaa pyydetyt toiminnot, ja tietovarasto eli tietokanta, joka säilöö ohjelman käyttämää ja käyttäjän tarvitsemaa tietoa. Tieto kulkee käyttöliittymästä tietokantaan ja takaisin erikielisten pyyntöjen ja vastausten välityksellä (tarkempaa tietoa tämän ohjelman toteutuksesta luvussa 5), ja tietokantaa voisi ajatella staattisena varastona, ”kirjahyllynä”, josta dynaaminen ohjelmakoodi kirjastonhoitajan tapaan noutaa vaadittua tietoa käyttäjälle ja tallettaa uutta tietoa.

Vaikka tietokanta itsessään onkin staattinen eli ikään kuin liikkumaton, on senkin oltava käytettävä, ja sen suunnitteluun on myös uutta järjestelmää luotaessa panostettava. George Tillman (2017) kirjoittaa teoksessaan osuvasti siitä, miten tärkeää huolellinen tietovarastorakenteen suunnittelu on, ja miten valitettavan usein ”sinne päin” toteutettu tietokantarakenne johtaa kuitenkin siihen, että tietokanta tai ohjelma toimii -mutta vain päällisin puolin (Tillman 2017). Mitä enemmän tietoa on varastoitava ja mitä monimutkaisempia yhteyksiä järjestelmän eri komponenttien ja tietokannan taulujen välillä on, sitä tärkeämpää on, että järjestelmän vaatima tieto on varastoitu huolellisesti ja sen käytettävyyden kannalta loogisesti. Myös tietokannan käytettävyydellä on väliä siinä missä käyttöliittymänkin, vaikkei tietokanta ohjelmassa päälle päin varsinaisesti näykään. Huonosti jäsennelty tai varastoitu tieto hankaloittaa minkä tahansa ohjelmiston kehittämistä ja toimintaa samaan tapaan kuin esimerkiksi sotkuinen kirjahylly hankaloittaa oikean kirjan etsimistä ja löytämistä.

4.1 Tietokannan rakenne

Tämän opinnäytetyön tietokannan suunnittelun aloitin käyttöliittymäprototyyppien valmistumisen ja testauksen jälkeen, varsinaisen käyttöliittymäohjelmoinnin yhteydessä. Samoin kuin käyttöliittymän, pyrin pitämään tietokannankin rakenteen mahdollisimman yksinkertaisena, jotta kannassa ei olisi mitään ohjelman kannalta epäoleellista ja siten niin ohjelmointityötä kuin ohjelman käyttöäkin mahdollisesti hankaloittavaa. Ohjelmani käyttämä tietokanta on ohjelman suppeudenkin vuoksi hyvin pelkistetty ja sisältää vain kolme taulua - Tulo, Meno ja Ostos. Tietokantakaavioon (kuva 6) olen kuitenkin sisällyttänyt selvyyden vuoksi hyvin löyhän taulun myös ohjelman käyttäjälle (Kayttaja), jotta saisin esitettyä käyttäjän ja käyttäjän kirjaamien tapahtumien suhteet.



Kuva 6. Opinnäytetyön tietokanta UML-luokkakaaviona

Kuten kaaviosta käy ilmi, yhdellä tulolla, menolla tai ostoksella voi olla vain yksi käyttäjä, mutta käyttäjä voi tietenkin kirjata itselleen monta tuloa, menoa tai ostosta. Kyseessä on siis kaikissa kolmessa tapauksessa yhden suhde moneen -suhde. Mikäli ohjelma olisi laajalevikkisempi ja toimintoja enemmän, sisältäisi se todennäköisesti enemmän tietoa etenkin käyttäjästä ja myös huomattavasti enemmän relaatioita (ostoslistan kohteita voisi esimerkiksi hankintojen myötä siirtää suoraan menoihin). Tämän kaltaiset toiminnot jäävät kuitenkin tämän opinnäytetyön rajauksen ulkopuolelle, ja tällaisenaan tietokanta kirjaimellisesti lähinnä varastoi käyttäjän syöttämää tietoa.

Vaikka tietokanta onkin tässä tapauksessa hyvin yksinkertainen, pyrin tämän kappaleen johdantoon viitaten olemaan senkin suunnittelussa mahdollisimman huolellinen ja nimeämään esimerkiksi attribuutit niin, että rakennekaaviota tai seuraavaksi esiteltävää tietohakemistoa katsomalla on heti selvää, mistä objektista on kyse. Pyrin myös välttämään tuplatietoa ja toisaalta jakamaan attribuutit järkeviin osiin niin, että attribuutteihin on helppoa viitata, mutta toisaalta käyttäjältä pyydetään ja tietokantaan tallennetaan vain oleelliset syötteet.

4.2 Tietohakemisto

Tietohakemisto sisältää tarkemmat tiedot siitä, millaista tietoa tietokantaan tallennetaan; siinä, missä esimerkiksi kuvan 6 tietokantakaavio keskittyy kuvaamaan tietokannan rakennetta visuaalisesti, tietohakemisto kertoo, mitä ja minkä muotoista tietoa

tietokantaan tallennetaan. Tietohakemistosta käy ilmi esimerkiksi se, montako merkkiä tekstikenttään syötettävä teksti saa maksimissaan sisältää (esimerkiksi ”varchar(250)” tarkoittaa, että tekstissä saa olla mitä tahansa merkkejä, ja se saa olla maksimissaan 250 merkkiä pitkä), ja onko jonkin kentän täyttäminen eli arvon antaminen pakollista (NOT NULL).

