SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

CCI-KÄYTTÖLIITTYMÄ

Opinnäytetyö

TEKIJÄ: Osku Leskinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Osku Leskinen	
Työn nimi CCI-Käyttöliittymä	
Päiväys 19.5.2016	Sivumäärä/Liitteet 25/0
Ohjaaja(t) Lehtori Jukka Kinnunen, Lehtori Sami Lahti	
Toimeksiantaja/Yhteistyökumppani(t) Enfo Zender Oy	

Tiivistelmä

Opinnäytetyön aiheena oli rakentaa helppokäyttöinen ja selkeä käyttöliittymä asiakkuuden hallintaan käytettävän tietokannan ohjaamiseen. Tietokanta sisältää asiakkaan lähettämille aineistoille määritellyt käsittelyt eli workflow't, sekä yleisiä ja asiakaskohtaisia parametreja, jotka liittyvät asiakkaalta vastaanotettavien aineistojen käsittelyyn sekä laskutukseen. Aineisto käsitellään määritettyjen prosessien mukaisesti ja välitetään edelleen loppuasiakkaalle esimerkiksi sähköisessä muodossa tai paperitulosteena. Parametrien hallinta vanhalla käyttöliittymällä on virhealtista ja monimutkaista, joten uuden käyttöliittymän tekeminen oli välttämätöntä, jotta uusi palvelualusta saadaan käyttöön. Sovelluksella voidaan lisäksi luoda uusia aineistokäsittelyitä tai muokata olemassa olevia. Sovelluksen käyttäjät ovat yrityksessä toimivia asiantuntijoita tai asiakaspalvelutyöntekijöitä, joten ohjelman on huolehdittava siitä, ettei tuotannossa olevia käsittelyitä voida muuttaa siten, että ne lakkaavat toimimasta.

Käyttöliittymä kehitettiin web-sovelluksena HTML, Javascript ja AngularJS -tekniikoilla. Käyttöliittymän kehittämiseen käytettiin Notepad++-tekstieditoria. Työssä käytettiin ketterää-, testivetoista- sekä taustajärjestelmätöntä kehitystä.

Lopputuloksena toteutettiin käyttöliittymä, jonka avulla eri rooleissa olevat käyttäjät voivat tehdä muutoksia yleisiin laskunvälityksen parametreihin sekä perustaa asiakkaalle uuden käsittelyn. Käyttöliittymään on rakennettu useita erilaisia varmistuksia, jolloin mahdollisuus siihen, että käyttäjä tallentaa virheellistä tietoa, on lähes olematon.

Avainsanat
HTML, Angular, Javascript, käyttöliittymä

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Osku Leskinen			
Title of Thesis CCI-Interface			
Date	19 May 2016	Pages/Appendices	25/0
Supervisor(s) Mr Jukka Kinnunen, Lecturer, Mr Sami Lahti, Lecturer			
Client Organisation /Partners Enfo Zender Oy			

Abstract

The purpose of this Thesis was to develop a user friendly, clear and easy to use interface for controlling customer configuration database. The database includes the processeses that handle the data received from the customer and common or customer parameters that control the prosessing and billing of the process. The data is processed and forwarded to the customer's customer according to the defined prosess as an electric invoice or paper invoice. Controlling the parameters with the old interface is complicated and risky, so it was necessary to build the new version to implement the new platform for the company. Creating new workflow's or editing existing workflow's is also possible in the application. The application is used by specialists or customerservice personnel in the company so it has to make sure that the user cannot edit workflows or parameters in a way that it would mess up the process in production environment.

The interface was developed as a web-application with HTML, Javascript and AngularJS tehcniques. The development was done in notepad++. Agile development, test-driven development and backend-less development methods were used in the process.

As a result of this thesis an interface was developed to help persons in different roles to edit the common parameters and create new workflow's for customers. There are many validations built in the interface to prevent user from saving invalid information in to the database.

Keywords
HTML, Angular, Javascript, Interface

ESIPUHE

Haluan kiittää toimeksiantajaa Enfo Zender Oy:tä mahdollisuudesta tehdä opinnäytetyö yritykselle. Haluan kiittää myös yrityksessä toimivia asiantuntijoita, Mika Saarta, Samuli Räikköstä ja Juha Savolaista, jotka toimivat työni ohjaajina yrityksessä.

Haluan myös kiittää lehtori Jukka Kinnusta työn ohjaamisesta.

Kuopiossa 19.5.2016
Osku Leskinen

SISÄLTÖ

TERMIT JA LYHENTEET

CCI	CustomerCareInterface, asiakkuuden hallintaan käytettävä ympäristö.
Workflow	Aineiston käsittelyketju. Määrittää, mitä ohjelmia aineiston käsittelyn yhteydessä suoritetaan, millaisia tulosaineistoja käsittelystä voi jakaantua, sekä miten näistä aineistoista laskutetaan ja raportoidaan asiakasta.
Parametri	Parametri on ohjelmassa olevalle funktiolle tai käskylle välitettävä tieto, jolla voidaan ohjata ohjelman toimintaa. Laskunvälityksessä parametrejä voivat olla esimerkiksi aineistosta lähettetävien raporttien raporttiosoitteet tai postitse lähettävien dokumenttien postiluokka, sekä lomakkeet, joille dokumentti tulostetaan. Parametreja voi olla yleisiä eli kaikilla käsittelyissä vaikuttavia, sekä asiakaskohtaisia, käsittelykohtaisia, että aineistokohtaisia parametrejä.
Aineisto	Asiakkaalta vastaanotettu data, joka voi sisältää laskuja, tai muita dokumenttejä, kuten asiakastiedotteen.

1 JOHDANTO

Opinnäytetyön aiheena on tehdä toimeksiantajalle ohjelma laskunvälitykseen liittyvien parametrien ja aineistokäsittelyiden hallitsemiseen. Työn toimeksiantaja on Enfo Zender Oy ja ohjelman tarkoitus on helpottaa sekä yleisten että asiakaskohtaisten parametrien hallintaa sekä uusien käsittelyiden luomista ja olemassa olevien käsittelyiden muutosta uudessa palvelualustassa.

Enfo Zender on Enfo Oyj:n itsenäinen tytäyhtiö, joka tarjoaa asiakkailleen kokonaisvaltaisia talousprosessipalveluita. Pääasiallinen toimi on laskuoperointi. Enfo Zenderin toimitusjohtajana toimii Tero Kosunen. Enfo Zender syntyi 1. lokakuuta 2012, kun Enfo Oyj:n tiedonvälityspalvelut yhtiöitettiin itsenäiseksi tytäryhtiöksi. Vuoden 2015 lopussa Enfo Oyj työllisti Suomessa ja Ruotsissa yhteensä 805 henkilöä, josta Enfo Zenderin osuus on noin 130 henkilöä. (Enfo Oyj, 2016)

Työn tilaaja on Enfo Zender Oy:n sisällä toimiva InformationLogistics-osasto. InformationLogistics on tiedonvälityspalveluihin keskittynyt osasto, jonka liiketoimintaa on asiakkaalta tulevan tiedon välittäminen loppuasiakkaalle. Tällaista tietoa ovat esimerkiksi laskut, asiakastiedotteet ja muut asiakirjat sekä sanomat.

Kun asiakkaan lähettämä aineisto saapuu palveluntarjoajalle (Enfo Zender Oy), sitä muokataan ja rikastetaan ennen lopullista välittämistä loppuasiakkaalle. Aineisto käy läpi prosessin, jota kutsutaan aineistokäsittelyksi eli workflow'ksi. Näitä toimenpiteitä ja niihin liittyvää laskutusta ohjataan yleisillä ja asiakaskohtaisilla parametreilla.

Tiedonvälityspalvelut ovat siirtymässä uuteen palvelualustaan. Uudessa alustassa asiakkaalta tulevien aineistojen käsittelyketjujen eli workflow'den tiedot ja näitä käsittelyjä ohjaavien parametrien hallinta tapahtuu tietokannan avulla. Kaikki käsittelyt sekä niitä ohjaavat parametrit on tallennettu tietokantaan ja niitä hallitaan erillisen käyttöliittymän avulla. Tästä hallintaympäristöstä käytetään nimeä CCI, CustomerConfigurationInterface.

Nykyinen käyttöliittymä on epäselvä ja hankalakäyttöinen eikä se enää sovellu käytettäväksi, kun asiakkaiden käsittelyitä aletaan enenevissä määrin siirtää vanhasta ympäristöstä uuteen palvelualustaan. Työn tavoitteena on tehdä uusi helppokäyttöisempi ja tehokkaampi työväline korvaamaan vanha käyttöliittymä. Uuden käyttöliittymän käyttäjät ovat joko asiantuntijoita tai asiakaspalvelun henkilöitä, joten käyttöliittymän on oltava helppokäyttöinen. Monentasoisten käyttäjien takia käyttöliittymässä on tarkastettava käyttäjän syöttämiä tietoja, jotta tuotannon käsittelyt eivät mene sekaisin. Työn tavoitteeksi sovittiin uuden käsittelyn eli workflow'n lisäämiseen ja yleisten parametrien hallintaan käytettävän käyttöliittymän rakentaminen.

