LAUREA AMMATTIKORKEAKOULU Uuden edellä

Selaimessa toimivan tiedonmallinnustyökalun kehittäminen

Case: Aava Ohjelmistot Oy

Lemmetyinen, Aleksi

2015 Kerava

Laurea-ammattikorkeakoulu
Kerava

Selaimessa toimivan tiedonmallinnustyökalun kehittäminen
Case: Aava Ohjelmistot Oy

Aleksi Lemmetyinen
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Joulukuu, 2015

Laurea-ammattikorkeakoulu **Tiivistelmä**
Kerava
Tietojenkäsittelyn koulutusohjelma

Aleksi Lemmetyinen

Selaimessa toimivan tiedonmallinnustyökalun kehittäminen. Case: Aava Ohjelmistot Oy

Vuosi 2015 Sivumäärä 27

Opinnäytetyössä toteutettiin ohjelmisto toimeksiantajalle toiminnallisena kehityshankkeena, jonka lopputuotteena syntyi tiedonmallinnukseen käytettävän työkalun käyttöliittymä. Työkalun tavoitteena oli korvata vanha työkalu, joten siltä edellytettiin vähintään vanhan työkalun toiminnallisuuden täyttävää tasoa. Kehityksessä käytettiin web-kehitysmenetelmiä, jotka valittiin joko toimeksiantajan tai toteuttajan toimesta ja niiden valinnat perustellaan, sekä kuvataan vaihtoehtoisia toteutustapoja kun niitä havaittiin.

Opinnäytetyö esittää kehitystyöhön käytetyt teknologiat ja kehitystyössä kohdatut haasteet sekä hankkeen lopputuloksena syntyneen tiedonmallinnusjärjestelmänkäyttöliittymän. Työ käsittelee millä periaattein käyttöliittymä vuorovaikuttaa käyttäjän kanssa sekä miten sen mahdollistava ohjelmistoarkkitehtuuri on koostettu rakenteellisesti. Pääosassa toteutusta esiintyivät selaimen tarjoamat ohjelmointirajapinnat Document Object Model, Ui Events ja Cascading Stylesheets, joihin käyttöliittymän toiminnot sekä tietorakenteet kytkettiin. Käyttöliittymälogiikan teossa apuna hyödynnettiin Backbone.js JavaScript-kirjastoa.

Käyttöliittymänohjelmoinnin haasteina ratkaistiin käyttöliittymän toiminnan monimutkaisuus koostamalla käyttöliittymän komponentit loogisiin kokonaisuuksiin näkymäolioilla Backbone.js JavaScript-kirjaston avulla. Muistivuodot korjattiin poistamalla yhteydet vanhoihin olioihin kun niitä ei enää tarvittu, jolloin roskienkerääjä pystyi vapauttamaan muistia tehokkaammin takaisin käyttöön ohjelmansuorituksen aikana. Suorituskykyongelmia kohdattiin suurten oliomäärien tapauksissa. Suorituskykyongelmia korjattiin suorittamalla pullonkaulaksi profiloituihin kohtiin tarpeellinen määrä lähdekoodin optimointia ja muokkausta, kun se ohjelman riittävän toiminnan- ja käytettävyydentason säilyttämisen kannalta katsottiin tarpeelliseksi.

Toteutuksessa hyödynnettiin apuvälineinä muun muassa Git-versionhallintaa sekä vapaanlähdekoodin ohjelmistokirjastoja, joilla työkalun kehitystyötä nopeutettiin ja tehostettiin, ja työnlaatua parannettiin.

Hankkeen lopputuloksena toimeksiantaja korvasi asiantuntijoidensa aiemmin käyttämän vanhan työkalun projektissa toteutetulla uudella mallinnustyökalulla. Työkalua hyödynnettiin heti sen valmistuttua myös toimeksiantajan ruotsalaisen yhteistyöyrityksen taholla osana heidän järjestelmätoimitushankettaan.

Asiasanat: Tiedonmallinnus, Backbone.js, Node.js, ohjelmistokehitys, käyttöliittymä

Laurea University of Applied Sciences
Kerava
Business Information Systems

Aleksi Lemmetyinen

Abstract

Developing a browser based data modeling application

Year 2015 Pages 27

The employer was in need of a data modeling software to replace the old legacy desktop application tool named ArgoUML, written in Java programming language, used in the company. This thesis investigated what different technologies and practices were used and available to craft a replacement for the old tool. The replacement tool produced is a browser ran JavaScript application primarily designed for Unix compliant systems and it featured several open source JavaScript libraries, most importantly the Backbone.js client-side library. The product had to have all the features included in the old tool to meet the completion criteria for the project set by the employer. In some cases there were other technology options available for the development of a particular feature and the reasons for the resulting selections are explained thoroughly.

Difficulties with extensibility and performance were encountered in the project and solutions were developed to address these concerns. It was found that by focusing attention on how the user interface is structured plays an important role in achieving acceptable performance characteristics through keeping the created object count during runtime as low as possible. It was specified that the tool needs to extensible and this influenced the design solutions in the project. The browser Events and Document Object Model browser application programming interfaces were exploited heavily in the browser application to improve the quality and usability of the product.

The tool was developed with resources, guidance and specifications provided by the employer Aava Software. The projects result was a success and the software produced was adopted into the use by the company almost immediately after the project was concluded.

Keywords: Backbone.js, Node.js, user interface, data modeling, software development

Sisällys

1 Johdanto

Tavoitteenani oli toteuttaa tietojärjestelmä, jota yritys pystyy hyödyntämään tiedonmallintamiseen graafisesti symbolein ja värein, joka pyrkii tukemaan tiedonmallintamisen tehokkuutta omaksuttavuudella. Toteutettavat ohjelmallisettoiminnot, testit ja dokumentaatio toteutettiin Lean- sekä Agile-ohjelmistokehitysmallien yhdistelmän periaatteita soveltavalla projektikehitysmenetelmällä. Tämä opinnäytetyö keskittyy käsittelemään mallinnuskäyttöliittymän toteutusta erityisesti teknisestä näkökulmasta.

Tarkoituksena oli toteuttaa ohjelma, jolla pystytään korvaamaan vanhastaan käytössä oleva ArgoUML-ohjelmisto. joka ei ominaisuuksiltaan ja laajennettavuudeltaan enää vastaa yrityksen tarpeisiin. Sen käyttäminen on kömpelöä ja hidasta kun usein käytettyjä toimintoja joutuu etsimään sekavista monikerroksisista valikoista. Uuden työkalun osalta tavoitteena oli, että se korvaisi ArgoUML:n ja olisi parannus siihen nähden käytettävyydeltään ja erikoisominaisuuksiltaan. Ohjelman ulkonäkö ei ollut enää nykypäivää vaan se näytti rumalta ja vanhanaikaiselta.

ArgoUML on vapaanlähdekoodinohjelma ja sen koodikanta käsittää tuhansia rivejä Javaa. Ohjelmiston suuri koko saa aikaan sen, että uusien ominaisuuksien suunnittelu ja toteutus on hidasta ja virhealtista sillä regressioita, eli uuden toiminnallisuuden lisäämisestä aiheutuvia virheitä vanhoihin toiminnallisuuksiin, voi syntyä helposti kun vanhaa koodia rupeaa muuttamaan. Uuden mallinnus käyttöliittymän tuli olla edeltäjäänsä helpommin laajennettavissa uusin ominaisuuksin, sekä koodin tuli olla tiiviimpää ja luettavampaa, siltä varalta, että projektia ryhtyy kehittämään joku uusi ohjelmoija.

Moderni ulkoasu oli tärkeä, sillä voidaan saada kolmannet osapuolet, esimerkiksi kumppaniyritykset, kiinnostumaan yrityksen teknologiasta jo ensisilmäyksellä. Monet suuret ohjelmistoyritykset kuten Apple ja Samsung ovat osoittaneet käyttöliittymien ulkoasun visuaalisen vetovoiman merkityksen tuotteen onnistuneessa markkinoinnissa ja myynnissä.

Ohjelman tuli sisältää sen käyttöä nopeuttavia ominaisuuksia kuten ennustavaa tekstinsyöttöä sekä näppäinoikoteitä. Ohjelman käytettävyyden ei tarvinnut olla niin hyvä, että sen eteen olisi jouduttu tekemään kompromisseja ominaisuuksien määrässä ja laadussa. Tämä johtuu siitä, että tämä projektityönä toteutettava uuden ohjelmiston graafinen käyttöliittymä tulee ensisijaisesti yrityksen omien asiantuntijoiden eikä maallikoiden käyttöön.

Ohjelma perustuu kolmansien osapuolten ohjelmistokomponenttien osalta vapaanlähdekoodin ohjelmistoihin, jotka sallivat niiden vapaan käytön yrityksen liiketoiminnassa.