Tiedon syöttämisen ehtoja on noudatettava tarkoin, ja väärin syötetty tai puutteellinen tieto johtaa yleensä heti ohjelman virheeseen tai poikkeuksenkäsittelyn mahdollisesti puuttuessa ohjelman kaatumiseen. Otetaan tietohakemistosta esimerkki: tulon tallentaminen tietokantaan vaatii käyttäjältä tulon nimekkeen, päivämäärän ja summan. ID:n ohjelma generoi automaattisesti, ja kommentille ei ole annettu tietotaulua luotaessa NOT NULL -rajoitus, joka tarkoittaisi automaattisesti sitä, ettei kenttä saa jäädä tyhjäksi. Mikäli käyttäjä kuitenkin yrittäisi syöttää tulon vaikkapa ilman päiväystä, ilmoittaisi ohjelma käyttäjälle virheestä välittömästi, sillä kentälle on annettu NOT NULL -rajoitus.

Tulo

Kenttä	Tyyppi	Kuvaus
id	int PK NOT NULL	Tulon ID
tulo_nimeke	varchar(50) NOT NULL	Tulon nimi/selite (esim. ”palkka”)
tulo_pvm	varchar(10) NOT NULL	Tulon päivämäärä, kirjaajan mieltymyksistä riippuen joko maksu- tai kirjauspäivä
tulo_summa	double NOT NULL	Tulon summa
tulo_kommentti	varchar(255)	Oma kommentti, ei pakollinen

Meno

Kenttä	Tyyppi	Kuvaus
id	int PK NOT NULL	Menon ID
meno_nimeke	varchar(50) NOT NULL	Menon nimi/selite (esim. ”sähkölasku”)
meno_pvm	varchar(10) NOT NULL	Menon päivämäärä, kirjaajan mieltymyksistä riippuen joko suoritus- tai kirjauspäivä
meno_summa	double NOT NULL	Menon summa (täytyy olla aina negatiivinen)
meno_kommentti	varchar(255)	Oma kommentti, ei pakollinen

Ostos

Kenttä	Tyyppi	Kuvaus
id	int PK NOT NULL	Ostoslistan kohteen ID
ostos_nimi	varchar(250) NOT NULL	Ostoslistan kohteen nimi/selite (esim. ”uusi takki”)

5 Järjestelmän toiminta ja keskeisimmät tekniset ratkaisut

Erilaisia teknologiavaihtoehtoja ohjelmistokehitykseen on lukuisia, ja eri tekniikat ja toteutustavat palvelevat myös erilaisia tarkoituksia. Niin ohjelmointikielissä, niihin perustuvissa sovelluskehyksissä kuin käyttöliittymäkirjastoissakin on valinnanvaraa; back end -ohjelmoijat suosivat esimerkiksi sellaisia kehyksiä kuin Node.js ja Angular, ja front end -kehittäjät puolestaan hyödyntävät käyttöliittymäkehityksessä nykyisin erityisesti React-kirjastoa (Gupta 2020). Työkalujen valintaan kulloisessakin ohjelmistoprojektissa vaikuttavat paitsi niiden käyttäjien omat vahvuudet ja mieltymykset, myös niiden lopullinen käyttötarkoitus. Tässä luvussa käyn läpi opinnäytetyötä varten valitsemani teknologiat sekä perustelen hieman valintojeni taustoja.

5.1 Käytetyt menetelmät, sovelluskehykset ja kirjastot

Itse ohjelma on toteutettu Javascript-ohjelmointikielellä Spring Boot -sovelluskehystä käyttäen. Käyttöliittymät on ohjelmoitu HTML-kielellä, ja ulkoasun muotoilut toteutettu CSS:llä Bootstrap-tyylikirjastoa sekä Google Fonts -fonttikirjastoa hyödyntäen, CSS-muotoilujen ollessa upotettuina suoraan HTML-tiedostoihin. Ohjelman tietokantana toimii PostgreSQL-tietokanta, jonka kanssa ohjelma kommunikoi Rest API-rajapintaa hyödyntäen JSON-kielellä (ks. myös kuva 7). Alkuperäinen aikomukseni oli käyttää H2-tietokantaa sen keveyden ja toisaalta myös aiemman kokemuksen vuoksi, mutta työn edetessä kävi ilmi, että PHP-skriptien kirjoittaminen H2-kantaa varten oli todella hankalaa, joten eri lähteitä ja vaihtoehtoja tutkittuani päädyin lopulta vaihtamaan tietokannan kokonaan toiselle alustalle.

Ohjelma itsessään on toteutettu Eclipse IDE -ohjelmalla ja Javascript-ohjelmointikielellä. HTML-kieliset käyttöliittymät on toteutettu Visual Studio Code -ohjelmalla, taustakuvan ollessa kuvapankkikuva. Käyttöliittymätestit on toteutettu koehenkilöiden kanssa kasvokkain ja testitilaisuudet on nauhoitettu; joitakin yksittäisiä kommentteja prototyypeistä ja käyttöliittymistä on myös pyydetty kirjallisena projektin eri vaiheissa, mutta näitä en laske käyttöliittymätestauksen piiriin. Käyttämääni projektimenetelmää voisin kuvata lähinnä Scrum-tyyppiseksi menetelmäksi sprintteineen, vaikka aivan täydellistä sprinttimäistä rytmitystä en projektille pystynytkään aikatauluhaasteista johtuen toteuttamaan. Scrum-menetelmää mukaillen ja myös oman työni helpottamiseksi pidin kuitenkin kiinni työskentelytavasta, jossa aina tietyn palan ohjelmaan toteutettuani kirjoitin siitä myös raporttiin. Ohjelmointipäivät toimivat sprintteinäni, ja raportin kirjoittaminen sprinttikatselmointinani.