2 KÄYTETYT TEKNIIKAT JA KEHITYSMENETELMÄT

2.1 Tekniikoiden valitseminen

Ohjelman tekemiseen käytettävät tekniikat määräytyivät sen mukaan, mitä uuden palvelualustan muihin komponentteihin oli jo käytetty. Aiemmat komponentit oli toteutettu web-pohjaiseksi HTML- ja Javascript-tekniikoilla, joten yhteensopivuuden vuoksi tämä ohjelma toteutettiin samoilla menetelmillä. Seuraavassa esitellään ohjelman tekemiseen käytettyjä tekniikoita sekä kehitysmenetelmiä.

2.1.1 HTML

Hypertext Markup Language eli hypertekstin merkintäkieli on avoimesti standardoitu kuvauskieli, jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä eli hypertekstiä. HTML tunnetaan erityisesti kielenä, jolla internetsivut on kirjoitettu. HTML-koodi on rakenteista tekstiä, joka koostuu sisäkkäisistä ja peräkkäisistä elementeistä, jotka merkitään kulmasulkein merkityillä tunnisteilla eli tägeillä (tag). (Wikipedia, HTML, ei pvm). HTML oli tämän sovelluksen näkyvien osien eli käyttöliittymän ohjelmointikieli.

2.1.2 Javascript

Javascript on Netscape Communications Corporationin kehittämä pääasiassa web-ympäristössä käytettävä dynaaminen komentosarjakieli. Javascriptin tärkein sovellus on mahdollisuus lisätä web-sivuille dynaamista toiminnallisuutta. (Wikipedia, Javascript, ei pvm). Tässä sovelluksessa Javascript oli pääasiallinen ohjelmointikieli ja sitä käytettiin käyttöliittymän komponenttien toiminnallisuuksien toteuttamiseen.

2.1.3 JSON

Javascript Object Notation on yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen. Sovelluksessa JSON-objekteja käytetään tiedon välittämiseen tietokantayhteyksissä eli tiedon hakemiseen ja tallentamiseen. (Wikipedia, JSON, ei pvm). Kaikki tiedonvälitys ohjelmassa tapahtuu JSON-objekteina. Tietokannasta haettavat tiedot sekä tietokantaan tallennettavat tiedot liikkuvat tietokannan, kontrollerin ja käyttöliittymän välissä ennalta sovitun kuvauksen mukaisina objekteina.

2.1.4 MULE

MULE on kevyt, Java-pohjainen integraatioviitekehys ja ESB (Enterprise service bus –arkkitehtuuri), jonka avulla kehittäjä voi nopeasti ja helposti yhdistää sovelluksia siten, että niiden välillä siirtyy tietoa. MULE mahdollistaa helpon intergaation olemassa oleville järjestelmille huolimatta siitä, että sovellusten tekniikat ovat erilaisia. MULEn tärkein etu on se, että se sallii ohjelmien kommunikoinnin keskenään toimimalla välityskanavana, joka siirtää dataa ohjelmasta toiseen yrityksen sisällä tai internetin yli. (What is Mule ESB, ei pvm)

Sovelluksen kehityksessä MULE-palvelua käytettiin palveluväylänä käyttöliittymän palvelun ja Java-pohjaisten tietokantakyselyiden välissä. Tietokantakyselyt tehtiin omaksi Java-komponentikseen ja MULElla ohjattiin yhteys käyttöliittymästä tähän komponenttiin. Mikäli myöhemmässä vaiheessa tarvitsee käyttää samoja kyselyitä uudelleen, voidaan mulella tehdä uusi yhteys uuden komponentin ja kyselyiden välille eikä kyselyitä siis tarvitse tehdä useaan paikkaan.

2.1.5 RESTapi ja RESTful

REST eli Representational State Transfer, on HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen. REST-arkkitehtuurin on tarkoitus parantaa suorituskykyä, skaalautuvuutta, yksinkertaisuutta, muunneltavuutta, luotettavuutta ja siirrettävyyttä. REST-mallin ajatus on, että ohjelma käyttää tiedon hakemiseen tarkasti määritettyjä kyselyitä. Nämä kyselyt sisältävät resurssin, josta tietoa haetaan sekä metodin jota halutaan käyttää. Näitä metodeja ovat GET (tiedon lukeminen), POST (uuden tiedon lisääminen), PUT (olemassa olevan tiedon päivittäminen) sekä DELETE (tiedon poistaminen). (Wikipedia, Representational state transfer, ei pvm). Tämän jälkeen määritetty resurssi palauttaa vastauksena HTTP-vastauskoodin. Vastauskoodit ovat kolminumeroisia koodeja, joita on viittä eri tyyppiä. 100-sarjan vastaukset ovat informatiivisiä, 200-sarjan koodit ovat onnistumisia, 300-sarja on uudelleenohjausta, 400-sarjan koodit tarkoittavat käyttäjän tekemää virhettä ja 500-sarja palvelinpään virhettä. (REST API Tutorial, ei pvm) . Ohjelmassa käytetään eniten kooodeja 200 (OK), 201 (tiedon luominen onnistui) tai 404 (resurssia ei löydy). Vastauksessa on lisäksi otsakkeet sekä vastauksen runko. Vastauksen runkona tässä sovelluksessa oli aina halutun tiedon sisältävä JSON-objekti. Vastauskoodit ovat kuitenkin jopa vastauksen runkoa tärkeämpiä, sillä niiden avulla pystytään ohjelmassa päättelemään, miten ohjelma menee eteenpäin. Ohjelman saamaa vastausta hyödynnetään esimerkiksi uuden workflow'n lisäämisessä. Mikäli ohjelma saa vastauksen 201 eli tiedon luominen on onnistunut, voidaan käyttäjälle näyttää ilmoitus onnistumisesta. Mikäli taas vastaus on jokin 400-sarjan vastaus, voidaan käyttäjälle näyttää asianmukainen virheilmoitus.

2.1.6 GruntJS

Grunt on Javascriptin tehtävänsuorittaja, joka automatisoi kehityksessä tarvittavien, useasti toistuvien tehtävien suorittamisen. Tällaisia tehtäviä ovat esimerkiksi lähdekoodin kääntäminen toimivaksi ohjelmaksi, yksikkötestaus, tyhjien tai turhien merkkien poistaminen eritoten Javascript-koodista sekä koodissa olevien, turhien virheiden havaitseminen. Tällaisia turhia virheitä ovat esimerkisi syntaksivirheet, kuten puuttuvat kaarisulkeet tai puolipisteen puuttuminen. Gruntin ominaisuuksiin kuuluu myös livereload-toiminto, joka päivittää sivuston heti, kun koodiin tehdään muutos. Jos ohjelmaa muokataan samalla, kun sivusto on auki, toiminto päivittää muutokset sivustoon heti, kun se on havainnut muutoksen sellaisissa tiedostoissa, joihin selaimella on pääsy. Grunt myös suorittaa sovelluksen spec tiedostoihin kirjoitetut testit ja varmistaa, että kaikki toimii oikein.
(GRUNT The JavaScript Task Runner, ei pvm)

2.1.7 Yeoman

Yeoman on pakettien hallintaan käytettävä komentoriviltä suoritettava järjestelmä. Yeomania käytetään myös projektitiedostojen generoimiseen. Yeoman tukee useita ohjelmointikieliä, kuten Javaa, Pythnonia tai HTML:ää. (The web's scaffolding tool for modern webapps, ei pvm). Sovellusta kehitettäessä Yeomania käytettiin ohjelmassa vaadittavien viitekehysten, kuten Angularin asentamiseen ja päivittämiseen sekä uusien projektien luomiseen.

2.1.8 AngularJS

AngularJS on Googlen ylläpitämä avoimen lähdekoodin Javascript-ohjelmointikehys. AngularJS:n tavoite on lisätä selainpohjaisiin sovelluksiin tuki MVC-arkkitehtuurille, jonka avulla sivustojen kehitys ja testaaminen helpottuvat. Kirjasto laajentaa HTML:n elementtejä uusilla attribuuteilla, jotka lisäävät olemassa oleviin elementteihin haluttuja toiminnallisuuksia mallipohjan mukaisesti muuttujiin liitettyinä. Muuttujia voidaan kontrolloida joko staattisesti, Javascriptillä tai dynaamisesti esimerkiksi JSON-objekteina. (Wikipedia, AngularJS, ei pvm) (Introduction to AngularJS, ei pvm). Kaikki sovelluksessa olevat näkymät on tehty AngularJS:n komponenteilla. Näihin komponentteihin on sidottu tietokannasta haettua dataa Javascriptin tai JSON-objektien avulla.