2 Tutkimusmenetelmä ja aiheen rajaaminen

Tutkimusmenetelmänä käytettiin toiminnallista tutkimusta sillä koin, että sen kautta voisin parhaiten kehittää ohjelmistokehitysosaamistani ja yritykselläni oli kova tarve toteuttaa korvaava työkalu vanhalle nopeassa aikataulussa. Koulutukseni ajan painotettiin oppimista kehittämisen kautta, joten periaatteet olivat hyvin tuttuja ja hyvin hallussa. Osaamiseni kehittyi mielestäni hankkeen koko elinkaaren ajan joten pidän tutkimusmenetelmä valintaani onnistuneena.

Toimeksiannon tarkoitus oli kehittää korvaaja ArgoUML-työkalulle. Tavoitteeksi otettiin toteuttaa samat toiminnallisuudet uuteen työkaluun kuin mitä vanhassa työkalussa oli. Tässä opinnäytetyössä selvitetään mistä komponenteista uusi työkalu koostuu ja miksi kyseiset komponentit valittiin osaksi toteutusta ja kuinka niitä on hyödynnetty. Lisäksi käsitellään erilaisia haasteita, kuten käyttöliittymän suorituskyky, joita kyseisessä projektissa kohdattiin ja kuinka ne ratkaistiin. Projektin lopputuloksena rakennettu käyttöliittymä on kuvattu. Työkalun kehityksessä ominaisuuksia toteutettaessa kiinnitettiin erityishuomiota siihen, että se on mahdollisimman helposti jälkikäteen laajennettavissa lisäominaisuuksilla.

3 Työkalut ja menetelmät

Seuraavassa kuvataan erilaisia teknologioita joita projektin toteutukseen käytettiin. Suuren osan teknologia valinnoista sanelivat sovellukselle toimeksiantajan määrittelemät rajapinnat, eikä niitä siksi voinut muuttaa.

Huomiota on kiinnitetty ensisijassa työkalun tai menetelmän soveltuvuuteen omaan projektiini ja sitä, että kyseiselle teknologialle on käyttäjiä ja dokumentaatiota olemassa, jotta sen käyttäminen on mahdollisimman sujuvaa.

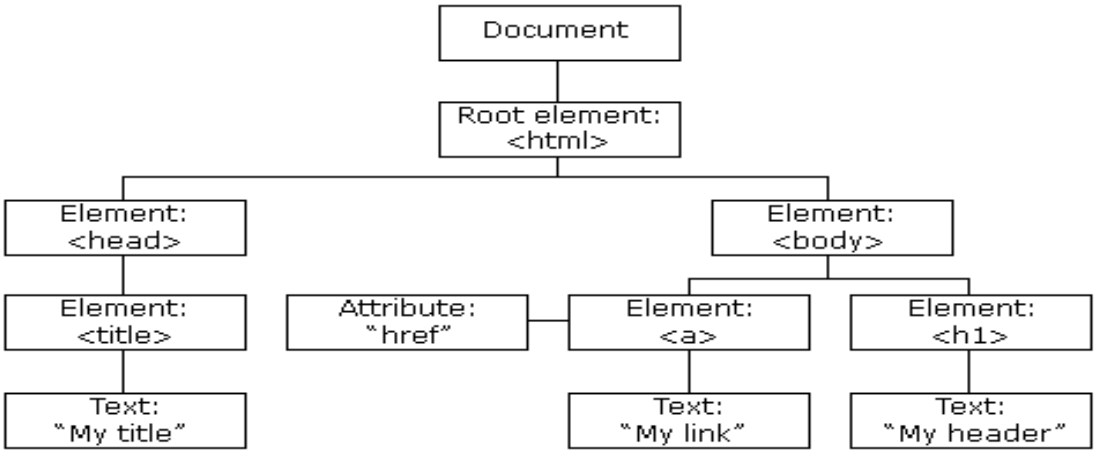
3.1 Node.js versionumero 0.12.0

Node.js on vapaan lähdekoodin ohjelmisto, jota käytetään JavaScript ohjelmien ajamiseen. Yksinkertaisen Node.js ohjelman teko on melko vaivatonta ja nopeaa, mutta kattavat ominaisuudet sisältävän ohjelman suunnitteluun ja kehittämiseen kuluu helposti runsaasti aikaa. Tässä toiminnallisessa opinnäytetyössä Node.js alustaa hyödynnetään ensisijaisesti HTTP/1.1-serverin suorittamiseen. Node.js oli mielestäni hyvin dokumentoitu ja sillä oli aktiivinen kehittäjäyhteisö esim. GitHub-portaalissa, joten tuen saaminen ja ohjeiden löytäminen oli helppoa. (Node.js Foundation 2015.)

3.2 HTML 5

HTML on World Wide Webissä käytetty kuvauskieli, joka kehitettiin alun perin tieteellisten dokumenttien esittämiseen. Tämän opinnäytetyön puitteissa HTML:ää hyödynnetään tiedonrakenteen esittämiseen käyttöliittymässä. HTML on vuosikymmenten kehitystyön tulos ja sitä on kehittänyt lukuisat eri toimijat tahoillaan eri paikoissa. Ominaisuuksia on sisällytetty siihen eri yhteisöiltä ympäri maailmaa, ja sen vuoksi kaikki ominaisuudet ja määrittelyt eivät välttämättä noudata yhtä johdonmukaista linjaa. HTML 5 on W3C:n kokoama HTML-kuvauskielestä julkaistu viimeisin spesifikaatio ja suositus. (Berjon ym. 2014.)

HTML on monipuolisesti laajennettavissa sovellusten käyttöön. Käyttäjät pystyvät luomaan omia elementtejään, luokka-attribuutin määrittelyillä. Kuvan 1. mukaisesti elementit järjestetään dokumentissa puurakenteeseen. HTML:llä voi lisätä metadataa tai ohjelmien hyödyntämää tietoa suoraan dokumentinrakenteeseen. Seassa voi olla kokonaisia ohjelmia, jotka pystyvät vuorovaikuttamaan dokumentin kanssa. (Berjon ym. 2014.)



Kuva 1: HTML-dokumentissa elementit on järjestettynä puurakenteeseen. (Berjon ym. 2014.)

3.3 DOM

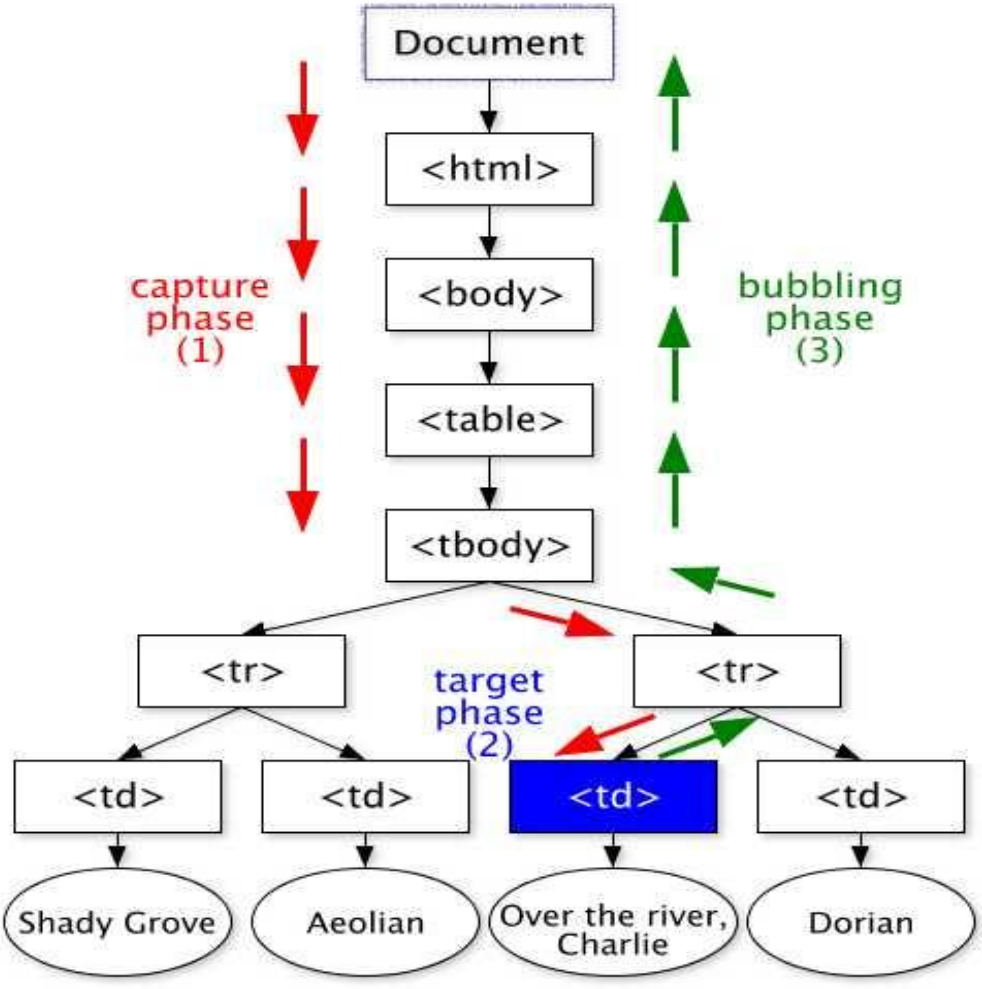
Verkkosivun latautuessa luodaan sivusta dokumenttiobjektimalli (Document Object Model). W3C standardi määrittää DOM:lle 3 erilaista mallia. Puhutaan joko Core DOM:sta, joka sisältää kaikki dokumenttityypit, XML DOM:sta joka määrittää standardimallin XML dokumenteille tai HTML DOM:sta kun kyse on HTML-dokumentista. (W3schools 2015.)