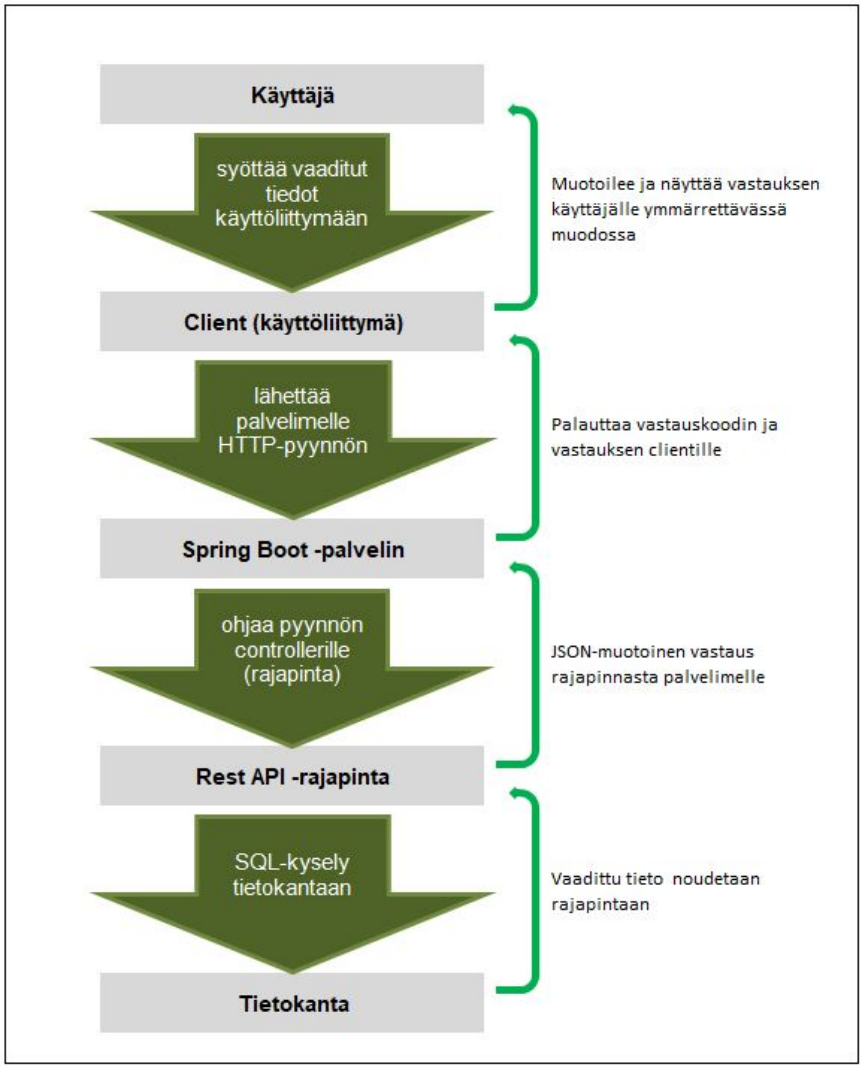
Sovelluskehyksen käyttöä itsessään on valintana varsin suoraviivaista perustella sillä, miten paljon se helpottaa ja nopeuttaa ohjelman kirjoittamista sovelluskehyksen tarjotessa ohjelmalle valmiina tietyt, samanlaisina ohjelmasta toiseen toistuvat rakennuspalikat. Kuten aiemmin mainittu, niin sovelluskehyksiä kuin myös ohjelmointikieliäkin on back end -kehitykseen tarjolla lukuisia. Eri lähteisiin ja vaihtoehtoihin tutustuttuani koin kuitenkin parhaaksi turvautua omassa ohjelmointityössäni Spring Boot -kehyksen ja Javascriptin yhdistelmään, sillä työn tavoite tai tarkoitus ei ollut toteuttaa erityisesti skaalautuvaa tai kovin monitahoista järjestelmää, vaan panostaa ennen kaikkea toimivan ohjelman aikaansaamiseen ja myös oman osaamiseni näyttämiseen ja edelleen kehittämiseen.

Edellä mainittu pätee myös käyttöliittymän toteuttamiseen ja sellaisten ulkoasuun liittyvien ratkaisujen kuin fontti- ja tyylikirjaston valintaan. Kuten aiemmin mainittu, React on nykyisin front end -kehittäjien suosikkikirjasto käyttöliittymien rakentamiseen. Sen käyttökohde ovat kuitenkin ennen kaikkea sellaiset kevyet, responsiiviset web-sovellukset, joiden on pystyttävä käsittelemään nopeasti muuttuvia tilanteita ja tietoa (Gupta 2020.) Oma ohjelmani puolestaan on melko staattinen eikä sen tarvitse skaalautua esimerkiksi laitteelta toiselle, joten tarvetta muulle kuin hyvin perinteiselle HTML:n ja CSS:n yhdistelmälle ei ollut - mikäli ohjelmastani aikoisi toteuttaa esimerkiksi mobiiliversion, voisi React Nativen kaltainen mobiiliohjelmointikehys tulla erittäinkin ajankohtaiseksi.