2.1.9 Notepad++

Notepad++ on avoimen lähdekoodin tekstieditori, jossa on tuki monille ohjelmointikielille, kuten Javascriptille ja HTML:lle. Kaikki ohjelman lähdekoodit kirjoitettiin Notepad++:lla.
(Notepad++ Home, ei pvm)

2.1.10 Postman

Postman on Googlen Chrome -selaimelle ladattava lisäosa, jolla voi kehittää, testata ja dokumentoida ohjelmointirajapintoja (API). Postmanin tärkeimpiin ominaisuuksiin kuuluu tehtyjen pyyntöjen historiatietojen tallennus sekä uusien pyyntöjen luonti. (Postman - Supercharge your APIs, ei pvm). Tässä työssä Postmania käytettiin MULEn kautta lähetettävien tietokantaan menevien pyyntöjen testaamiseen sekä luomiseen. Postmanin avulla pyyntöön tulevan vastauksen sekä vastauksena tulevan objektin tietorakenteen pystyi nopeasti ja helposti tarkistamaan.

2.1.11 GIT

GIT on versionhallintaan käytettävä ohjelmisto, joka on suunnitelty vuonna 2005 Linuxin kehittämistä varten. Gitin toimintaperiaate on se, että käyttäjä lataa omalle tietokoneelleen GITin käyttöliittymän. Tämän käyttöliittymän kautta määritetään palvelimella olevan projektikansion eli repositoryn osoite ja haetaan projektista työkopio omalle työasemalle. Kun sovellusta kehitetään, tehdään muutokset tähän työkopioon. Muutosten tekemisten jälkeen käyttäjä lähettää muutokset käyttöliittymän kautta palvelimella sijaitsevaan projektiin, jonka jälkeen ne ovat myös muiden kehittäjien käytettä-

vissä. Käyttäjä voi myös päivittää omaan työkopioonsa muiden tekemiä muutoksia hakemalla uusimmat muutokset omaan kopioonsa. Jokainen commit eli varsinaisen projektin päivitys, versioidaan ja muutokset dokumentoidaan. Näin voidaan aina palata takaisin johonkin tiettyyn versioon, mikäli oma työkopio tai varsinainen projektikansio menevät käyttökelvottomaksi.
(Wikipedia, Git (software), ei pvm)

2.2 Kehitysmenetelmät

2.2.1 Test driven development

Testivetoinen kehitys on ohjelmointia tukeva tekniikka, jossa tehdään ohjelman toiminnallisuuksia etukäteen kirjoitettujen testitapausten avulla. Menetelmä toimii siten, että ensin kirjoitetaan kuvauksia sellaisista toiminnoista, joista valmiin ohjelman on pystyttävä suoritumaan. Tämän jälkeen kehitetään ohjelma siten, että se läpäisee nämä testit. Kehittäminen ja virheiden korjaaminen on turvallisempaa, koska olemassa olevia testejä suorittamalla huomataan myös uudet virheet. Tämän työn kehityksessä tehtiin käyttöliittymälle sekä yksikkötestit, että end-to-end eli e2e-testit, joiden avulla varmistuttiin siitä, että komponentit ja niiden toiminnot toimivat oikein ja että ohjelman käsittelemät tietorakenteet ovat oikean muotoisia. End-to-end testit toimivat siten, että testitiedostoon kirjoitetaan toiminnot siten kuin käyttäjä niitä käyttäisi. Tällaisia toimintoja ovat esimerkiksi erilaisten syötteiden täyttäminen tai valintalaatikoiden valinta. Tämän työn testeissä testattiin enimmäkseen ohjelmaan rakennettuja tarkastuksia, sekä käyttöliittymän komponenttien toimintaa. Yksikkötestiin kuvattiin suoritettava toiminta, sekä haluttu lopputulos, kun kenttä on täytetty tai painiketta painettu. Kuvassa 1 on esitelty workflow'n lisäämisessä käytettävä end-to-end testi, joka tutkii virheilmoituksen esittämistä ja jatkamiseen käytettävän painikkeen lukittumista, jos täytetyn workflow'n tiedot on annettu väärin.

```
describe('When I am filling out the workflow basic information', function() {

    beforeEach(function() {
        browser.get('http://localhost:9002/#/workflow-manager/');
    });

    it('An error should be displayed if the enterd workflow is invalid', function() {

        var workflowInput = element(by.id("workflowInput"));
        workflowInput.sendKeys('ABC', protractor.Key.ENTER);
        element(by.css("#content > md-content")).click();

        var inputContainer = element.all(by.css("#tab-content-3 > div > md-content > form > md-input-container")).first();
        expect(inputContainer.getAttribute('class')).toContain('md-input-invalid');

    });

    it('Valid value for workflow should be accepted without validation error', function() {
        var workflowInput = element(by.id("workflowInput"));
        workflowInput.sendKeys('ABC123', protractor.Key.ENTER);
        element(by.css("#content > md-content")).click();

        var inputContainer = element.all(by.css("#tab-content-3 > div > md-content > form > md-input-container")).first();
        expect(inputContainer.getAttribute('class')).not.toContain('md-input-invalid');
    });

    it('Add workflow button should be disabled if form is invalid', function() {
        var button = element(by.id("addWorkflowButton"));
        expect(button.getAttribute("disabled")).toBe("true");

    });
```

KUVA 1 Yksi workflow'n lisäämisen e2e testeistä

(Wikipedia, Testivetoinen kehitys, ei pvm)

2.2.2 Ketterä kehitys ja Scrum

Scrum on projektinhallinnan viitekehys, jota käytetään yleisesti ketterässä ohjelmistokehityksessä. Tarkoitus on, että projekti jaetaan kehitysjaksoiksi eli sprinteiksi. Sprintit ovat 1-4 viikon mittaisia aikarajoja, joiden aikana tuotetaan valmiin määritelmän täyttävä, käyttökelpoinen tuoteversio. Jokaisen sprintin sisältö sovitaan suunnittelupalaverissa ennen sprintin aloitusta. Toteutettavaksi valitaan sellaisia kehitysjonon kohtia, joilla on sillä hetkellä suurin merkitys projektin onnistumiselle. (Wikipedia, Scrum, ei pvm)

Tässä työssä sovellettiin scrumia siten, että kehitykseen käytettävät käyttäjätarinat eli kuvaukset ohjelman toiminnoista, kirjoitettiin työvaiheiden ohjaukseen tarkoitettuun Jira-järjestelmään, josta ne tehtiin numerojärjestyksessä. Projektin omistaja oli tehnyt ja järjestänyt nämä käyttäjätarinat siten, että kehitys eteni toivotussa järjestyksessä.

Käyttäjätarinat laadittiin siten, että niiden perusteella pystyi suunnittelemaan sekä ohjelman käyttöliittymää, että logiikkaa. Käyttäjätarinat kuvasivat pieninä palasina ohjelman haluttua toimintaa ja niissä oli selkeästi kerrottu, miten ohjelman tulee kussakin tilanteessa toimia. Käyttäjätapauksissa oli myös kuvattu rooleja, joiden oikeuksilla kyseisiä muutoksia voidaan ohjelmassa toteuttaa.

Sprinttien edistymistä käytiin läpi muutaman viikon välein pidettävissä palavereissa, joissa myös sovittiin seuraavista kehitysvaiheista.

2.2.3 Backend-less development

Backend-less development on kehitysmenetelmä, jossa käyttöliittymän eli front-endin kehitystä tehdään toiminnalliseksi ennen kuin palvelinpuoli eli back-end on toiminnassa. Tällaisella kehitystavalla vältetään tilanne, jossa käyttöliittymän kehitys pysähtyy, koska tietoa käsitteleviä back-end ohjelmia ei vielä ole olemassa. Backend-less development toimii siten, että käyttöliittymään toteutettavat HTTP-pyynnöt luodaan käyttäen palvelun tietokantapyynnöissä parametrina $httpBackendiä, joka esittää varsinaista HTTP-pyyntöä. Tämän parametrin avulla ohjataan kysely tiedostoon, johon on kirjoitettu ennalta määritettyjä vastauksia kyselyille. Näitä vastauksia kutsutaan nimellä mock. Mockit ovat simuloituja objekteja, jotka kuvaavat HTTP-pyynnöillä vastaukseksi tulevia, oikeita objekteja. Kun varsinainen back-end on saatu valmiiksi, poistetaan tietoa hakevista funktioista tuo mockille ohjaava $httpBackend jolloin nämä kutsut toimivat ja palauttavat varsinaisesta tietokannasta haettavaa dataa. Tämä helpottaa ja nopeuttaa kehitystä, koska testaamiseen ja tuotannolliseen käyttöön ei tarvita eri koodeja. (Amin, ei pvm)

Koska sovellusta kehitettäessä aloitettiin ensin tekemään käyttöliittymästä halutun näköistä, ja palvelinpään toteutusta vasta myöhemmin, oli backend-less development erityisen tärkeää. Kun komponenttien testidatana käytettiin jo heti alusta lähtien oikeaa dataa, saatiin komponentit heti toimiviksi. Pahimmassa tapauksessa komponenttien ja sitä kautta koko sovelluksen toiminnallisuus olisi jouduttu rakentamaan uudelleen, jos varsinainen data olisi ollut täysin erilaista testidataan nähden.