DOM voidaan nähdä objekteista koostuvana puurakenteena. HTML DOM on standardi tapa lukea, muuttaa lisätä ja poistaa elementtejä. HTML-dokumentin root-elementiksi eli juurielementiksi kutsutaan sitä elementtiä joka on dokumentti objektin ensimmäinen lapsi, eli sillä itsellään ei ole dokumentissa ylempää elementtiä. Muut elementit ovat tämän juuri-

elementin sisällä. HTML-dokumentissa on elementtien lisäksi tekstiä. (Kacmarcik & Leithead 2015.)

Kaikilla HTML elementeillä on aloitus -tagi, esim. <html> ja lopetus-tagi </html>. Kaikki HTML DOM:in sisältämät elementit ovat käsiteltävissä ohjelmallisesti JavaScriptillä ja muilla ohjelmointikielillä. HTML on media riippumaton esitystapa dokumenteille, ne voidaan esittää mm. piirtämällä ruudulle tai muuttamalla puhesyntetisaattorin kautta ääneksi. Kaikkien HTML-elementtien tulee toteuttaa HTML DOM:ssa niille määritellyt rajapinnat elementtien hyödyntämiseen. (Kacmarcik & Leithead 2015.)

DOM Events on rajapinta, joka on määritelty Ui Events standardissa. Sen avulla pystytään HTML-elementeille tapahtuvia asioita kuten näppäinpainalluksia seuraamaan ja kaappaamaan tapahtumankuuntelijaolioilla. Tapahtumankuuntelijat pystyvät reagoimaan tapahtumiin ja suorittamaan niille määriteltyjä funktioita. Kuten kuvasta 2. näkyy kuplivat tapahtumat koko puurakenteen läpi, ellei niiden kulkua estetä kesken matkan. (Kacmarcik & Leithead 2015.)



Kuva 2: Eventin lähetyksen kolme eri vaihetta (Kacarcik & Leithead 2015.)

3.4 JavaScript/ECMAScript

JavaScript-ohjelmointikieli sai alkunsa vuonna 1995. Se on yksi 1997 julkaistun ISO/IEC 16262 standardin toteuttavista ohjelmointikielistä. Vuonna 2011 julkaistu ISO/IEC 16262:2011 3rd edition, joka vastaa ECMA-262 ECMAScript versiota 5.1 on se versio standardista jota pyrittiin projektin kehitystyössä JavaScript-ohjelmoinnissa noudattamaan. (ECMA International 2011.)

ECMAScript on alun perin suunniteltu toimimaan selaimessa. JavaScriptillä pystyy kehittämään selaimessa toimivia JavaScript ohjelmia, jotka voivat vuorovaikuttaa käyttäjän ja tietokoneen muiden komponenttien kanssa. Selain tarjoaa asiakasympäristön, jossa voi JavaScriptiä suorittamalla piirtää käyttäjälle näkyviin ikkunoita, grafiikkaa, suorittaa laskutoimituksia tai ottaa vastaan käyttäjän komentoja. JavaScript koodi on HTML-verkkosivun sisällä HTML DOM-rakenteessa, tekstin, muiden elementtien sekä staattisen sisällön, kuten kuvat, seassa. JavaScript web-serverit kuten Node.js sisältää oman ajoympäristönsä jossa JavaScriptiä suoritetaan selaimen tavoin. JavaScript koodia siis voidaan hyödyntää sekä selaimessa (frontend) ja serverillä (backend), eli tämä osa web sovellus arkkitehtuurista voidaan hoitaa pelkästään JavaScript:llä, niin kuin tässä projektissa on tehtykin. JavaScript ei sisällä mitään varsinaista pääohjelmaa vaan se on sarja funktioita, joilla otetaan yhteyttä selaimeen rakennettuun käyttöliittymään. (ECMA International 2011.)

JavaScript määrittelee valmiiksi joukon olioita ja operaattoreita. JavaScript on oliopohjainen (object oriented paradigm) ohjelmointkieli, mutta se ei sisällä luokkia samassa merkityksessä kuin esim. C++. JavaScriptin syntaksissa on otettu suuria vaikutteita Java-ohjelmointikielestä. Taulukosta 1 ilmenee, että JavaScript kielen perusrakenteet ovat hyvin samanlaisia muiden objektiorientoituneiden kielten kanssa. JavaScript oliot koostuvat erilaisista arvoista, JavaScriptissä konstruktorifunktion kautta luodut oliot hyödyntävät automaattisesti prototyyppipohjaista perintää ja jaettuja arvoja. JavaScriptissä new kutsu johtaa implisiittisesti constructor funktion kutsuun. (ECMA International 2011.)

Olio, objekti	Kokoelma arvoja esim: 1, 2, 3 ja ”foo”
String	Merkkijono esim. ”abc”
Prototyyppi	Prototyypiltä periytyy luoduille objekteille jaettuja arvoja esim: 1, 2, 3
Funktio	Function konstruktorin luoma olio, joka voidaan suorittaa aliohjelmana
Metodi	Funktio olion arvona

Taulukko 1 Esittelee joukon fundamentaalisia JavaScript käsitteitä. (ECMA International 2011.)

3.5 CSS

CSS eli Cascading Style Sheets, eli suomeksi tyyliarkit, on joukko W3C:n suosituksia, jotka määrittelevät rajapinnan HTML-dokumentin sisältämien elementtien esittämiselle. Rajapinnan avulla pystyy mm. vaihtamaan elementtien ulkoasua, sekä skaalaamaan niitä erilaisiin mittasuhteisiin selaimessa. Niiden sisältämien määritysten avulla kehittäjä pystyy siirtämään paljon näyttämiseen liittyvää logiikkaa selaimen vastuulle. Tämä vähentää tarvetta hoitaa sitä JavaScript kielellä manuaalisesti, joka säästää ohjelmointityöhön kuluvaa aikaa. Ei esimerkiksi tarvitse tehdä erikseen ohjelmaa, jonka tulee muuttaa jonkin HTML-elementin sisällä olevien elementtien tietyn tyyppisen elementin ominaisuuksia sen attribuutin vaihtaessa arvoa. Sen pystyy tiivistämään paljon yksinkertaisempaan ja yksiselitteisempään muotoon CSS-määrittelykäskyllä. (Atkins Jr. & Etemad 2015.)

CSS määrittelykäskyt ovat aiemmin keskittyneet lähinnä esittämiseen ruudulla, mutta niiden avulla pystytään myös määrittelemään tulosteiden ulkoasua. Tätä tukea on parannettu tai ainakin pyritty siihen julkaisemalla tuoreita standardisuosituksia. Niitä hyödyntämällä tyylien määrittely hoituu elementtikohtaisesti ja näyttö määritykset noudattavat tiettyä presedenssiä. CSS-tiedostot ladataan käyttämällä siihen suunniteltua html määritystä, mutta määrityksiä on mahdollista kirjoittaa suoraan HTML-dokumentin rakenteeseen. Toisinkuin erillisillä HTML-dokumentin otsakkeeseen kirjoitetuilla määrityksillä tai erillisissä tiedostoissa ladatuilla määrityksillä, on HTML-dokumentin rakenteeseen kirjoitetuilla attribuuteilla niiden vaikuttavuusalue rajoitettu vain kyseisen attribuutin omistavaan elementtiin. (Atkins Jr. & Etemad 2015.)