Vaikka käyttöliittymäni perustuikin ns. ”karvalakkiratkaisuihin”, vaikutti valintaan myös se, että HTML-sivut taipuivat integroitujen CSS- ja PHP-lauseiden sekä Javascriptin avulla käytännössä kaikkeen tarvitsemaani. Näin sain pidettyä ohjelman tiedostorakenteen miellyttävän yksinkertaisena, ja muutosten tekeminen ja testaaminen kävivät helpommin. Käyttöliittymän ohjelmointi helpottui myös huomattavasti Bootstrap-kirjaston tarjotessa tiettyjä elementtejä (esimerkiksi taulukot ja napit) hyvin perustasolla valmiina, kun kaiken rakentamista ei tarvinnut aloittaa aivan alusta. Toisaalta Bootstrapin tarjoamat ”rakennuspalikat” olivat CSS:n avulla myös muunneltavissa melko lailla loputtomiin, mikä teki niistä tämän projektin kannalta paitsi helposti käytettäviä, myös helposti muokattavia. Kaikkiin valintoihini vaikutti edellä mainittujen tekniikoiden yleisyys, joka teki niin tiedon kuin avunkin etsimisen ongelmatilanteissa helpommaksi.

5.2 Ohjelman tekninen toiminta lyhyesti

Ohjelman tärkeimmät tekniset komponentit ja niiden väliset yhteydet on esitelty alla olevassa kaaviossa kenties kaikista ymmärrettävimmässä muodossaan (kuva 7):

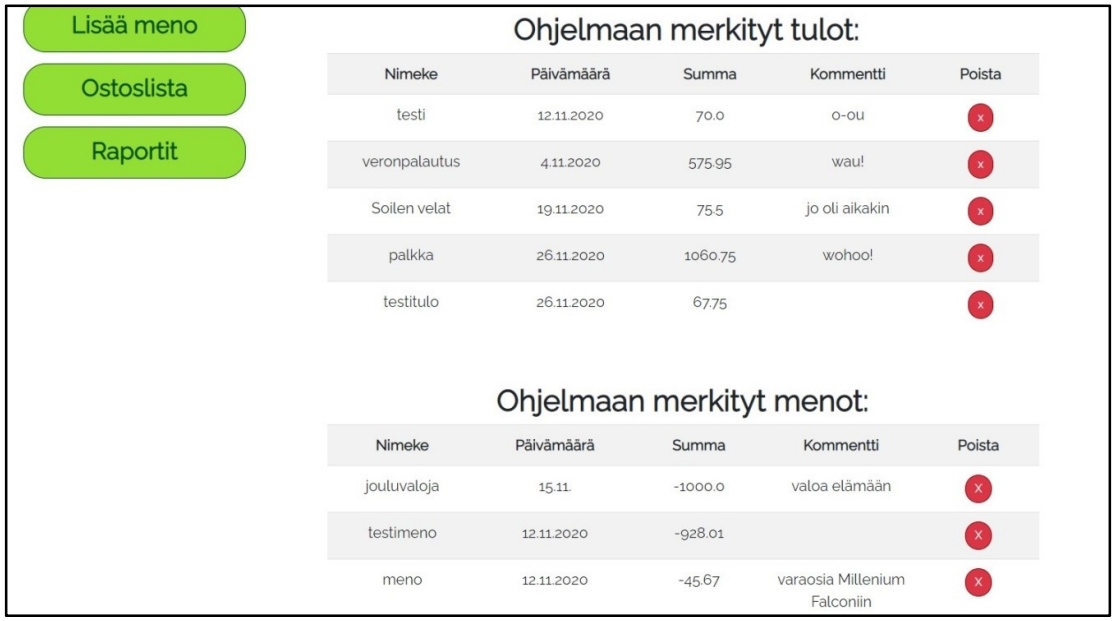


Kuva 7. Järjestelmän keskeisimmät komponentit ja niiden väliset yhteydet

Ohjelma vaatii sisäänkirjautumisen, ja mikäli kyseessä olisi kaupallinen tai muuten laajalevikkisempi työ, olisi jokaisella ohjelman käyttäjällä ehdottomasti yksilölliset käyttäjätunnukset ja salasanat ja sellaisiin tietoturvateknisiin asioihin kuin esimerkiksi sessiopohjaisuus ja salasanojen kryptaaminen olisi kiinnitetty paljon enemmän huomiota. Tämän opinnäytetyön kannalta sisäänkirjautumisella on, kuten jo työn rajauksessa kävi ilmi, lähinnä demonstraatioarvoa. Ohjelmaan on luotu yksi käyttäjä, jonka käyttäjätunnus on ”user” ja salasana ”password”. Mikäli sisään yrittää kirjautua väärällä käyttäjätunnuksella tai salasanalla, ohjelma ilmoittaa tästä käyttäjälle (”Käyttäjätunnus tai salasana on virheellinen.”)

Sisäänkirjautumisen jälkeen käyttäjälle aukeaa ohjelman etusivu, jonka yläreunassa bannerikuvan päällä näkyvät käyttäjän nimi, kyseisen päivän päivämäärä (ohjelma hakee banneriin automaattisesti kuluvan päivän päivämäärän) sekä uloskirjautumispainike.