Myös palvelinpään tekeminen helpottui, kun käyttöliittymään oli jo rakennettu kutsuja hoitavat funktiot ja oli olemassa rajapintakuvaukset ohjelmaan halutusta datasta.

Tästä esimerkkinä on asiakashaku, joka tehtiin siten, että ensin luotiin käyttöliittymässä valikko, johon haluttiin ladata asiakkaat. Kentässä näytetään asiakkaan nimi ja arvona on asiakkaan id. Tämän valikon toiminnallisuuteen luotiin funktio, joka sivun latautuessa hakee ja täyttää kenttään asiakkaiden tiedot.

Tälle funktiolle luotiin mock-palvelu, joka palautti sovellukselle oikean muotoista esimerkkidataa (kuva 2).

```
$httpBackend.whenGET(/\/cciadmin\/rs\/v1\/customers/).respond(function(method, url, data)
{
    var customers =
    {
        "customers":
        [
            "name":"Asiakas 1","id":"4"},
            {"name":"Asiakas 2","id":"2"}
        ]
    };
       return [200, customers, {}];
});
```

KUVA 2 Asiakashaun mock-palvelu

Näin saatiin komponenttiin ladattua lista asiakkaista ja testattua sovellusta. Kun palvelinpään toteutus saatiin valmiiksi, muutettiin vain ohjelma palvelinkutsuja ohjaavassa tiedostossa olevan funktion rakenne tästä

```
$httpBackend.whenGet(/.html$/).passThrough();
```

tähän

```
$httpBackend.whenGet(/^https?:\/\/[^l]/).passThrough();
```

Jolloin asiakaslista haettiinkin mockin sijaan määritetystä tietokannasta.

3 SOVELLUS

3.1 Tarkoitus

Sovelluksen tarkoitus on helpottaa suunnittelijoiden ja asiakaspalvelijoiden työtä tekemällä tuotantoa ohjaavan parametrien ja käsittelyketjujen lisäämisestä sekä muutoksesta helpompaa ja vähemmän virhealtista. Käyttöliittymä on suunniteltu siten, että esimerkiksi uuden aineistokäsittelyn eli workflow'n lisääminen tehdään vaihe vaiheelta ja seuraavaan vaiheeseen ei pääse jatkamaan, ennen kuin kaikki läpikäytävässä vaiheessa vaaditut tiedot on annettu ja tarkastettu.

3.2 Suunnittelu

Sovelluksen suunnittelu aloitettiin vanhaan versioon tutustumisella. Vanhasta versiosta käytiin läpi toiminnot sekä järjestelmän puutteet ja parannusehdotukset. Tämän jälkeen aloitettiin uuden käyttöliittymän suunnittelu näiden tietojen pohjalta. Tässä vaiheessa selvitettiin myös ohjelman vaatimukset ja työvaiheet. Ensimmäisenä sovittiin tehtäväksi uuden workflow'n lisääminen ja siihen vaadittavat toiminnallisuudet. Workflow'n lisäämiseen suunnitellut komponentit sovittiin toteutettavaksi siten, että samoja komponentteja voidaan käyttää myös olemassa olevan workflow'n tietojen muuttamiseen. Kun työvaiheet oli saatu sovittua, piirrettiin paperille suunnitelma ensimmäisestä käyttöliittymän versiosta. Tähän vedokseen pyydettiin kommentit käyttäjiltä, ja kun korjausehdotukset oli kuultu, alettiin työstää ensimmäistä toiminnallista versiota suunnitelman mukaan. Ensin tehtiin käyttöliittymä ja sen jälkeen toteutettiin toiminnallisuutta mockien avulla. Tämän jälkeen ulkoasua ja prosessien läpikäymiseen liittyviä järjestyksiä hiottiin kommenttien avulla paremmaksi. Käyttöliittymän suunnittelun ja mockien toteutuksen rinnalla tehtiin myös hahmotelmat siitä, millaisia tietokantakyselyitä eri näytöille, komponenteille ja prosessin vaiheille tarvitaan toteuttaa.

3.3 Rakenne

Sovellus on modulaarinen, mikä tarkoittaa sitä, että se on rakennettu useista erilaisista toiminnoista, joita käytetään yhtenä sovelluksena pääsivun kautta. Kaikki viittaukset ohjelmien käyttämiin kirjastoihin ja viitekehyksiin sisällytetään ohjelman juuressa sijaitsevaan index.html-sivuun, jonka kautta sovellusta käytetään. Tämä tiedosto sisältää myös ulkoasun määritykset ja komponentit, joten niistä ei tarvitse huolehtia muualla ohjelmassa. Näiden viittausten lisäksi index.html sisältää ainoastaan linkit toimannallisiin moduuleihin, jotka sijaitsevat ohjelman alihakemistossa. Uudet toiminnallisuudet luotiin käyttäen Angularin omaa generatoria, joka osasi lisätä viittaukset pääsivulle automaattisesti. Varsinaista ohjelmointia tehtiin siis partial- ja service-alihakemistoissa sijaitseviin moduuleihin. Sovelluksen taustalla on jo olemassa oleva MySQL-tietokanta, jota käytetään tiedon hakemiseen ja säilyttämiseen.

3.3.1 Partial

Partial-hakemistossa sijaitsevat kaikki sovelluksessa olevien moduulien käyttöliittymien tiedostot ja lähdekoodit. Partial on siis erillinen käyttöliittymän osio, joka tässä tapauksessa on CCI-käyttöliittymä. Uuden moduulin luominen aloitettiin luomalla uusi käyttöliitymä eli partial angularin generatorilla. Käskylle annettiin parametrina haluttu moduulin nimi. Uuden partialin luominen onnistui komentokehotteen käskyllä

```
yo cg-angular:partial cci-interface
```

Käsky loi uuden cci-interface-nimisen hakemiston partial-hakemiston alle ja lisäsi sinne HTML, Javascript ja spec -tiedostot. HTML-tiedostoon lisättiin kaikki käyttöliittymän ulkoasuun liittyvät muotoilut ja komponentit. HTML-tiedostossa on määritetty viittaus kyseisen ohjelman controlleriin eli partialin alla sijaitsevaan Javascript-tiedostoon. Controller-tiedosto sisältää kaikki komponenteissa vaadittavat ja käytettävät toiminnallisuudet eli käyttöliittymän logiikan. Controller ja käyttöliittymä ovat sidotut toisiinsa ja niiden välissä on kaksisuuntainen yhteys. Tämä tarkoittaa sitä, että kun käyttäjä esimerkiksi valitsee pudotusvalikosta asiakkaan, jolle uusi workflow lisätään, tämä tieto vaihtuu myös controllerin puolella olevaan olioon, johon tallennetaan uuden workflow'n tiedot. Näin objekti on ajantasainen koko prosessin ajan eikä käyttäjän tarvitse erikseen vahvistaa tai tallentaa valintojaan. Tämä yhteys estää myös sellaiset tapaukset, joissa käyttäjä workflow'n lisäämisen loppuvaiheessa palaa takaisin alkuun muuttamaan valittua asiakasta ja tietokantaan tallennettaisiin uusi workflow väärälle asiakkaalle.

Koska jokaiselle sovelluksen osalle tehtiin omat partialinsa, ei yhden moduulin sisältö ollut välttämättä kovinkaan monimutkainen. Kuvassa 3 on esitelty yleisten parametrien muuttamiseen tarkoitetun moduulin ensimmäisen version HTML:n sisältö. Tässä moduulissa oli vain controllerissa sijaitsevan olion mukaan luotu taulukko, jossa käyttäjälle tulostettiin yleisten parametrien tiedot sekä muutettaville kohteille syötekentät, joissa muutettavia kohteita pystyi muuttamaan.

```
<md-content ng-controller="CciParametersCtrl" layout="column" layout-padding flex="80">

<h2 class="md-display-1">Edit common parameters</h2>
<md-content class="md-padding" layout="column">
<form name="ParameterForm" ng-submit="$event.preventDefault()">
<div layout-xs="row" flex-xs="100">
<table>
<td width="120"><b>Parameter</b></td>
<td width="120"><b>Parameter group</b></td>
<td width="90"><b>Value</b></td>
<td width="150"><b>subgroup</b></td>
<td width="120"><b>deliverychannel</b></td>
<td width="60"><b>Overwrite</b></td>
<td><b>Comment</b></td>
<tbody>
  <tr ng-repeat="x in ParameterGroups">
    <td>{{ x.parametername }}</td>
    <td>{{ x.parametergroup }}</td>
    <td>
    <md-input-container flex="50">
    <input ng-model="x.value" required>
    </md-input-container>
    </td>
    <td>{{ x.Subgroup }}</td>
    <td>{{ x.deliverychannel}}</td>
    <!--
    <td>
    <md-select ng-model="x.overwrite" required="true">
    <md-option class="mini" ng-repeat="val in yesno" value="{{val}}}">{{val}}</md-option>
    </md-select>
    </td>
    -->
     <td>
    <md-input-container flex="35">
    <input ng-model="x.ovewrite" required>
    </md-input-container>
    </td>
    <td>
    <md-input-container flex="">
    <input ng-model="x.comment">
    </md-input-container>
    </td>
  </tr>
 </tbody>
</table>
</form>
```