Yksi määrittelyiden tärkeä sovellusalue on elementtien sijainnin määrittäminen suhteessa toisiinsa sekä tarvittaessa niiden koon mukauttaminen esim. käytössä olevan näyttöpäätteen resoluution perusteella. Elementtien siirtäminen toistensa taakse visuaalisesti on mahdollista. Määritysten merkityksiä on helppo muuttaa ja jo muutaman merkin muuttaminen CSS-määrittelyissä voi tuottaa tarvittavan vaikutuksen tuottamiseen käyttöliittymän esityksessä esimerkiksi vaihdettaessa määritys koskemaan yhden elementin sijasta kaikkia elementtejä vaikkapa fonttikoon osalta. CSS-määrittelyt olivat tehokkuutensa ja mielestäni määrätyn työpanoksen tuottaman ohjelmistotoiminnallisuuden korkean palautussuhteen ansiosta eräs tärkeimpiä työkaluja joilla vaikutin käyttöliittymän ulkoasuun. (Atkins Jr. & Etemad 2015.)

3.6 HTTP-protokolla

HTTP/1.1 on geneerinenrajapintaprotokolla keskenään viestiville tietojärjestelmille, jota World-Wide Web Information Initiative on käyttänyt vuodesta 1990 lähtien. HTTP/1.1 on pyyntö-vastaus mallia noudattava tilaton protokolla. Serveri lähettää vastauksen asiakkaan

lähettämään pyyntöön. HTTP/1.1 protokolla hyödyntää Uniform Resource Identifieriä (URI) pyyntöjen kohdentamiseen, uudelleenohjaustenosoituksiin sekä suhteiden määrittämiseen. (Fielding & Reschke 2014a.)

Se on kehittynyt paljon vuosien saatossa ja uusin versio HTTP 2.0 tuo mukanaan uusia ominaisuuksia web-ohjelmistojen kehittäjän avuksi, kuten HTTP push viestit, eli serveriltä käyttäjälle suuntautuvan viestinnän. Nykyselaimet eivät kuitenkaan vielä täysin tue kaikkia HTTP 2.0:n ominaisuuksia, joten päädyin käyttämään HTTP 1.1 protokollan mukaista viestintää tämän projektin puitteissa. HTTP 1.1 verrattuna HTTP 1.0 sisältää tiukemmat määrittelyt protokollan luotettavaan ja onnistuneeseen toteutukseen. Protokolla määrittelee, että asiakas (client) lähettää pyynnön serverille, joka sisältää pyynnön metodin, eli http verbin, URI:n sekä protokolla version, pyyntöön liittyvät otsakkeet sekä useimmissa tapauksissa myös varsinaisen viestin rungon (body). Useimmiten HTTP pyynnöt kohdistuvat johonkin palvelimella olevaan resurssiin. Yleensä HTTP viestintään käytetään TCP/IP-protokollaa ja porttia TCP 80, mutta viestintä voidaan toteuttaa muita portteja käyttämällä. Esimerkiksi, jos viestintä portin 80 läpi halutaan estää, tai se ei jostain muusta syystä ole mahdollista. (Fielding & Reschke 2014a.)

HTTP-pyyntöihin tulleet vastaukset sisältävät numerokoodin, josta pystyy päättelemään onnistuiko pyynnön suoritus, vai keskeytyikö sen suoritus jostain syystä eikä pyyntöä pystytty suorittamaan onnistuneesti loppuun. Em. numerokoodit voidaan jakaa viiteen kategoriaan. Numerolla 1 alkavat statuskoodit indikoivat informatiivista pyyntöä, esimerkiksi protokollan vaihtoa johonkin muuhun. Numerolla 2 alkavat koodit taas on suoritettu onnistuneesti. Numerolla 3 alkavat taas on uudelleenohjattu johonkin toiseen URI:iin, kuin mihin pyyntö alun alkaen kohdistui. Numerolla 4 alkavat taas kertovat, että pyyntöä ei voitu täyttää sen takia, että asiakasohjelma on tehnyt virheen, pyyntö on syystä tai toisesta väärin muodostettu. Se voi esimerkiksi sisältää väärän osoitteen, jonka kohdistamaa resurssia ei palvelimelta löydy. Numerolla 5 alkavat taas kertovat, että serverillä on tapahtunut virhe. (Fielding & Reschke 2014b.)

3.7 CoffeeScript

JavaScript oli selvä valinta käyttöliittymän pohjaamaksi ohjelmointikieleksi, sillä se on hyvin tuettu kaikissa suosituimmissa selaimissa. JavaScriptin kirjoittaminen on mielestäni kuitenkin helpompaa hyödyntämällä siinä CoffeeScript nimistä ohjelmointikieltä. CoffeeScript on yksi avain teknologioista, joita hyödynsin käyttöliittymän toteutuksessa. Se oli minulle tuttu jo muista yhteyksistä, koska koin sen hyödyllisenä apuvälineenä koodinkirjoituksessa päätin käyttää sitä tässä projektissa. CoffeeScript on ohjelmointikieli, joka muutetaan JavaScript

koodiksi Ubuntu-käyttöjärjestelmässä käyttämällä erillistä coffee nimistä apuohjelmaa. (CoffeeScript documentation 2015.)

CoffeeScript muistuttaa mielestäni syntaksiltaan paljon Ruby-ohjelmointikieltä. Siksi se toimii hyvin omassa tapauksessani kun Ruby on tällä hetkellä eniten ohjelmoimani kieli. CoffeeScript tarjoaa syntaktistasokeria perinteiseen JavaScript ohjelmointiin. Se on tiiviimpää koodia, sisältäen vähemmän merkkejä, ja on siksi helpompi painaa työmuistiin kun ohjelmaa kehittää. Mielestäni se osaltaan helpotti ohjelmointityötä vähentämällä ohjelmointivirheitä, ja vähentämällä aaltosulkeiden sekä puolipisteen kirjoituksen tarvetta ohjelmakoodissa. CoffeeScriptiä käyttäessä täytyy muistaa, että CoffeeScript on JavaScriptiä joten useimmat JavaScriptin ominaisuuksista on hyödynnettävissä. Tunsin CoffeeScriptin ja JavaScriptin perusteet jo ennestään, joten niidenkään opetteluun ei erityisesti kuluisi projektissa aikaa, mikä osaltaan puolsi niiden hyödyntämistä yhtenä projektin työkaluista kehityksen tehostamisessa. Huonona puolena on mm. se, että koska sisennykset ovat merkitseviä CoffeeScriptissä voi koodin seasta olla vaikeaa huomata missä on tehnyt virheitä. Huomasin, että tämä ongelma pieneni melko nopeasti kun kieltä kirjoitti enemmän, ja opin paikallistamaan ongelmakohtia koodista tarkemmin. (CoffeeScript documentation 2015.)

3.8 YAML/YML

YAML eli YML Ain't Markup Language on ihmisen luettavaksi tarkoitettu datan serialisointiin tarkoitettu kieli. Sen suunnittelussa on käytetty seuraavaa seitsemää periaatetta tärkeysjärjestyksessä.

1. Se on helposti ihmisen luettavissa
2. Sen kuvaama data on helposti siirrettävissä ohjelmointikielestä toiseen
3. Vastaa ketterien kielien natiiveja tietorakenteita
4. Tarjoaa johdonmukaisen tavan geneeristen työkalujen hyödyntämiselle
5. Tukee yhden pyyhkäisyn prosessointia
6. Ilmaisuvoimainen ja laajennettava
7. Helppo toteuttaa ja käyttää

YML on epävirallisesti aina tukenut JSON:n muotoisen datan säilömistä YML:nä, mutta version 1.2 myötä tuki on virallistettu, ja voidaan katsoa JSON formaatin olevan sisällytetty YML formaattiin. Kaikki JSON tiedostot ovat siis kelvollisia ja spesifikaation määritelmät täyttäviä YML tiedostoja. (döt Net, Evan & Oren 2009.)

4 Kehitysmenetelmät

Projektissa käytin sekoitusta erilaisista vakiintuneista ohjelmistokehitysmalleista kuten Agile-mallista ja Lean-mallista, mutta ne eivät mielestäni mikään edusta tehokkainta toimintatapaa, joten päädyin yhdistelmään johon valikoin jokaisesta mallista niiden parhaita käytäntöjä ja tapoja toimia. Päädyin tähän pääasiassa sen takia, että täytyi ottaa huomioon, kuinka useat ohjelmistokehitysmallit on suunniteltu suurempia kehitystiimejä ja organisaatioita silmälläpitäen.

Omassa mallinnustyökalun kehitysprojektissani ainoa kehitysresurssi olin minä itse joten esimerkiksi mitään päivittäisiä tiiminsisäisiä palavereita projektinedistymistä tai vastuualueita koskien ei tarvittu. Toki monet kollegoistani edesauttoivat projektia, ja olivat esimerkiksi määrittelyn saralla aivan keskeisiä, mutta mihinkään päivittäiseen juoksevaan asiaan he harvemmin puuttuivat. Kollegani ja yritysorganisaatio tarjosivat tukea koskien kehitystyön suuria linjoja ja ohjeistivat monissa asioissa, mutta en törmännyt mihinkään pikkutarkkaan työni hallintaan. Otin mielelläni paljon vastuuta projektista.