Etusivun keskeisin tekninen elementti navigaation lisäksi ovat tulojen ja menojen listaukset (kuva 8), joihin ohjelma hakee tietokannasta sinne syötetyt tapahtumat ja näyttää ne kummassakin taulukossa lisäyspäivämäärän mukaisessa järjestyksessä. Kyseessä on kummankin listan tapauksessa käytännössä Bootstrap-taulukko, johon sisältö haetaan tietokannasta. Ohjelma muodostaa tietokannan tapahtumista listan ja syöttää sen taulukkoon rivi kerrallaan, ja taulukon solut ottavat parametreinä seuraavat muuttujat: tapahtuman nimi, tapahtuman päivämäärä, tapahtuman summa ja tapahtuman kommentti. Lisäksi jokaisen taulukon rivin perässä on rivin poistopainike, joka tarjoaa käyttäjälle mahdollisuuden poistaa kyseinen tapahtuma tietokannasta.



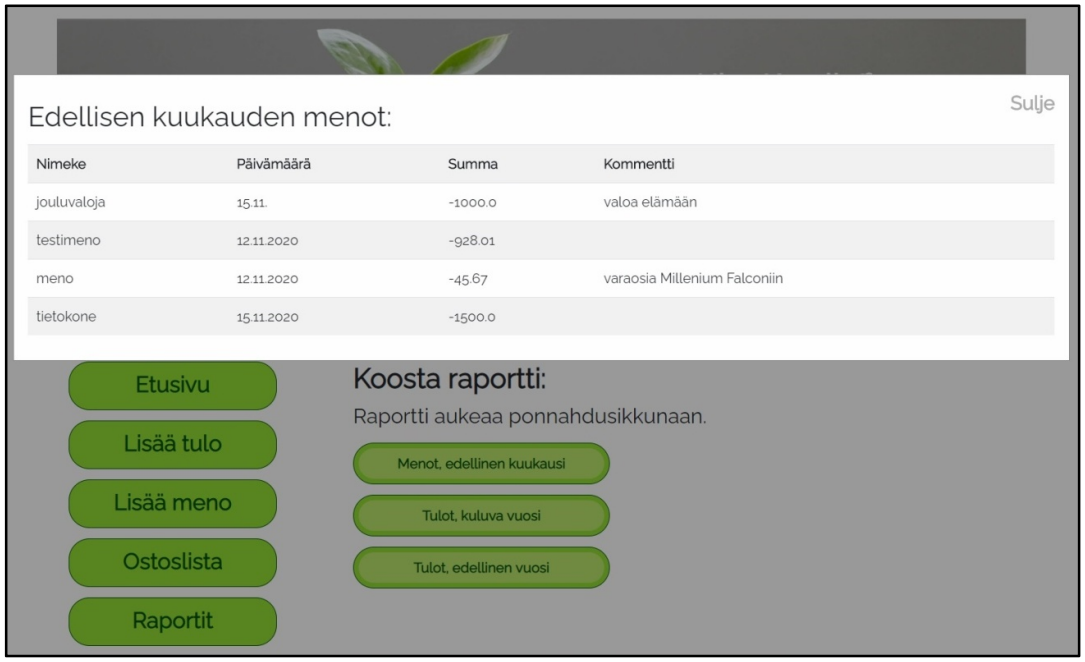
Kuva 8. Etusivun listaukset tuloista ja menoista

Sekä tulon että menon lisäys ovat tekniikaltaan aivan samanlaisia. Ainoa ero tulon ja menon syöttämisessä järjestelmään on se, että menon on oltava negatiivinen luku, eli summan eteen on kirjoitettava miinus-merkki (tarkemmin tiedon käsittelyn ja tallentamisen ehdot on käsitelty tämän raportin luvussa 4).Toiminnallisesti sekä tulo että meno lisätään tietokantaan samalla tavalla: käyttäjä syöttää lomakkeeseen vaaditut tiedot, ja Lisää-painike tallentaa annetut tiedot tietokantaan, tapahtumasta riippuen joko Tulo- tai Meno-tauluun. Mikäli käyttäjä syöttää virheellistä tietoa tai pakollinen tieto jää puuttumaan, ohjelma ilmoittaa tästä käyttäjälle.

Ostoksen lisääminen ostoslistaan, eli ohjelman tietokantaan, toimii täysin samalla tavalla kuin menon tai tulonkin lisääminen tietokantaan, mutta käyttäjältä pyydetään ostoslistaan vain ostoksen nimi. Ostoslista-sivu sisältää samanlaisen taulukkoelementin kuin etusivukin. Ohjelma muodostaa tietokannan Ostos-taulun sisällöstä listan ja syöttää sen

taulukkoon rivi kerrallaan ostoksen nimen ollessa muuttujana. Kunkin rivin perässä on myös rivin poistopainike, jota painettaessa ostos poistuu taulukosta ja sen tiedot tietokannasta.

Ohjelman viimeinen sivu ”Raportit” tarjoaa käyttäjälle mahdollisuuden koostaa muutaman yksinkertaisen raportin ohjelmaan merkittyjen tietojen pohjalta - käyttäjä voi tarkastella joko viimeisen kuukauden menoja tarkastelupäivästä lukien, kuluvan vuoden tuloja tai edellisen vuoden tuloja. Käyttäjän painaessa jotakin raporttisivun kolmesta painikkeesta haluttu raportti aukeaa ponnahdusikkunaan kuvan 9 mukaisesti.