KUVA 3 Parametrien muuttamiseen käytettävän käyttöliittymän ensimmäisen version lähdekoodia

Yleisten parametrien muuttamisen tarkoitetun controllerin tehtävänä on hakea servicen kautta käyttöliittymään tulostettavaksi olio, joka sisältää kaikki yleiset parametrit ja niiden tiedot, sekä tallennettaessa lähettää muuttunut olio takaisin servicelle tietokantaan muutosta varten. Kyseistä controlleria varten ei tehty omaa serviceä, vaan parametrien haku ja tallennus toteutettiin yhteiseen cciInterface-serviceen, johon rakennettiin kaikki asiakkuuksien hallintaan liittyvät taustapalvelut. Kuvassa 4 on yleisten parametrien tallentamiseen käytetyn funktion lähdekoodi.

```
angular.module('app').controller('CciParametersCtrl',function($scope, $http, cciInterface){

// Hakee CCIn servicellä yleiset parametrit
$scope.ParameterGroups = cciInterface.getCommonParameters();

$scope.SaveParameterForm = function (x)
{
    // Lähetetään muuttunut objekti takaisin serviselle, jossa se voidaan tallentaa kantaan
    cciInterface.setCommonParameters($scope.ParameterGroups);
};
});
```

KUVA 4 Parametrien muuttamisen käytettävän controllerin lähdekoodi

SPEC-tiedosto on komponenttien testaamiseen tarkoitettu tiedosto, johon ohjelman tekijä voi kirjoittaa testitapauksia, joiden avulla varmistetaan, että käyttöliittymä ja controller toimivat yhteen. Asiakashakua varten kirjoitettiin testitapaus, jolla varmistettiin, että komponentille tuleva data on oikeassa muodossa. Grunt suorittaa nämä kirjoitetut testit silloin, kun ohjelma avataan, ja ilmoittaa testien tuloksen. Testitapaukseen kirjoitetaan testin kuvaus sekä haluttu vastaus. Mikäli kyselyn tulos täsmää halutun vastauksen kanssa, testi merkataan onnistuneeksi.

```
it('should get a filtered list of customers', inject(function()
{
        httpBackend.expect('GET', u.getService('cciguiapi').url+"/customers")
        .respond(function(method, url, data, headers, params)
        {
            var result = [{CustId:1,name:"Cust1"},{CustId:2,name:"Kust1"}];
            var status;
            return [200, result];
        });

        httpBackend.flush();
        var filteredCustomers = scope.querySearch("Cust");
        expect(filteredCustomers.length).toEqual(2);
}));
```

KUVA 5 Testi, jolla tutkitaan asiakaslistan suodatusta. Testin haluttu tulos on tarkoituksella merkattu väärin.

Kuvassa 5 olevassa testissä testataan asiakaslistan suodattamista. Testitapauksessa palautetaan suodattavalle funktiolle asiakaslista, joka sisältää kaksi asiakasta, joiden nimet ovat Cust1 ja Kust1. Tutkittavana on suodatuksen toimivuus ja testitapaukseksi on kirjoitettu, että listasta halutaan suodattaa sellaiset asiakkaat, joiden nimessä on Cust. Koska kyseessä on tarkoituksellisesti virheelliseksi tehty testitapaus, odotetaan lopputuloksena listaa, jonka pituus on kaksi asiakasta. Koska palautuvan listan pituus eroaa odotetusta vastauksesta, tulee testiä suorittaessa virheilmoitus. Tämä virheilmoitus on esitelty kuvassa 6.

```
Start:
  CciInterfaceCtrl
    V should get a list of customers
V     » should get a filtered list of customers
»
Finished in 0.015 secs / 0.774 secs

SUMMARY:
V 1 tests completed
» 1 tests failed

FAILED TESTS:
  CciInterfaceCtrl
    » should get a filtered list of customers
      PhantomJS 2.1.1 (Windows 7 0.0.0)
    Expected 1 to equal 2.

Warning: Task "karma:during_watch" failed.
```

KUVA 6 Virheilmoitus epäonnistuneesta testistä

3.3.2 Service

Service on puolestaan taustajärjestelmän eli tietokannan ja käyttöliittymän välille tehtävä yleiskäyttöinen komponentti, jota voidaan hyödyntää myös muissa toteutuksissa saman sovelluksen sisällä. Tässä tapauksessa CCI:n REST APIn käyttöä varten luotiin oma servicensä, johon rakennettiin kaikki asiakkuuden hallintaan käytettävät tietokantakyselyt. Myös servicen luominen onnistui angularin generaattorilla, johon komento oli

```
yo cg-angular:service cci-interface
```

Komento loi service-hakemiston alle uuden kansion ja tähän kansioon Javascript- ja spec-tiedostot.

Serviceen luodaan resursseiksi yhteydet tietokantakyselyjä varten. Resurssi sisältää osoitteen, josta tietoa haetaan, ja kyselyn määrittelyn. Resurssin osoite muodostetaan ympäristömuuttujasta ja tarkenteesta. Ympäristömuuttujaa hyödynnetään, kun siirretään sovellus kehityspuolelta tuotantoon, jolloin ohjelman yleisiin muuttujiin määritetään käytettäväksi tuotantoympäristö. Kyselyihin ei tarvitse tässä vaiheessa enää tehdä muutoksia, vaan ne ohjautuvat automaattisesti tuotantoalustaan. Kyselyihin määritetään kyselyn tyyppi, joita ovat POST, GET, PUT ja DELETE. Hakukyselyiden tyyppi on GET, ja kun tietoa tallennetaan, valitaan POST tai PUT sen mukaan, luodaanko uutta vai päivitetäänkö vanhaa tietoa. Resurssiin määritellään lisäksi, onko vastauksena saatava tieto taulukko vai ei. Resurssiin voidaan määrittää tämän lisäksi myös hakuehto, esimerkiksi tietyn asiakkaan id. Asiakaslistan hakemiseen käytetty resurssi muodostetaan seuraavasti:

```
var allCustomers =
$resource(env.cciguiapi+"/customers/", {'query': {method:'GET', isAr-
ray:false}});
```

Resurssin käyttöä varten luodaan servicen sisään funktio, jota voidaan kutsua partialin sisällä olevasta controllerista. Funktion sisällä luodaan muuttuja, jonka arvoksi tulee tietokantakyselyn palauttama objekti. Kysely suoritetaan käyttämällä haluttua resurssia ja sille määritettyä methodia eli

```
var Tulos = allCustomers.get (function () {});
```

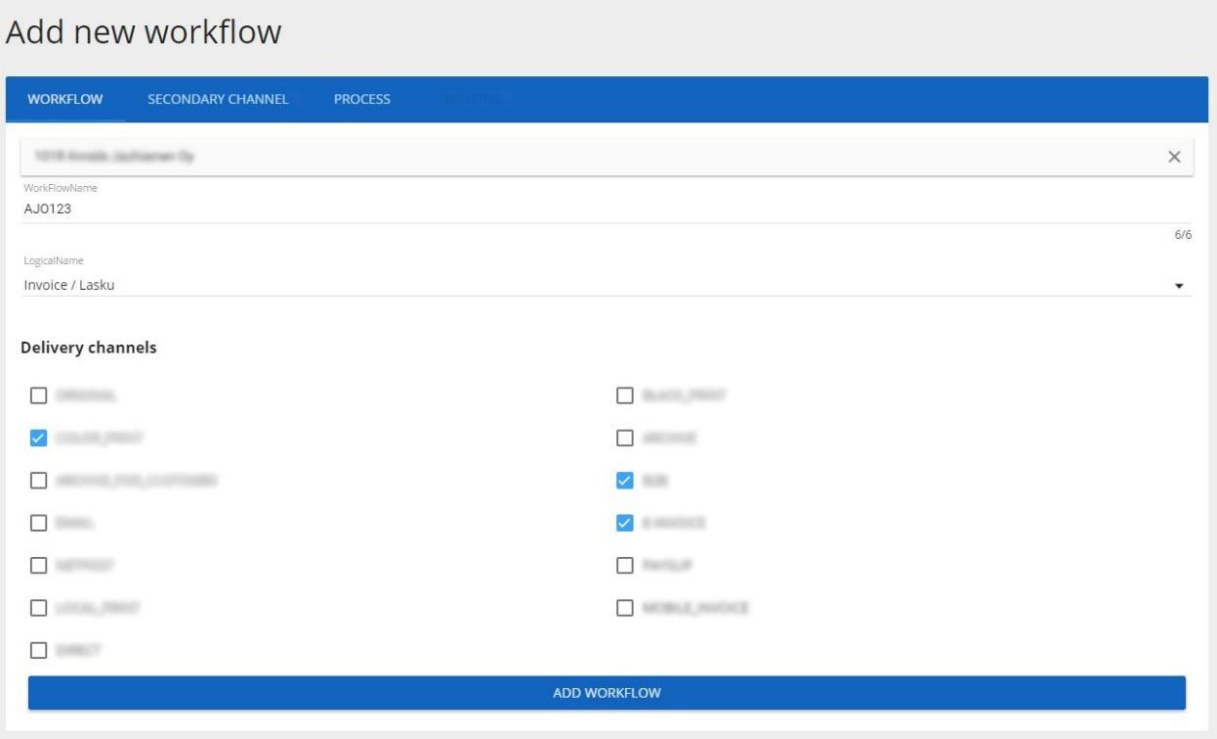
Myös servicellä on oma spec-osionsa, jonne voidaan kirjoittaa tietokantakyselyiden toiminnallisuuksiin liittyviä testejä.