Oli mukavaa nauttia luottamusta, joka mahdollisti jopa muutaman viikon sprinttien eli intensiivisten kehitysjaksojen toteuttamisen ilman, että ratkaisujani kyseenalaistettiin jatkuvasti. Mielestäni oli projektin tehokkaan toteuttamisen kannalta avaintekijöitä, että minuun uskallettiin luottaa, eikä tekemiseeni liiaksi puututtu. Se salli kokonaisarkkitehtuurin pysymisen selkeämpänä, kun arkkitehtuuria ei tarvinnut muuttaa oleellisesti kesken projektin. Silloin sain itseluottamusta hioa itse ohjelmaa paremmaksi sieltä missä sille näin tarvetta, eikä aikaa kulunut byrokratiaan vaan kehitystyötä oli projektin toteutukseen kuluneesta ajasta yli 90 %.

Ohjelmointivirheitä pyrittiin välttämään kiinnittämällä huomiota siihen miten ja missä järjestyksessä koodia suoritettiin.

Projektin edistymistäni seurattiin päivätasolla, mutta sen osalta riitti, kun kerroin lyhyesti viime päivän aikana toteutetut ominaisuudet tai kehitystyön kohteena olevan asian jos se ei työpäivässä (7,5h) ollut ehtinyt valmistua. Mihinkään pikkutarkkaan yksityiskohtien läpikäyntiin ei käytetty aikaa.

Varsinkin projektin alkuvaiheessa arkkitehtuuri haki vielä itseään. Silloin minun ei tarvinnut sen kummemmin perustella esimerkiksi ajoittain kehitystyöhön integroituna tekemiäni lähdekoodin refaktorointeja, eli koodinkannan paikallista uudelleen kirjoitusta, sillä minuun ja arviointikykyyni luotettiin. Usein käytännön syyt, kuten jonkin ominaisuuden kehitykseen kuluva

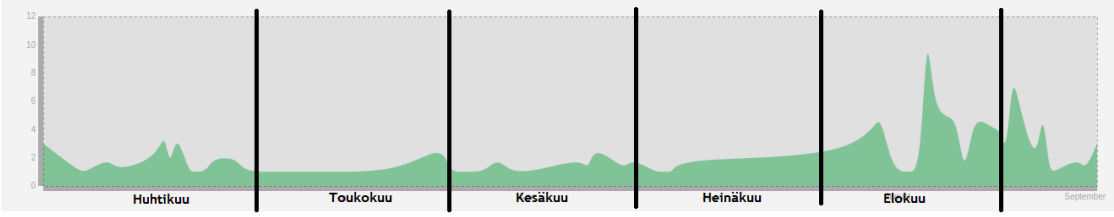
aika, saneli ehdot sille kuinka paljon siihen ohjelman kyseisessä kehitysiteraatiovaiheessa käytettiin aikaa.

5 Versionhallinta

Versionhallinta projektissa sen koko kehityskaaren ajan toteutettiin hyödyntämällä yhdistelmää Git ja GitLab vapaanlähdekoodinohjelmistojen suorituskykyjä koodin hallintaan, koodin seurantaan ja code revieweihin, eli toimeksiantajayrityksen koodin sisäisiin läpikäynteihin ja auditointeihin. Sen kautta saatiin talteen ja näkyville tärkeätä tietoa ohjelmaa koskevasta kehityshistoriasta, joka on luettavissa Git:istä esimerkiksi mikäli joku kolmas taho joskus tulevaisuudessa mahdollisesti jatkokehittää projektia.

Projektissa tehtiin 165 commit-tapahtumaa, joiden sijoittuminen elinkaaren aikana näkyy kuvassa 3. Projektin commit-tapahtumat tapahtuivat pääasiassa päiväsaikaan painottuen iltapäivään, tarkemmin kuvassa 4.

Ohjelmisto sisälsi riippuvuuksineen kokonaisuudessaan noin 350 tuhatta koodiriviä, joista versiohallittavia itse tuotettuja, eli tiukasti tekijänoikeuden suojaamia koodirivejä on noin 20 tuhatta.



Kuva 3: Aikajanalla näkyy 165 commit-tapahtumaa projektin kehitys elinkaaren ajalta.

Päivä	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Ma	1									1	1	1	2	1	2	3	3	9						
Ti										1	2	3		5	2		4	8	1					
Ke		1									4	1	1		2	6	5	5	1					
To		1									1	3		11	6	4	4	10			2	2		
Pe		1		1							3	8	1	5	3	2	6	10	1					
La		1	1													1								1
Su																								

Kuva 4: Kuvassa on projektin aikana versionhallintajärjestelmään tehtyjen commit-tapahtumien määrät vuorokauden ajan sekä viikonpäivän mukaan. Useat tapahtumat sijoittu-

vat kello 16.00 - 18.00 väliselle ajanjaksolle ja aamupäivällä tehtyjä commit-tapahtumia ei määrällisesti ole niin paljon kuin iltapäivällä tehtyjä.

6 Suunnittelu

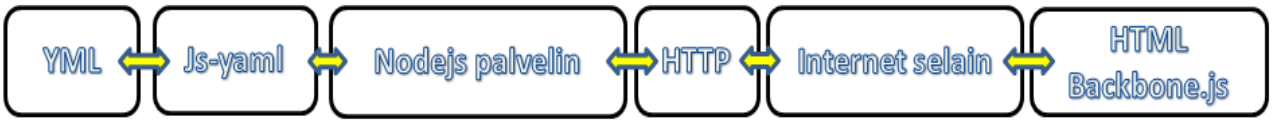
Ohjelman suunnitteluun käytettiin prototyyppimenetelmää, jossa nopeassa syklissä eritellään spesifinen ominaisuus, toteutetaan se ohjelmallisesti ja testataan sen toimivuus käyttöliittymän kautta selaimessa. Esimerkiksi jonkin tiedon tallennus on helppo tällä syklillä iteroimalla ratkaistavissa oleva ongelma. Siinä suunnittelun tukena ei käytetty tarkkoja määrittelyjä, vaan abstrakteja kokoelmia ja näkymäolioita hahmottamaan eri sisäkkäisten olioiden sekä niiden attribuuttien komponenttien toiminnallisuus kokonaisuuksia, ja niiden välisiä suhteita toisiinsa.

Pohdittiin mitä vuorovaikutussuhteita eri komponenttien välille muodostuu ja sen perusteella päätettiin miten käyttöliittymä jaetaan alinäkymäolioihin ja miten tietorakenteet kannattaisi koostaa sekä kytkeä näkymiin. Yritettiin löytää semmoisia ratkaisuja, jotka mahdollistivat ominaisuuksien toteutuksen mahdollisimman vähällä koodin määrällä, jotta toteutus pysyisi helposti muunneltavana ominaisuuksia lisättäessä.

Tehtiin tutkimustyötä eri ohjelmistokirjastoista, joiden sisältämät rajapinnat mahdollistivat mahdollisimman suoraviivaisen ja käytettävän toteutuksen eri ominaisuuksille lyhyessä ajassa. Pyrittiin pitämään toteutukseen valittavien ohjelmistokirjastojen laatu korkeana ja määrä alhaisena.

7 Työkalun arkkitehtuuri

Projektissa käytettiin käyttöliittymän toimintalogiikan runkona Backbone.js-JavaScript kirjastoa. Backbone.js valittiin siksi, että se on toimeksiantajalla käytössä jo muissakin kohteissa ja siitä on itselläni myönteisiä kokemuksia. Projektissa käytettiin monia eri teknologioita, jotka viestivät keskenään kuten kuvasta 5. ilmenee.



Kuva 5 Kuvassa näkyy mallinnustyökalun arkkitehtuuri projektin ensimmäisessä kehitysvaiheessa. Keltaiset nuolet osoittavat miten kaikki komponentit ovat yhteydessä toisiinsa.

Oli mielestäni mm. aikataulussa pysymisen edellytys valita työkaluiksi ainakin osittain semmoisia teknologioita, joiden toimintalogiikka oli minulle jo ennestään ainakin jonkun verran

tuttu. Pystyin jo kokemuksesta sanomaan, että Backbone.js soveltuu käyttöliittymän rungoksi hyvin, sen päälle pystyisin rakentamaan mahdollisesti tarvittavat muut suunnittelumallit ja logiikat, jotka ovat jokaisessa projektissa omanlaisensa.