Kuva 9. Raporttinäkymä

6 Pohdinta

Toteuttamani järjestelmä itsessään on loppujen lopuksi nykyaikaiseksi tietokoneohjelmaksi jokseenkin vaatimaton, ja projektin lähtökohdista - ohjelma henkilökohtaiseen budjetointiin - saisi jatkokehittämällä epäilemättä aikaan monitahoisemman ja ns. ”hienommankin” lopputuloksen. Omista henkilökohtaisista lähtökohdistani katsoen olen kuitenkin opinnäytetyön lopputulokseen tyytyväinen. Tekemäni ohjelma palvelee tarkoitustaan opitun näyttämisessä hyvinkin kirjaimellisesti sen koostuessa kaikesta ”koulussa opitusta”, josta minulla ei ollut minkäänlaista aiempaa kokemusta. Olen toteuttanut kursseilla ohjelmista osia - yhdellä kurssilla käyttöliittymää, toisella ohjelmakoodia, kolmannella tietokannan suunnittelua - mutta en ole aiemmin toteuttanut kokonaista ohjelmaa ja sen kaikkia osia alusta loppuun itse, back endistä front endiin. Vaikka ohjelmani on yksinkertainen, osoittaa se mielestäni minun omaksuneen lähtökohdat, joista voin edelleen kehittää ohjelmointiosaamistani.

Käyttöliittymätestien yhteydessä oli mielenkiintoista huomata kaikkien koehenkilöiden olevan sitä mieltä, että tällaiselle järjestelmälle on ehdottomasti tarvetta, ja että luomassani ohjelmassa olisi potentiaalia jatkokehityksen kautta kasvaa aivan oikeaksi tuotteeksi. Kuten jo johdannossa mainitsin, henkilökohtaiseen rahavirtojen seurantaan on tarjolla yllättävän vähän työkaluja, etenkin työpöytätoteutuksina mobiiliohjelmien sijaan. Opinnäytetyöni idea ei missään vaiheessa pohjautunut kaupallisuuteen vaan puhtaasti opinnäytetyön tuottamiseen, mutta ajatuksella voi aina leikitellä - mitä tapahtuisi, jos tätä pientä budjetointijärjestelmää lähtisi kehittämään eteenpäin? Pienillä lisäyksillä, kuten menojen luokittelu ja raprttien tulostusmahdollisuudet, pääsisi jo melko pitkälle. Tämän työn puitteissa nämä kehitysehdotukset eivät toteudu, mutta koen, että pienen järjestelmäni alusta loppuun asti tuottaminen on kasvattanut valmiuksiani niin järjestelmien kuin itsenikin ammatilliseen edelleen kehittämiseen. Jatkokehitys tulevaisuudessa ei siis missään nimessä ole täysin mahdotonta.

Koko ohjelmointi- ja opinnäytetyöprosessi itsessään opetti minulle teknisen osaamisen lisäksi entistä enemmän ennakkoluulotonta tiedon etsintää, opitun soveltamista ja osaamisen kehittämistä törmätessäni ongelmiin ja haasteisiin, jotka oli yksinkertaisesti pakko saada selvitettyä tai kierrettyä ohjelman toimivuuden varmistamiseksi. Jotkin aivan opintojen aluksi oppimani asiat olivat saattaneet opinnäytetyön aloitukseen mennessä jo muuttua (kuten IT-maailmassa yleistä on), ja toisaalta esimerkiksi tietokantayhteyden rakentamisen opettelin käytännössä kokonaan itse lähdemateriaalin pohjalta. Ohjelmointi onkin lajina mielestäni huomattavasti ennalta-arvaamattomampaa kuin ”pelkkä” raportin kirjoittaminen, joten projektin eteneminen ja aikataulutus elivät sen mukaan jatkuvasti.

Aivan täydelliset pattitilanteet ja sen kautta kohtuuttomat viivästykset onnistuin välttämään, ja lopputuote täyttää sille jo johdannossa asetetut vaatimukset.

Valintana toiminnallisen opinnäytetyön tekeminen oli jo oma haastensa, sillä keksittyäni lopulta, millaisen opinnäytetyön voisin tehdä, oli minun tuotettava sekä ohjelma itse että raportti ohjelmasta ja sen toteuttamisesta. Koen ratkaisun kuitenkin lopulta olleen oikea, vaikkakin työteliäs; pääsin tehokkaasti kertaamaan ja edelleen syventämään oppimiani ohjelmointitaitoja, ja toisaalta myös käyttämieni teknologioiden teoriaan oli mielenkiintoista tutustua tarkemmin. Käytännönläheisen opinnäytetyön tekeminen tuntui itselleni luontevalta, ja vaikka yleinen tapa ei olekaan tehdä toiminnallista opinnäytetyötä ilman toimeksiantajaa, antoi tämä toteutustapa minulle niin vapauksia toteuttaa omaa näkemystäni kuin myös mahdollisuuden työskennellä omassa tahdissani. Työskentelytahtini kärsi hieman Covid19-pandemian aiheuttamista ongelmista ja tilanteeseen sopeutuminen kesti osaltani pitkään, mutta olosuhteet huomioon ottaen olen ylpeä suoriutumisestani niin ohjelman kuin raportinkin osalta.

Vaikka itse järjestelmälle esitettiinkin jatkokehitysehdotuksia aina mobiilisovelluksesta alkaen, koin opinnäytetyön edetessä viihtyväni etenkin ohjelman käyttöliittymän suunnittelun ja toteuttamisen parissa. Huomaankin tämän opinnäytetyön valmistuessa nyt toivovani, että pääsisin tulevaisuudessa työskentelemään enemmänkin tämän aihealueen parissa, tapahtuipa se tähän työhön valitsemillani teknologioilla tai muilla, vielä itselleni tuntemattomammilla tavoilla (kuten luvussa 5 mainitsemani React). IT on onneksi ala, jolla kukaan ei todennäköisesti ole koskaan täysin valmis sen jatkuvasti muuttuessa; niinpä omaa osaamistaan ei voikaan muuta kuin alati kehittää!