4 SOVELLUKSEN TOIMINNOT

4.1 WorkFlow'n lisääminen

Uuden workflow'n lisääminen tapahtuu "Add workflow" -ominaisuuden kautta. Tämä ominaisuus sisältää sarjan näyttöjä, jotka käydään vaihe vaiheelta läpi annetussa järjestyksessä. Jokaisen näytön vaadittavat syötteet tarkastetaan, ja seuraavaan vaiheeseen pääsee vasta, kun kaikki tiedot ovat kunnossa. Jokainen näyttö luodaan käyttäjän valintojen perusteella, jolloin samoja toimintoja voidaan käyttää myös esimerkiksi workflow'n muokkauksessa.

Ensimmäinen näyttö sisältää workflow'n perustiedot. Tässä näytössä on alasvetovalikko, johon haetaan tietokannasta kaikki asiakkaat. Tästä listasta valitaan asiakas, jolle uusi workflow lisätään. Tämän jälkeen käyttäjä antaa lisättävän workflow'n perustiedot eli nimen sekä aineistosisällön kuvaukseen käytettävän loogisen nimen. Loogiset nimet haetaan tietokannasta samaan tapaan kuin asiakkaatkin ja sekä asiakastiedot että looginen nimi lisätään uuden workflow'n olioon omina objekteinaan. Nämä objektit sisältävät asiakkaan tai loogisen nimen selkeäkielisen nimitiedon sekä yksilöivän ID:n. Kun nämä kohdat on täytetty, valitaan käsittelystä jakaantuvat tulosaineistot. Tulosaineistoja ovat esimerkiksi paperille menevät laskut sekä sähköisiin kanaviin jaettavat laskut. Ensimmäisen näytön sisältö on esitetty kuvassa 7.

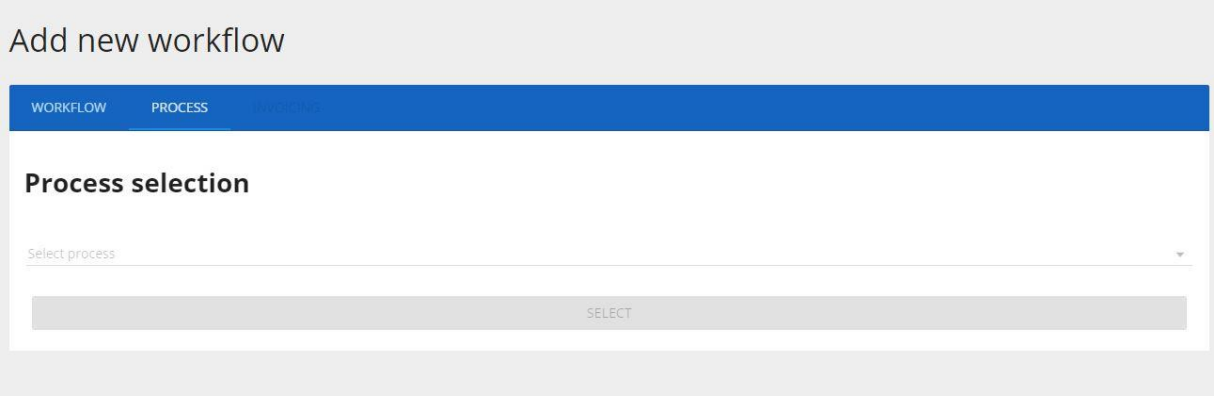


KUVA 7: Sovelluksen ensimmäinen näyttö, jossa annetaan workflow'n perustiedot

Secondary channel näytöllä voidaan valita jokaiselle ensimmäisessä näytössä valitulle jakelukanavalle pudotusvalikosta vaihtoehtoinen jakelukanava. Vaihtoehtoista jakelukanavaa käytetään silloin, kun dokumenttia ei voida toimittaa pääasialliseen kanavaan. Tällainen tilanne voi syntyä esimerkiksi

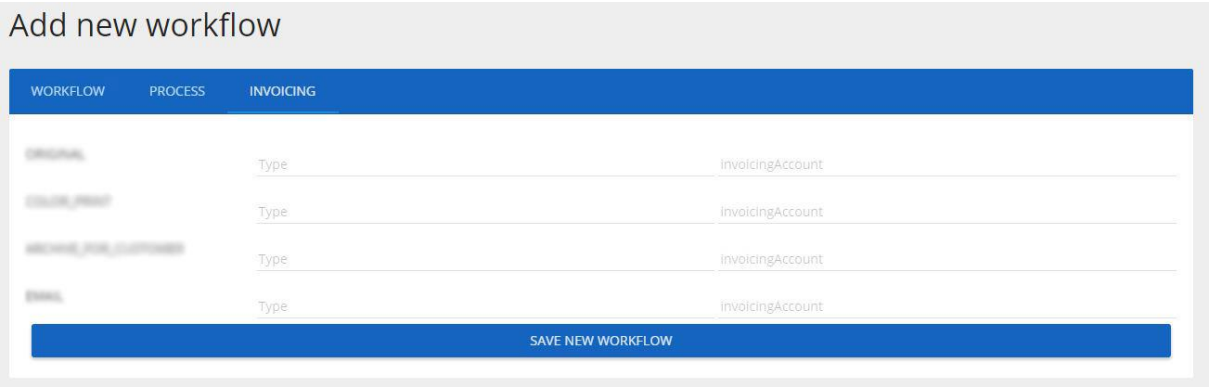
silloin, kun pankkiin toimitettavan sähköisen laskun reititystiedot ovat virheelliset. Tässä tilanteessa sähköinen lasku ohjattaisiin vaihtoehtoiseen kanavaan, joka voi olla esimerkiksi paperitulostus.

Kuvassa 8 esitetyssä näytössä valitaan prosessi, jonka mukaisesti kyseisen workflow'n aineistot käsitellään. Prosesseja on valittavana useita erilaisia, ja jokaisessa käsitellään annettua aineistoa eri tavoilla.



KUVA 8 Prosessin valinta (tarkastamaton näyttö, jolloin jatkaminen seuraavaan ei onnistu)

Prosessin valitsemisen jälkeen siirrytään viimeiseen näyttöön (kuva 9). Tässä näytössä annetaan workflow'n valituille jakelukanaville nimet ja laskutustiedot. Tulosaineistot ja workflow't nimetään vakiokäytännön mukaisesti. Workflow'n nimi on kuusi merkkiä pitkä, ja se sisältää asiakaskohtaisen lyhenteen sekä numeron. Ajonimi on siis muodossa AJO120. Jokainen ajosta syntyvä tulosaineisto eli eri jakelukanaviin jaettavat aineistot nimetään tämän lisäksi vielä siten, että siitä on tunnistettavissa workflow sekä jakelukanava. Tulosaineistojen nimet olisivat siis esimerkkitapauksessa mallia AJOx12, jossa x olisi jakelukanavakohtainen tunniste.



KUVA 9 Laskutustietojen antaminen valituille jakelukanaville

Käyttäjän antamat tiedot ja valinnat ovat koko ajan tallessa JSON-objektissa, jota muokataan jokaisessa näytössä. Viimeisen näytön save-painikkeella tämä objekti lähetetään servicelle, joka tallentaa workflow'n tiedot tietokantaan. Tietokantakysely on rakennettu siten, että uusi worfkflow voidaan tallentaa tietokantaan suoraan JSON-objektista eikä erillistä objektin purkamista enää tarvita.