Backbone.js on selaimessa suoritettava JavaScript-kirjasto, joka tarjoaa useimpien webapplikaatioiden tarvitsemat primitiiviset palvelut, kuten tiedon säilönnän malleihin (Model) ja kokoelmiin (Collection). Lisäksi se tarjoaa valmiin näkymä luokan, jolta löytyy paljon JavaScript-funktioita, jotka auttavat asynkronisesti toimivan käyttöliittymän rakentamisessa ja tiedon esittämisessä selaimessa. (Backbone.js documentation 2015.)

Itse verkkoasun ulkoasu kuvataan HTML-kuvauskielellä, mutta sen avulla ei ole mahdollista rakentaa kuin hyvin alkeellista logiikkaa. Jokainen ei triviaali selaimessa toimiva sivusto hyödyntää JavaScriptiä tai muuta ohjelmointikieltä varsinaisena moottorina – tai aivoina, joka sivustoa pyörittää. Jo pelkästään asynkroninen viestiminen palvelimen ja selaimen välillä edellyttää JavaScriptin käyttöä. HTML-kuvauskielellä määritellään vain tiedonrakenne käyttöliittymässä. (Berjon ym. 2014)

Palvelimen kanssa viestimiseen käytetään JSON-muotoisia HTTP-viestejä, JSON on käytännössä tapa serialisoida JavaScript olioita tekstimuotoon. Objekteja eli olioiden tiloja voidaan siirtää tekstimuotoon ja palauttaa ne täsmälleen samanlaisina koneen ymmärtämään muotoon toisessa päässä. JSON-lyhenne tuleekin sanoista JavaScript Object Notation.

Node.js-palvelimella otetaan JSON muotoisia viestejä vastaan, HTTP-viestin tyyppi määrittää halutaanko palvelimelta pyytää resurssin (GET-pyynnöt), lähettää palvelimelle resurssin tallennettavaksi (POST-pyynnöt) tai poistaa resurssin (DELETE-pyynnöt). Lisäksi on olemassa muunkinlaisia HTTP-viestejä, mutta niitä ei hyödynnetty tässä projektissa. Kun palvelin vastaanottaa GET-viestin, niin palvelin lukee js-yaml nimistä kirjastoa käyttäen tietokoneen tiedostojärjestelmässä sijaitsevassa kansiossa olevat YML-tiedostot, joko kaikki tai URI:n määrittelemän yksittäisen YML-tiedoston. Node.js palvelin hyödyntää Unix järjestelmistä löytyviä systeemikutsuja kuten spawn, stat ja mkdir käyttääkseen koneen muita resursseja kuten käyttämään tiedostojärjestelmää tai suorittamaan ohjelman kannalta oleellisia aliprosesseja. (Node.js documentation 2015.)

YML-formaatti oli valittu mallinnukseen mm. sen yleiskäyttöisyyden ja hyvän tuen ansiosta, lisäksi YML-formaatissa sijaitsevat mallit ovat helposti versionhallinnan hyödynnettävissä. Jotta YML-tiedostot ovat helposti ja turvallisesti muiden ohjelmien käytettävissä, on niiden käsittely hoidettu palvelin päässä käyttämällä tiedostojärjestelmään päin synkronisesti toimivia JavaScript-funktioita, koska JavaScript ei selaimessa suoraan mahdollista koneen tiedostojär-

jestelmän käsittelyä on Node.js palvelinohjelmaan toteutettu paljon funktioita C++-laajennuksina. (döt Net ym. 2009.)

8 Mallinnustyökalun käyttöliittymän semanttinen rakenne ja suorituskyvylliset ominaisuudet

Tiesin, että käyttöliittymä tulisi yrittää pitää kevyenä, jotta suuren tietomassan käsittely olisi mahdollista mahdollisimman lähelle reaaliajassa. Jouduin pitkin toteutusvaihetta kiinnittämään suorituskyvyllisestä näkökulmasta huomiota siihen, että tietokone teki mahdollisimman paljon sitä mitä piti, eli tiedon esittämistä ja tallentamista. Pyrin pitämään mm. näkymä olioiden lukumäärät pienenä, jotta niiden luomiseen käytettävä aika pysyisi mahdollisimman pienenä. Kaikki loogiset ja aritmeettiset operaatiot pyrittiin pitämään riittävän tehokkaina kohdistamalla ne vain siihen osajoukkoon kuin kulloinkin oli tarpeellista. Täytyi kiinnittää paljon huomiota siihen miten usein iteroi kokoelmia läpi, sillä jokaisella kerralla kun mennään aliobjektin sisään ja käydään läpi sen sisältämät arvot on jo mennyt monia suoritus käskyjä ja ne kertaantuvat aina suurempina mitä laajempaa objektijoukkoa kulloinkin käsitellään.

Jos suorituskykyyn ei kiinnitä huomiota on suuria ongelmia toteuttaa monimutkaisia käyttöliittymiä, sillä vasteajat käyvät sietämättömiksi. Ei silti saa alkaa optimoimaan ohjelmia liian aikaisessa vaiheessa. Pullonkaulat kannattaa korjata suurimmasta pienimpään, vasta projektin siinä vaiheessa kun suorituskyky putoaa niin huonoksi että se haittaa kehitystyötä. Tai kun saadaan julkaisukandidaatti valmiiksi jolloin uusia ominaisuuksia ei enää toteuteta käyttöliittymään vaan se siirretään hyväksyttäväksi toimeksiantajalle. Hyödyntämällä underscore.js ja jQuery kirjastojen sisältämiä iteraattorifunktioita pyrittiin pitämään käyttöliittymän kokoelmien iteroinnit tehokkaina ja ohjelmointivirheettömämpinä. (jQuery API each() function 2015.)

Tiedonhakuun ja järjestykseen sekä näiden suhteisiin toisiinsa kiinnitettiin tiedonhallinnollinen ja suorituskykynäkökulma huomioiden paljon huomiota. Knuth kiinnittää huomiota siihen, että on paljon tehokkaampaa hakea järjestetystä joukosta binäärihakua käyttäen haettua tekstiä kuin satunnaisjärjestyksessä olevasta käyttäen jotain tehottomampaa algoritmia. (Knuth 2014, 409.)

Binäärihakuakin parempi on käyttää taulukosta hakuun interpolaatiohakua. Sekään ei kuitenkaan kaikissa tapauksissa kompensoi kasvavaa kuormaa suoritukselle paitsi jos haettava taulukko on todella iso. (Knuth 2014, 420.)

Backbone mahdollisti DOM:n käsittelyn helposti elementti kerrallaan, jolloin ei tarvitse piirtää koko dokumenttia uusiksi kerralla, vaan voidaan päivittää ne osat joita muutokset koskevat.

Yhtä objektia sadasta muutettaessa arvoja ei tarvitse parhaimmillaan päivittää käyttöliittymään kuin sen yhden objektin osalta sadan sijaan, joka on suorituskyvyllisesti tietenkin erittäin edullista verrattuna toiseen vaihtoehtoon jossa kaikki piirrettäisiin.

Esimerkiksi kyseessä olevassa käyttöliittymässä oli tarpeellista näyttää jopa satoja rinnakkaisia elementtejä, joista jokainen oli oma Backbone.View näkymänsä. Nämä näkymät pitivät sisällään monia alinäkymiä ja nekin saattoivat tahollaan pitää sisällään monia alinäkymiä jne. Tämä aiheutti monesti ongelmia ohjelman suorituskykyyn pitkin kehityskaarta, ja ne piti aina yksi kerrallaan pistää kuntoon. Muuten käyttöliittymä saattoi jämähtää täysin mikä taas hidasti internetselaimessa tehtyä ohjelmakoodin ja käyttöliittymän testausta.

Kaikki tiedon tallennus pyrittiin tekemään niin että tallennettu informaatio oli mahdollisimman hyvälaatuista (Shannon & Weaver 1964, 8-16).

9 Käyttöliittymän rakenne

Käyttöliittymää lähdetään rakentamaan selaimesta käytettäväksi. Apuna käytetään mm. Twitter Bootstrap-webkirjastoa versio 3, josta saadaan mm. valmiita CSS tyyliteltyjä JavaScript-toiminnallisuutta sisältäviä komponentteja, jotka on rakennettu tavallisten HTML-elementtien pohjalta. Bootstrap auttaa käyttöliittymän ulkoasun rakentamisessa, tarjoamalla grid-järjestelmän, jolla verkkosivun ulkoasu jaetaan riveihin ja niiden sisältämiin sarakkeisiin, jotka skaalantuvat CSS luokkien perusteella automaattisesti sen mukaan kuinka suuri resoluutioiselta näytöltä käyttöliittymää käytetään. (Twitter 2015.)