Lähteet

Cardinal, Mario. 2013. Executable Specifications with Scrum: A Practical Guide to Agile Requirements Discovery. Addison-Wesley Professional. New Jersey. Luettavissa: https://learning.oreilly.com/library/view/Executable-Specifications-with/9780132776530/?ar. Luettu: 20.11.2020.

Chandramouli, Dutt. 2015. Software Project Management. Pearson India. Luettavissa: https://learning.oreilly.com/library/view/software-project-management/9789389552782/?ar. Luettu: 3.12.2020.

Chapple, Mike. 2020. The Fundamentals of SQL. Luettavissa: https://www.lifewire.com/sql-fundamentals-1019780. Luettu: 3.12.2020.

Douglass, Bruce Powell. 2014. Real-Time UML Workshop for Embedded Systems, 2nd Edition. Newnes. Boston. Luettavissa: https://learning.oreilly.com/library/view/real-time-uml-workshop/9780124077812/?ar. Luettu: 26.11.2020.

Gupta, Kanika. 2020. Software Development Frameworks: A Guide for your next Product Idea. Luettavissa: https://www.classicinformatics.com/guide/selecting-development-framework. Luettu: 3.12.2020.

HTML.com. 2020. What is HTML? Luettavissa: https://html.com/#What_is_HTML. Luettu: 3.12.2020.

Knight, Westley. 2018. UX for Developers: How to Integrate User-Centered Design Principles Into Your Day-to-Day Development Work. Apress. New Jersey. Luettavissa: https://www.oreilly.com/library/view/ux-for-developers/9781484242278/?ar. Luettu: 17.11.2020.

MDN Web Docs. 2020. Working with JSON. Luettavissa: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. Luettu: 8.12.2020.

Php.net. 2020. What is PHP? Luettavissa: https://www.php.net/manual/en/intro-whatis.php. Luettu: 3.12.2020.

Pulkkanen, Aleksi. 2019. Agile, Waterfall, Kanban ja muut: 6 yleistä menetelmää projektityöhön - ja miksi sinun kannattaa valita omasi? Agendium. Luettavissa: https://www.agendium.com/projektinhallinta/menetelmat-projektityohon. Luettu: 3.12.2020.

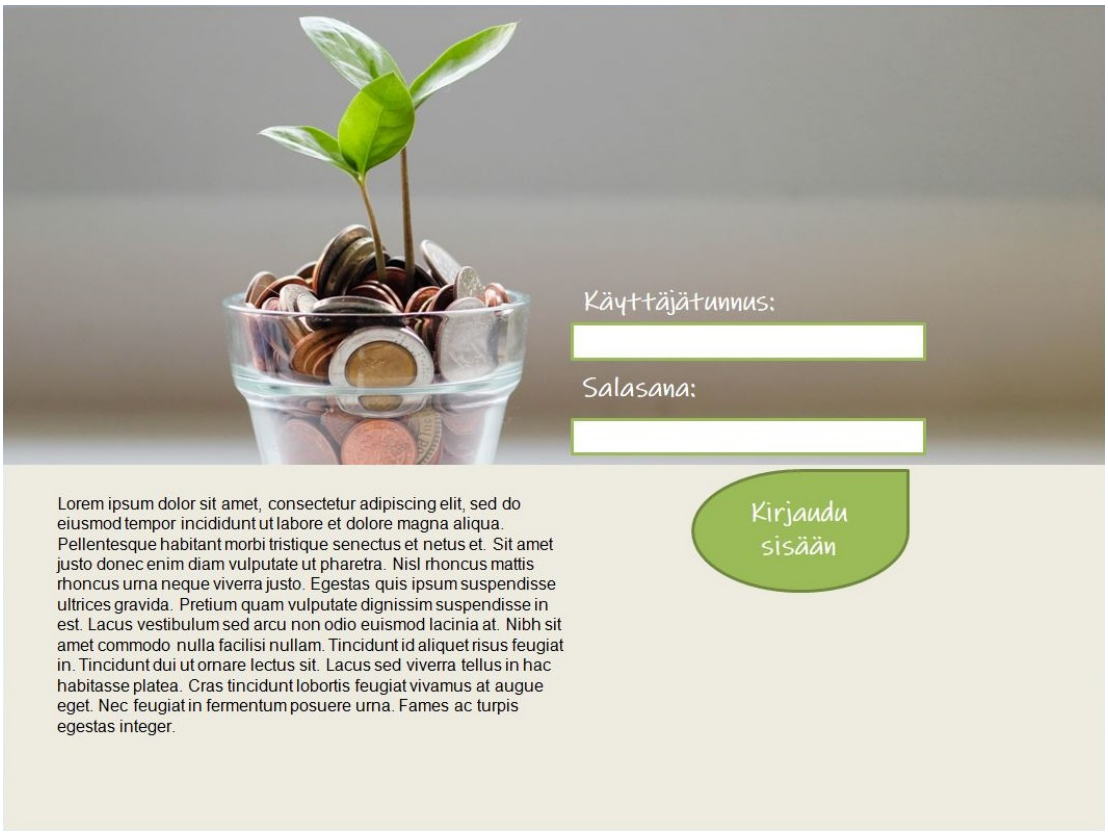
Schwaber, K. & Sutherland, J. 2017. Scrum-opas: Scrumin määritelmä ja pelisäännöt. Luettavissa: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Finnish.pdf. Luettu: 3.12.2020.