4.2 Olemassa olevan workflow'n muuttaminen

Olemassa olevan workflow'n muuttaminen tapahtuu samalla näytöllä kuin uuden lisääminen. Erona uuden lisäämiseen on, että muokkaukseen tultaessa valitaan ensin haluttu workflow. Tämän jälkeen kyseisen workflow'n tiedot haetaan servicen kautta käyttöliittymän kontrollerissa olevaan objektiin. Tämä objekti on sama, jota käytetään uuden workflow'n luomisessa. Erona muokkauksessa on se, että objekti esitäytetään workflow'n tiedoilla. Käyttöliitymän komponentit on rakennettu siten, että niissä on kaksisuuntainen tiedon sitominen tähän objektiin. Näin valitun workflow'n tiedot ovat valmiiksi täytettyinä käyttöliittymän komponenteissa, kun tiedot on saatu haettua. Näitä voidaan tämän jälkeen muuttaa samalla tavalla kuin uutta workflow'ta lisätessä. Jokaisen näytön tiedot tarkastetaan ennen seuraavaa vaihetta. Workflow'ta ei siis voida muuttaa sellaisella tavalla, että se tallennettaisiin tietokantaan virheellisenä.

Käyttöliittymässä muokataan koko ajan olemassa olevaa oliota, joten muutokset voidaan tallentaa missä vaiheessa tahansa. Workflow'ta muokatessa ei siis tarvitse käydä läpi koko prosessia yhden muutoksen takia, vaan näytöllä voidaan suoraan hypätä haluttuun vaiheeseen, tehdä muutos ja tallentaa workflow muutettuna.

Olemassa olevan käsittelyn muutosta ei ehditty toteuttaa täysin valmiiksi työn aikana, sillä workflow'n tietojen hakemista ei ehditty toteutettaa. Tämän vuoksi toiminnallisuuden testaaminen jäi vajaaksi.

4.3 Yleisten parametrien muuttaminen

Yleisten parametrien hallintaan käytettävästä näytöstä ehdittiin tehdä kaksi erilaista versiota. Ensimmäinen versio oli yksinkertaisempi näyttö, jossa yleiset parametrit tulostettiin käyttäjän muokattavaksi taulukkonäkymään. Työn lopussa ehdittiin vielä suunnitella ja toteuttaa helppokäyttöisempi ja käyttäjäystävällisempi versio, jota voidaan hyödyntää myös asiakaskohtaisten parametrien hallinnassa.

4.3.1 Ensimmäinen versio

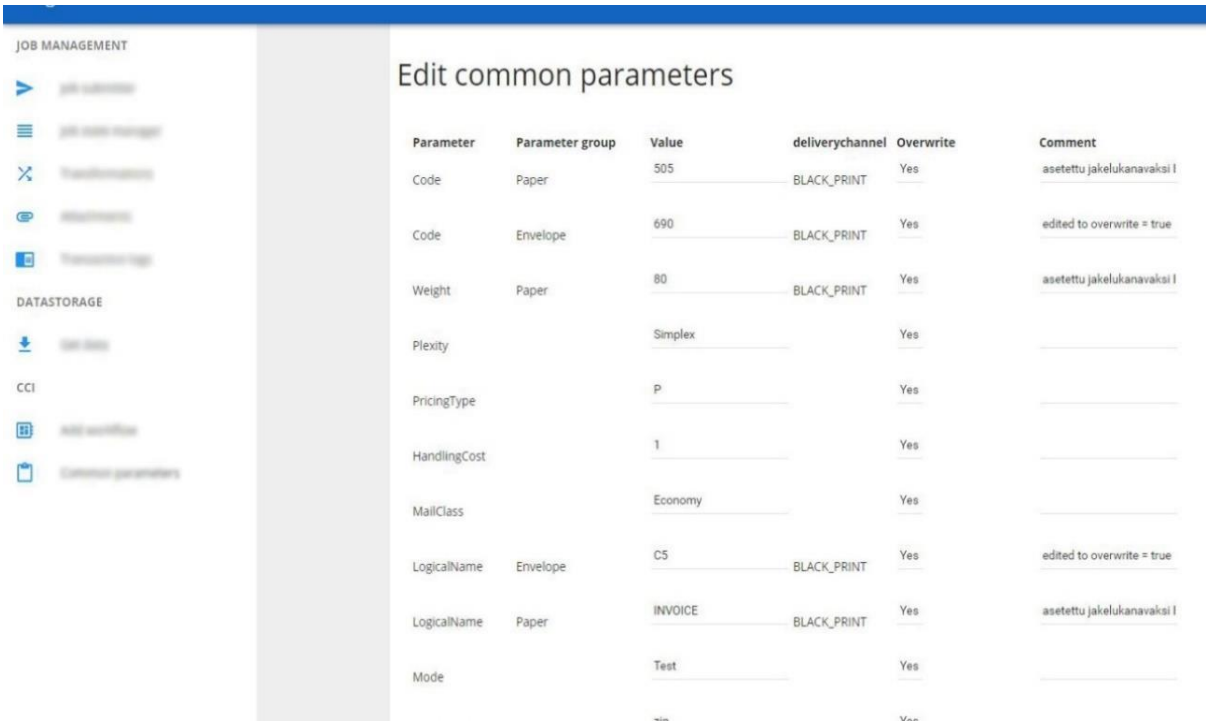
Yleisiä parametreja muutetaan ”Common parameters” -ominaisuuden kautta. Tähän ikkunaan tultaessa haetaan ensin servicen kautta tietokannasta kaikki yleiset parametrit kontrollerissa olevaan objektiin. Objektin sisältö koostuu siis yhdestä tai useammasta parametrista, joilla on parametrin ja parametriryhmän yksilöivät tiedot. Objekti, jossa on yksi parametri, olisi siis kuvan 10 kaltainen.

```
parameters
[
    {
        id: 1
        name: Paperi
        ryhmä: printti
        arvo: a4
        aliryhmä id: 5
        jakelukanava: tulostus
        ylikirjoittava: kyllä
        kommentti: Arvoa muutettu 1.1.2016
    }
]
```

KUVA 10 Parametri-objektin sisältö

Samalla parametriryhmän nimellä voi kuitenkin olla useita eri käytössä olevia parametrejä. Esimerkiksi väritulosteella ja mustavalkotulostuksella on molemmilla erilliset parametrit paperille. Nämä molemmat kuitenkin ovat tietokannassa nimellä paperi. Tällaiset tapaukset tunnistetaan aliryhmän ID:n perusteella ja tätä ID:tä käytetään näytössä myös parametrien loogiseen ryhmittelyyn. Kun samalla aliryhmän ID:llä olevat parametrit sijaitsevat näytössä allekkain, syntyy järkevä kokonaisuus.

Kun objekti on muodostettu, sen arvot tulostetaan taulukkoon käyttäjän muokattavaksi. Sellaiset kentät, joiden sisältöä ei ole mahdollista muuttaa, tulostetaan tekstikenttinä, kun taas muutettavat arvot tulostetaan syöte-elementtinä. Sellaiset kentät, joiden arvoja on mahdollista muuttaa, tarkastetaan ennen kuin tallentaminen on mahdollista. Mikäli käyttäjä siis poistaa vaaditun arvon, ei parametrien tallentaminen enää ole mahdollista vaan puutteet on ensin korjattava. Angularin kaksisuuntaisen arvojen sitomisen ansiosta käyttöliittymässä käsitellään koko ajan objektin arvoja, joten muutokset tulevat suoraan objektiin. Tallennusvaiheessa lähetetään muuttunut objekti takaisin servicelle tietokantaan tallennettavaksi.

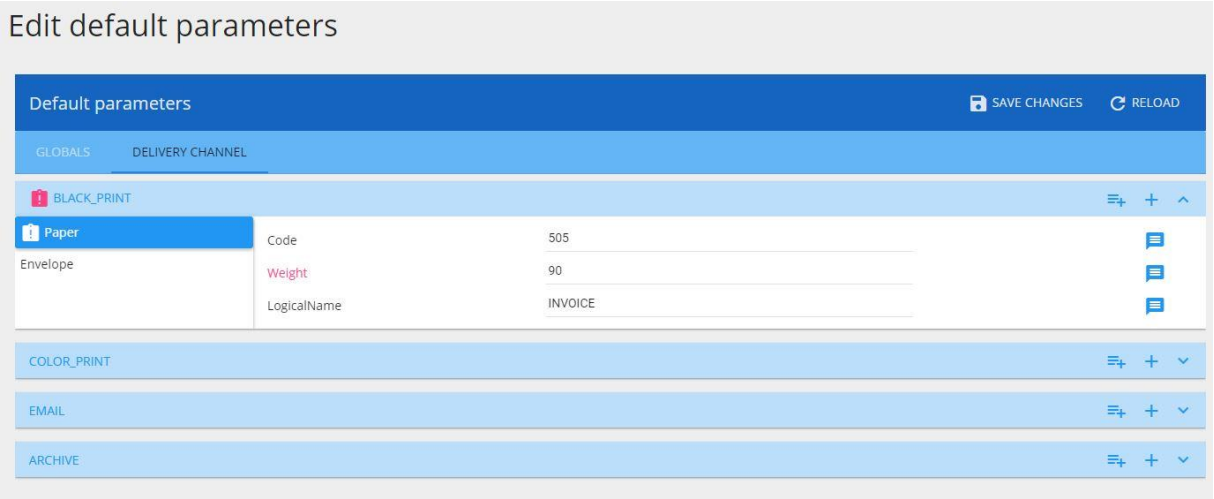


KUVA 11 Yleisten parametrien muuttamiseen käytettävä näyttö (ensimmäinen versio)

4.3.2 Toinen versio

Projektin loppuvaiheessa saatiin vielä suunniteltua parametrien hallinnasta toinen versio. Tässä versiossa on helppokäyttöisempi ja mukautuvampi käyttöliittymä, joten tätä komponenttia voidaan jatkossa käyttää myös asiakaskohtaisten parametrien hallintaan. Uudessa versiossa näkymä muuttui taulukkonäkymästä komponenteista rakennetuksi. Uusi näyttö rakennettiin direktiiviksi eli tätä komponettia voidaan käyttää myös muissa näytöissä, joissa parametrejä halutaan hallita, esimerkiksi asiakaskohtaisten parametrien hallinnassa.