Bootstrap komponentit, niiden käyttäytyminen ja ulkoasu, määritellään pääasiassa asettamalla Jade-template pohjissa määritellyille HTML-elementeille Bootstrapin dokumentaatiossa ohjeistettuja luokkanimiä. Niitä ovat esimerkiksi ”row” ja ”form-control”. Bootstrap kirjaston hyödyntämiseen löytyy esimerkkejä verkosta mm. Bootstrap-kirjaston kotisivuilta, joita hyödynsin paljon. (Berjon ym. 2014; Twitter 2015.)

Käyttöliittymän rakenteen suunnittelu tuli minulle yrityksen tuotekehityksestä valmiina, ja kaaviokuvista hahmottuivat miten valikot ja itse päänäkymä sijoittuvat suhteessa toisiinsa. Minun täytyi toteuttaa logiikka, jolla pystyn esittämään nuo kaaviokuvat, sekä kaiken niissä piilevän toiminnallisuuden kuten liukuvalikot ja muut tietojen asetuskentät validointeineen HTML-elementteinä internetselaimessa.

Harkitsin HTML5 standardissa julkaistun canvas HTML-elementin käyttöä pohjana luokkien ja attribuuttien mallinnustoiminnallisuuden toteuttamiseen. Totesin sen kuitenkin liian suureksi riskiksi, koska en ollut varma soveltuisiko se tähän projektiin. Minulla ei ollut siitä lainkaan

kokemusta ja se olisi aiheuttanut siitä syystä varsinkin aikataulullisia riskejä, koska olisin joutunut ensiksi selvittämään soveltuisiko se tämänkaltaisen työkalun toteuttamiseen.

Lomakepohjaisen mallintamisen periaatteet taas olivat itselleni tuossa kohtaa paljon paremmin hallussa, ja tiesin että pystyisin sitä hyödyntämällä toteuttamaan ohjelman. Tiesin kuitenkin, että canvas-teknologia pystyi mm. graafiseen mallintamiseen paljon luontevammin kuin lomakepohjainen käyttöliittymä. Huomasin, että jo tästä johtuen olisi syytä yrittää pitää käyttöliittymän esityskerros erillään mallinnuslogiikasta niin paljon kuin aikataulut antaisivat myöten, jotta esim. canvas elementtiin pohjautuvan käyttötyökalukerroksen liittäminen onnistuisi tämän projektin puitteissa toteutetun primitiivisen käyttöliittymän paikalle. (Berjon ym. 2014.)

Backbone.View:stä periytetyt näkymäoliot pitävät sisällään rajapinnan, jolla mallinnusta käytetään. Backbone.View-pohjaiset oliot on pyritty pitämään mahdollisimman rajapinnaltaan geneerisinä jolloin ne pitäisi pystyä tehokkaammin muuttamaan reagoimaan muidenkin kuin tämän projektin puitteissa toteutetun esityskerroksen HTML-elementtien muutoksiin. Tajusin, että minun olisi kuitenkin mahdotonta säilyttää mitään universaalia yhteensopivuutta tällaisen vasta spekulatiivisen mahdollisuuden kanssa, vaan oli tyydyttävä niihin työkaluihin mitä kyseisellä hetkellä on käytössä ja hylättävä liika tulevaisuuden miettiminen ja spekuloiminen, tämä liittyen aiemmin mainittuun HTML5 canvas-teknologiaan.

Oli hyvin riskialtista lähteä monimutkaistamaan käyttöliittymää aikaisessa vaiheessa projektia vaan oli pyrittävä toteuttamaan ominaisuudet vain mahdollisimman kevyesti. Silloin niiden korvaaminen jälkeenpäin jollain paremmalla olisi mahdollista kohtuullisella vaivalla, tästä ratkaisusta ei mielestäni aiheutunut ylimääräistä kehitystä vaikeuttavaa ja hidastavaa kuormaa.

Mikäli haluaisi pitää käyttöliittymän logiikan ja esityspinnat täysin erillään ja toisistaan tietämättöminä, vaatisi se toimestani paljon suurempaa työmäärää kuin tässä produktissa toteuttamani tunnit. Ajatuksena oli pyrkiä toteuttamaan käyttöliittymä niin kevyesti kuin mahdollista jolloin teoriani mukaan sen vaihtamiseen kuluva aika on pienempi kuin, jos käyttöliittymään liittyvää ohjelmakoodia on paljon. Kevyenä pitäminen oli hyödyllistä iteroivassa kehityksessä kun jonkin virheen huomaamisen aiheuttamat korjaustyöt järjestelmään olivat mahdollisimman pienet tuntimäärällisesti.

Hyödynsin paljon CSS3 ominaisuuksia käyttöliittymässä, niiden avulla sain siirrettyä vähällä vaivalla paljon HTML-elementtien esittämiseen liittyvää logiikkaa selaimen vastuulle, yksinkertaisia CSS-määrittelyjä hyödyntämällä.

HTML-elementtien tilaa suhteessa muihin pystyttiin vaihtamaan kätevästi JavaScriptillä, näkymiä voitiin piirtää käyttäjän huomaamatta jo valmiiksi, jolloin vasteajat pysyivät mahdollisimman pieninä, render() funktiokutsut pystyttiin kohdistamaan vain joukkoon näkymä- tai malliobjekteja kerralla.

Käyttöliittymä on selain näkymässä pääosin tietosisältöä pl. vasen ja ylävalikko, kuten kuva 6. osoittaa.

Kuva 6: Mallinnustyökalun käyttöliittymä.

10 Käyttöliittymän toiminta

Käyttöliittymän pohjana on HTML-dokumentti, jonka tyypiksi on asetettu HTML5 standardin mukainen html. Kyseinen dokumentti pitää sisällään viittaukset muihin ohjelman tarvitsemiin tiedostoihin, eli CSS- ja JavaScript-tiedostoihin. Dokumentin latauduttua selaimeen tapahtuu valmis-tapahtuma. Sitä kuuntelemaan on asetettu EventHandler, eli tapahtumiin reagoiva JavaScript-olio. Se sieppaa valmis-tapahtuman ja aloittaa JavaScript koodin suorituksen selaimessa. Suoritus alkaa siitä, että alustetaan ohjelman tietosisältöä eli dataa hallitsevat Backbone.Collections- ja Backbone.Models-oliot. Pääkokoelmaa, eli tietomalleja hallitseva Backbone.Collection-olio lähettää GET-pyynnön serverille, joka vastaanottaa pyynnön sekä päättelee siitä osoitteesta mihin pyyntö on lähetetty, että mitä palvelimen odotetaan pyyntöön vastaavan. (Backbone.js documentation 2015.)

REST-ohjelmistosuunnittelumallin mukaan, kun GET-pyyntö lähetetään collection-reittiin, eli esimerkiksi osoitteeseen tiedot/. Ilman id-tarkenninta olevaa GET-pyyntöä käytettäessä halutaan hakea kaikki tiedot kokoelman sisältämät resurssit. Mallinnustyökalun tapauksessa ne ovat luokkia, ja kaikki asetetaan periytetyn Backbone.Collections-objektin funktioiden avulla kokoelman attribuuteiksi, eli muodostuu yhdestä moneen suhde kokoelman ja luokkien välille. Sen jälkeen, kun kokoelma on sijoittanut kaikki lataamansa resurssit yksitellen kokoelman sisään, jatkuu ohjelman suoritus alustamalla näkymä eli Backbone.View-objektista periytetyt objektit, jotka suorittavat varsinaisen HTML-sisällön selaimen näytölle piirtämisen. Näkymät on kukin sidottu ainakin yhteen HTML-elementtiin, vakiotapauksessa näkymän elementti on tallennettu näkymän $el nimiseen arvoon talteen myöhempää käyttöä varten. (Backbone.js documentation 2015; Rodriguez 2015.)

Päätason näkymiä on toistaiseksi toteutettu mallinnustyökaluun kolme. Ne ovat ylävalikkonäkymä, vasensivuvalikkopalkkinäkymä sekä päänäkymä, joka pitää huolen varsinaisten tietomallien näyttämisestä, ylävalikkonäkymä ja vasensivuvalikkopalkkinäkymä ovat päänäkymää tukevia näkymiä, ja sisältävät pääasiassa luokkien muokkausta helpottavaa toiminnallisuutta, kuten tallenna ja avaa napit. Kaikki ladatut tietomallit ovat vasemmassa sivuvalikkopalkissa, josta niitä voi helposti selata klikkaamalla, jolloin päänäkymä vierittää itsensä klikatun tietomallin kohdalle. Kaikki muut Backbone.View-objektista periytetyt näkymäoliot ovat jonkin näiden kolmen päätason näkymän alinäkymiä, niiden suoritus alkaa vasta kun päänäkymää aletaan piirtämään.