Tillman, George. 2017. Usage-Driven Database Design: From Logical Data Modeling through Physical Schema Definition. Apress. New Jersey. Luettavissa: https://learning.oreilly.com/library/view/Usage-Driven-Database/9781484227213/?ar. Luettu: 18.11.2020.

Thornsby, Jessica. 2016. Android UI Design. Packt Publishing. Birmingham. Luettavissa: https://learning.oreilly.com/library/view/Android-UI-Design/9781785887420/?ar. Luettu: 13.11.2020.

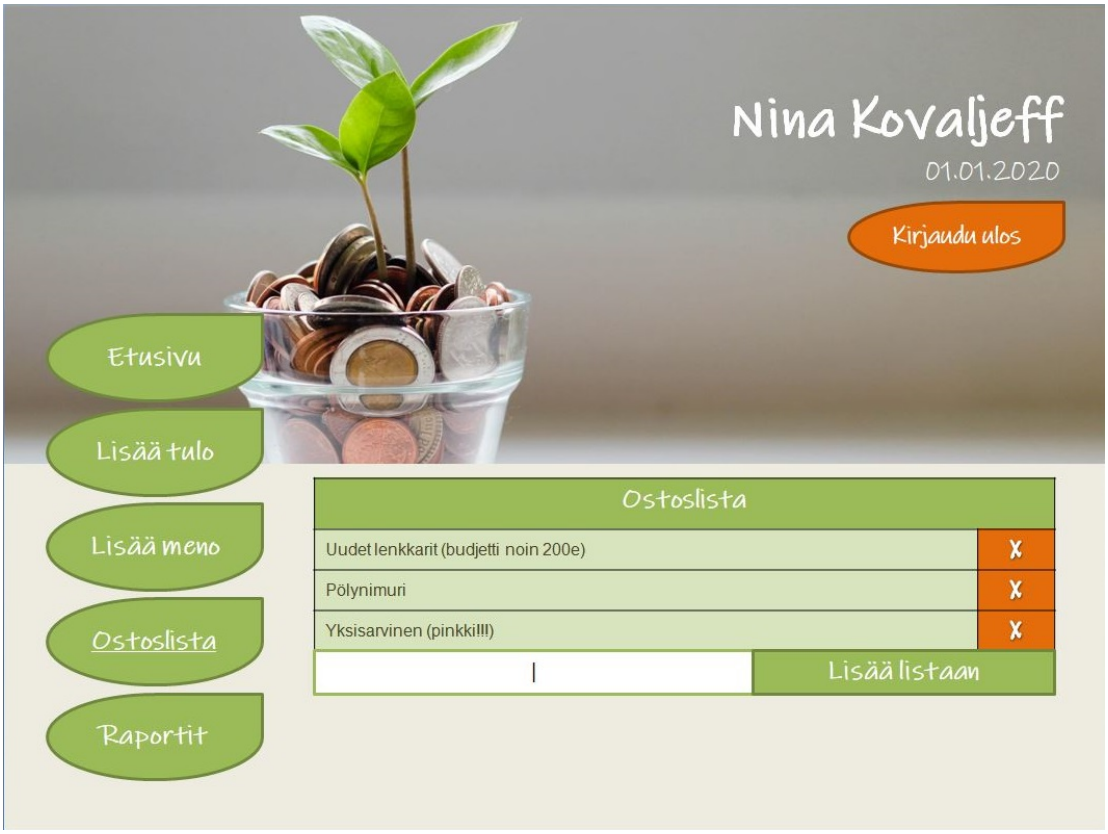
Visual Paradigm. s.a. What is Use Case Diagram? Luettavissa: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/. Luettu: 11.12.2020.

Liite 1. Käyttöliittymän rautalankaprototyypit




Nina Kovaljeff
01.01.2020
Kirjaudu ulos
Etusivu
Lisää tulo
Lisää meno
Ostoslista
Raportit
Nimeke:
Veronpalautukset
Päivämäärä:
24.06.2020
Summa:
507,65
Kommentti:
ou jee!!
Lisää tulo


Nina Kovaljeff
01.01.2020
Kirjaudu ulos
Etusivu
Lisää tulo
Lisää meno
Ostoslista
Raportit
Nimeke:
WRC+
Päivämäärä:
25.06.2020
Summa:
-8,90
Kommentti:
rallia~
Lisää meno


Nina Kovaljeff
01.01.2020
Kirjaudu ulos
Etusivu
Lisää tulo
Lisää meno
Ostoslista
Raportit
Ostoslista
Uudet lenkkarit (budjetti noin 200e)
X
Pölynimuri
X
Yksisarvinen (pinkki!!!)
X
Lisää listaan


Nina Kovaljeff
01.01.2020
Kirjaudu ulos
Etusivu
Lisää tulo
Lisää meno
Ostoslista
Raportit
Koosta raportti:
Menot, edellinen viikko
Menot, edellinen kuukausi
Tulot, edellinen kuukausi
Tulot, viime vuosi
Menot, edellinen viikko
Sähkölasku -98,70
Yksisarvinen -100 000,10
Ruokakauppa -23,75
Ruokakauppa -76,77
Spotify -9,90
Yhteensä 101 209,22 euroa