Tälle näytölle annetaan parametrina olio, joka sisältää muokattavat parametrit. Nämä ryhmitellään aliryhmän perusteella osioiksi. Yksi osio sisältää aina kaikki sellaiset parametrit, jotka liittyvät tähän osioon. Näyttö on jaoteltu kahteen osaan, globaaleihin ja jakelukanavakohtaisiin parametreihin. Globaaleja parametreja ovat sellaiset parametrit, jotka vaikuttavat kaikkiin jakelukanaviin. Näytölle tulostetaan osioita allekkain sen mukaan, mitä parametrina annettu olio sisältää. Kuvan 12 tapauksessa käsitellään jakelukanavakohtaisia parametreja ja tässä tapauksessa jakelukanavia on black print, color print, email ja archive.
Muutetut arvot näytetään huomiovärillä. Tämän lisäksi näytetään muuttuneesta tiedosta ilmoittava huomio jakelukanavan nimen vieressä.



KUVA 12 Parametrien hallinta, uusi versio

Kun käyttäjä on muuttanut haluamansa parametrit, hän painaa tallennuspainiketta. Tällöin käyttäjälle näytetään yhteenveto muutetuista parametreista ja pyydetään vahvistamaan muutokset. Vahvistamisen jälkeen muutetut parametrit tallennetaan tietokantaan.

5 YHTEENVETO JA POHDINTA

Työn tavoitteena oli saada valmiiksi sovellus, jolla on pystyttävä lisäämään uusi workflow tietylle asiakkaalle sekä hallitsemaan yleisiä laskunvälityksen parametrejä. Näihin tavoitteisiin päästiin, sillä molemmat toiminnallisuudet saatiin valmiiksi. Koska workflow'n lisäämiseen käytettävää sovellusta voidaan käyttää myös olemassa olevan workflow'n muuttamiseen, päästiin myös hieman asetettujen tavoitteiden yli. Alussa hahmoteltuja toiminnallisuuksia versionhallinnan toiminnallisuuksia ei kuitenkaan saatu tämän projektin puitteissa toteutettua. Näiden toiminnallisuuksien valmistumisen lisäksi havaittiin useita jatkokehitysideoita, jotka tullaan toteuttamaan.

Työn aikana ongelmia aiheuttivat ajan puute sekä käytettävissä olevan työajan resurssointi. Ohjelman kehittäminen tapahtui päivittäisen työn ohella, joten irtaantuminen kehitystyöhön ei aina onnistunut halutulla tavalla. Ohjelman kehitys aloitettiin päivittäisten työtehtävien suhteen vuoden kiireisimpään aikaan, joten tässä projektissa alkuun pääseminen oli hidasta. Angularin ja muiden tekniikoiden opettelu sekä ajan puute johtivat siihen, että ensimmäisten kuukausien ajan eteneminen oli käytännössä olematonta. Työtä ohjaavilla työntekijöillä taas oli kiirettä uuden palvelualustan kehittämisessä. Yhteisen ajan löytäminen uusien työvaiheiden suunnitteluun ja etenemisen seuraantaan oli haasteellista.

Työn ansiosta opin paljon uusia tekniikoita ja menetelmiä web-kehityksestä. Ohjelman kehittämiseen käytetty Angular oli ennen projektia täysin tuntematon, mutta työn aikana se tuli tutuksi. Työn aikana kävi selväksi scurmin hyödyt tällaisessa projektissa. Kun tehtävät jaettiin pieniin osaan, työ eteni nopeammin. Opin myös sen, miten tärkeää on käytettävissä olevan ajan määrittäminen tarkasti sekä ajan käytön resurssointi. Mikäli kehitystä tehdään silloin, kun sille jää muilta töiltä aikaa, aikaa ei koskaan ole. Kun käytettävät tunnit määritetään tarkasti ja pysytään tässä suunnitelmassa, saadaan paljon enemmän aikaiseksi.

Projektin alussa suunniteltiin alustavasti, että mikäli halutut toiminnot valmistuvat etuajassa, voidaan sovellukseen lisätä myös olemassa olevien käsittelyiden ohjaustiedostojoen hakeminen, muokkaus sekä tallentaminen GIT-hakemistosta, jotta saadaan näiden tiedostojen muodostamiseen intergroitua versionhallinta. Näitä toimintoja ei kuitenkaan ehditty projektin aikataulussa toteuttamaan, joten ne on sovittu jatkokehityskohteiksi. Seuraavaksi kehityskohdaksi on sovittu asiakaskohtaisten parametrien hallinta. Tämän toiminnallisuuden hahmottaminen on monimutkainen kokonaisuus, sillä parametrejä voi olla kaikkiin asiakkaan käsittelyihin vaikuttavia eli asiakaskohtaisia-, sekä yhtä käsittelyä ohjaavia eli käsittelykohtaisia parametreja. Tämän lisäksi on vielä parametrejä, jotka ohjaavat jotain tietyn asiakkaan, tietystä käsittelystä syntyvää tiettyä tulosaineistoa.

Mielestäni työ onnistui hyvin, sillä tavoitteisiin päästiin ja pääsin kehittämään omaa osaamistani uusien tekniikoiden avulla.

6 LÄHTEET

Amin, M. R. (ei pvm). *Backendless development with AngularJs*. Haettu 31. Maaliskuu 2016 osoitteesta https://ruhul.wordpress.com/2014/11/03/backendless-development-with-angularjs/

Enfo Oyj. (2016). Tilinpäätöstiedote. Noudettu osoitteesta http://www.enfo.fi/-/media/Default-Website/Financial-Statement/Enfo15_tilinptstiedote_final.ashx

GRUNT The JavaScript Task Runner. (ei pvm). Haettu 6. Huhtikuu 2016 osoitteesta http://gruntjs.com/

Introduction to AngularJS. (ei pvm). Haettu 6. Huhtikuu 2016 osoitteesta w3schools: http://www.w3schools.com/angular/angular_intro.asp

Notepad++ Home. (ei pvm). Haettu 29. Maaliskuu 2016 osoitteesta https://notepad-plus-plus.org

Postman - Supercharge your APIs. (ei pvm). Haettu 6. Huhtikuu 2016 osoitteesta https://www.getpostman.com/

The web's scaffolding tool for modern webapps. (ei pvm). Haettu 6. Huhtikuu 2016 osoitteesta http://yeoman.io/

What is Mule ESB. (ei pvm). Haettu 30. Maaliskuu 2016 osoitteesta https://www.mulesoft.com/resources/esb/what-mule-esb

Wikipedia. (ei pvm). *AngularJS*. Haettu 6. Huhtikuu 2016 osoitteesta https://fi.wikipedia.org/wiki/AngularJS

Wikipedia. (ei pvm). *Git (software)*. Noudettu osoitteesta https://en.wikipedia.org/wiki/Git_(software)

Wikipedia. (ei pvm). *HTML*. Haettu 5. Huhtikuu 2016 osoitteesta https://fi.wikipedia.org/wiki/HTML

Wikipedia. (ei pvm). *Javascript*. Haettu 5. Huhtikuu 2016 osoitteesta https://fi.wikipedia.org/wiki/JavaScript

Wikipedia. (ei pvm). *JSON*. Haettu 5. Huhtikuu 2016 osoitteesta https://fi.wikipedia.org/wiki/JSON

Wikipedia. (ei pvm). *Representational state transfer*. Noudettu osoitteesta https://en.wikipedia.org/wiki/Representational_state_transfer

Wikipedia. (ei pvm). *Scrum*. Haettu 27. Maaliskuu 2016 osoitteesta https://fi.wikipedia.org/wiki/Scrum

Wikipedia. (ei pvm). *Testivetoinen kehitys*. Haettu 31. Maaliskuu 2016 osoitteesta https://fi.wikipedia.org/wiki/Testivetoinen_kehitys