Jokaisella päänäkymällä on oma elementtinsä jonka sisään se piirtää itsensä sekä mahdolliset alinäkymänsä. Samalla kun näkymiä piirretään, asetetaan jokaisen näkymä olion sisältämän events nimisen arvon määrittelemät tapahtumankuuntelijat, jotka kuuntelevat joko tietyn tyyppisiä tai kaikkia tapahtumia mitä ohjelmassa tapahtuu. Kun tapahtuma aktivoidaan esimerkiksi painamalla jotain nappia käyttöliittymässä, lähtee tapahtuma kuplimaan (bubble) ylös dokumentin puurakennetta, jos jokin elementti tämän ylöspäin kuplimisreitin varrella on asetettu kuuntelemaan ko. tapahtumatyyppiä, siirtyy koodin suoritus näkymän events-tietorakenteessa määriteltyyn funktioon, funktion sisällä on mahdollista katkaista tapahtuman ylöspäin kupliminen siitä elementistä alkaen, jos sille on tarvetta. Kun jokainen päätason näkymä alinäkymineen on piirretty, on käyttöliittymä kokonaisuudessaan piirretty selainikkunaan ja se on toimintavalmis. (Backbone.js documentation 2015; Kacmarcik & Leithead 2015.)

11 Haasteita projektin toteutuksessa

Projektin toteutuksessa ratkaistiin mm. seuraavia kysymyksiä:

- Miten voidaan varmistaa käyttöliittymän laajennettavuus jälkikäteen mahdollisimman pienellä työllä ja niin, että projektiin on mahdollista kouluttaa uusi työntekijä jälkikäteen?
- Täytyy käyttää hyvin dokumentoituja teknologioita, jotka voidaan olettaa useimpien kehittäjien hallitsevan.
- Miten löytää oikeat työkalut, jotka tarjoavat kompromissin kustannusten ja ajankäytön kanssa?
- Täytyy seurata tiiviisti sosiaalista mediaa ja muita kehittäjien käyttämiä viestintäkanavia ja lukea muiden kokemuksia eri kehitysteknologioista ja niiden toimivuudesta eri konteksteissa.
- Käytössä on vapaan lähdekoodin ohjelmat, miten voi varmistaa, että niitä tuetaan tulevaisuudessa?
- Käytetään valinnassa apuna tutkimusta millä työkaluilla on aktiivinen ja lukumääräinen kehittäjäkunta, mielestäni ohjelmalla on silloin suurempi mahdollisuus tarjota tukea tulevaisuudessa päivityksin.

12 Lopputuloksen arviointi

Lopputuloksena saavutettiin se mitä lähdettiin hakemaan, eli ArgoUML onnistuttiin korvaamaan toimeksiantona toteutetulla työkalulla. Ensimmäiset asiakasprojektit on jo siirretty järjestelmän piiriin, ja mallinnetaan uudella työkalulla. Työkalua on kiitelty mm. ulkoasusta ja käytettävyydestä, mutta monia ohjelmointivirheitä on havaittu lopputuloksessa, kaikki eivät kuitenkaan ole suoranaisia virheitä vaan subjektiivisia näkemyksiä siitä miten tietyt ominaisuudet tulisi olla toteutettuna. Erityiskiitosta sai HTTP-teknologian hyödyntäminen, joka on mahdollistanut työkalun helpon etäkäytettävyyden esim. etätöissä.

Työkalu on ollut myös toimeksiantajan ruotsalaisen yhteistyökumppaniyrityksen käytössä Ruotsissa, jossa sillä on tehty tiedonmallintamista. Työkalun jatkokehittäminen on aloitettu, vaiheessa 2. toteutetaan mm. prosessienkäsittelyn mahdollistava toiminnallisuus työkaluun lisäominaisuutena. Projektin kehitystyön aikana oma web-osaamiseni on kasvanut ja monipuolistunut. Alla on listattuna järjestelmän merkittävimmät ominaisuudet, jotka saatiin projektissa toteutettua.

- Toimii Unixiin pohjautuvissa järjestelmissä.

- Käyttöliittymän pystyy asentamaan ja käynnistämään komentoriviltä yhdellä shell-komennolla.
- Riippuvuuksineen noin 350,000 riviä koodia, itsetehtyä noin 20,000 riviä.
- Pystyy lataamaan YML tiedostot selaimeen näytettäväksi.
- Pystyy lisäämään YML tiedostoon käyttöliittymän kautta lisää luokkia ja luokille voi lisätä attribuutteja.
- Pystyy selaamaan luokkia vasemmalla laidalla olevan valikon kautta, sekä piilottamaan tai palauttamaan ne näkyviin.
- Luokan sekä attribuutin kaikkia tietoja pystyy muokkaamaan laajemmin molemmille toteutetun modaali-ikkunan kautta, joka avataan päänäkymästä, attribuuttien kohdalla muokkausnäkymä on mukautettu attribuutintyypin mukaan.
- Pystyy poistamaan luokkia ja niiden sisältämiä attribuutteja levyltä käyttöliittymästä käsin.
- Pystyy käskemään palvelinta ajamaan mallinnettujen luokkien migraatiot.
- Pystyy järjestämään attribuutit käyttöliittymässä eri järjestykseen raahaamalla ja muutokset tallentuvat YML-tiedostoon.
- Pystyy ryhmittelemään tiedostot vasemmassa valikossa kokonaisuuksiin luomalla ryhmiä ja raahaamalla luokkia niihin sekä luomaan ryhmien sisälle uusia ryhmiä.
- Pystyy vaihtamaan käyttöliittymän kieltä ja antamaan attribuuttien nimet useille eri kielille, esim. suomi, englanti ja ruotsi.
- Käyttöliittymässä on eri napeilla ohjetekstit, jotka ilmestyvät kun vie hiiren info kuvakkeen päälle.
- Laajennettavissa jatkokehityksellä.

Lähteet

Kirjalliset lähteet

Knuth, D. 2014. The Art of Computer Programming Volume 3 Second Edition. 31. painos. Westford: Courier Westford.

Shannon, C. & Weaver, W. 1964, The Mathematical Theory of Communication. 10. painos. Urbana: University Of Illinois Press.

Sähköiset lähteet

Atkins Jr, T. & Etemad, E. 2015. CSS Cascading and Inheritance Level 3 W3C Candidate Recommendation, 16 April 2015. Viitattu 8.10.2015. http://www.w3.org/TR/css-cascade-3/

Backbone.js documentation. 2015. Viitattu 7.10.2015. http://backbonejs.org/

Berjon , R. Doyle Navara, E. Faulkner, S. Hickson, I. Leithead, T. O'Connor, E. & Pfeiffer, S. 2014. HTML5 A vocabulary and associated APIs for HTML and XHTML W3C Recommendation 28 October 2014. Viitattu 15.9.2015. http://www.w3.org/TR/html5/

CoffeeScript documentation. 2015. Viitattu 29.9.2015. http://coffeescript.org/

döt Net, I. Evans, C. & Oren, B. 2009. YAML Ain't Markup Language (YAML™) Version 1.2 3rd Edition, Patched at 2009-10-01. Viitattu 12.9.2015. http://yaml.org/spec/1.2/spec.html

ECMA International. 2011. Standard ECMA-262 5.1 Edition / June 2011 ECMAScript® Language Specification. Viitattu 7.9.2015. http://www.ecma-international.org/ecma-262/5.1/

Fielding, R. &Reschke, J. 2014a. RFC7230. Viitattu 5.10.2015. https://tools.ietf.org/html/rfc7230

Fielding, R. & Reschke, J. 2014b. RFC7231. Viitattu 5.10.2015. https://tools.ietf.org/html/rfc7231

jQuery API each() function. 2015. Viitattu 12.9.2015. http://api.jquery.com/jquery.each/

Kacmarcik, G. & Leithead, T. 2015. UI Events (formerly DOM Level 3 Events) W3C Working Draft 28 April 2015. Viitattu 15.10.2015. http://www.w3.org/TR/uievents/

Node.js Foundation, Node.js v0.12.8 Manual & Documentation. 2015. Viitattu 15.9.2015. https://nodejs.org/docs/latest-v0.12.x/api/

Rodriguez, A. 2015. RESTful Web services: The basics 09 February 2015. Viitattu 25.9.2015. http://www.ibm.com/developerworks/library/ws-restful/ws-restful-pdf.pdf

Twitter. 2015. Bootstrap 3 documentation. Viitattu 15.10.2015. http://getbootstrap.com/css/

W3schools, 2015. JavaScript HTML DOM. Viitattu 30.9.2015.http://www.w3schools.com/js/js_htmldom.asp

Kuvat

Taulukot