KARELIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Jari Tolkki

GRAAFISEN VERKKOSIVUEDITORIN TEKNINEN SUUNNITELMA

Opinnäytetyö
Marraskuu 2019

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Marraskuu 2019
Tietotekniikan koulutusohjelma

Tikkarinne 9
80200 JOENSUU
+358 13 260 600

Tekijä
Jari Tolkki

Nimeke
Graafisen verkkosivueditorin tekninen suunnitelma

Toimeksiantaja
atFlow Oy

Tiivistelmä

Opinnäytetyön tavoitteena oli tuottaa tekninen suunnitelma graafisen verkkosivueditorin toteuttamista varten. Tässä opinnäytetyöraportissa käydään läpi projektin määritys- ja suunnitteluvaiheet, joissa on käytetty prototyyppilähestymistapaa. Lisäksi kerrotaan millaiset tekniset ratkaisut mahdollistavat editoriin suunniteltujen ominaisuuksien toteuttamisen. Käytetyt teknologiat ja ohjelmistokehitysmenetelmät vaiheineen esitellään raportin teoriaosuudessa.

Verkkosivueditoria on tarkoitus käyttää verkkoselaimella, ja se on yhdistettävissä toimeksiantajan omiin järjestelmiin. Editori tuotetaan pääasiassa erilaisilla web-ympäristössä käytettävillä teknologioilla, kuten Node.js:llä, jQueryllä ja MongoDB:llä. Teknologioiden käytettävyys varmennettiin tuottamalla ohjelmistoprototyyppi. Sitä käytettiin samalla suunnittelun tukena, ja testaamaan muidenkin ideoiden toimivuutta.

Prototyyppi, suunnitelmadokumentti ja sitä varten tehdyt UML-kaaviot toimitettiin toimeksiantajalle ohjelmiston tuotantoversion kehittämiseksi.

Tästä raportista on rajattu pois toimeksiantajan käyttämien järjestelmien yksityiskohdat.

Kieli
suomi

Sivuja 35
Liitteet 3
Liitesivumäärä 3

Asiasanat
ohjelmistosuunnitelma, web-teknologiat, vesiputousmalli, verkkosivueditori

THESIS
November 2019
Degree Programme in Information Technology

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600

Author
Jari Tolkki

Title
Technical plan for graphical web site editor

Commissioned by
atFlow Oy

Abstract

The purpose of this thesis was to produce a technical plan for programming of a graphical web page editor. This thesis report contains a description of the planning and specification phases which are made by the prototyping method. In addition, technical solutions which made it possible to implement the features planned for the editor are described. Technologies and software development methods with phases used during the planning are introduced in the theory part of this report.

The web page editor is intended to be used by a web browser, and it is connectable to systems used by the client. The editor is mainly produced by different web technologies, like Node.js, jQuery and MongoDB. The usability of these technologies was assured by creating a prototype. It was used to support planning and to test the usability of other ideas.

The prototype, planning document and the UML schemas made for the document were delivered for the client to produce the production version. This report does not contain details about the systems used by the client.

Language
Finnish

Pages 35
Appendices 3
Pages of Appendices 3

Keywords

software engineering, web technologies, waterfall model, web page editor

Sisältö

Käsite- ja lyhenneluettelo 5
1 Johdanto 9
2 Ohjelmistojen määrittely ja suunnittelu 9
2.1 Ohjelmistotuotanto vesiputousmallin mukaan 9
2.2 Määrittelyvaihe 10
2.3 Suunnitteluvaihe 11
2.4 Prototyyppimallinen suunnittelu 12
3 MongoDB-tietokanta 13
3.1 Yleistä tietokannoista 13
3.2 MongoDBn käyttö 14
3.3 MongoDBn vahvuudet ja heikkoudet 15
3.4 Mongoose 15
4 Tilanhallinta 16
5 Frontendin ohjelmistokehykset 17
5.1 Yleistä 17
5.2 React 18
6 Projektin määritysvaihe 18
6.1 Vaatimusmääritelmän luonti 18
6.2 Vaatimusanalyysi 19
7 Projektin suunnitteluvaihe 20
7.1 Suunnittelun aloitus 20
7.2 Arkkitehtuurisuunnittelu 21
7.3 Prototyyppi 22
7.4 Frontendin arkkitehtuurin suunnittelu 23
7.5 Backendin rakenne 24
7.6 Suunnitteluvaiheen havainnot 25
8 Frontendin toiminnallisuudet ja niiden suunnittelu 25
8.1 Tilanhallinta 25
8.2 Raahaustoiminto 26
8.3 Työtilaelementin ylläpito 27
8.4 Teemat 27
8.5 Komponenttien asetukset 28
9 Projektin lopputuotos 29
10 Pohdinta 30
Lähteet 32

Liitteet

Liite 1 Puurakenteisen datan kulku valmiin järjestelmän eri osissa
Liite 2 MongoDBllä ja Mongoosella toteutetun tietokannan rakenne
Liite 3 Tilanhallintajärjestelmän tietorakenne

Käsite- ja lyhenneluettelo

Backend on käyttäjälle näkymätön osa tietojärjestelmissä, verkkosivuilla tarkoittaa palvelimella ajettavaa ohjelmistoa (Rouse 2019a).

Bisneslogiikka (*Business Logic*) on käyttöliittymän ja tietokannan välissä oleva ohjelmointi, joka hoitaa erilaisten proseduurien toteuttamista (Rouse 2013)**.**

Bootstrap on responsiivinen frontend-kirjasto, joka sisältää valmiita web-ohjelmointia helpottavia CSS-luokkia ja JavaScript-toiminnallisuuksia (Bootstrap-team 2019).

CSS – Cascading Style Sheets. HTMLn kanssa käytettävä ulkoasunluontikieli, joka määrittää miltä HTML -elementin tulee näyttää esimerkiksi näyttöruudulla (developer.mozilla.org 2019a).

Denormalisoitu data tarkoittaa datan muotoa, jossa viittaukset muualla olevaan dataan on pyritty korvaamaan kopioimalla data useaan eri paikkaan. Datataulut kasvavat suuriksi, mutta niitä on nopea käyttää (Rouse 2017).

DOM – Document Object Model. Dokumenttimalli esittää HTML:stä rakennetun dokumentin rakennetta puurakenteena, ja se hallitsee verkkosivun toimintaa. Se on myös selaimen ohjelmointirajapinta, josta hallitaan dokumenttimallia (developer.mozilla.org 2019b).

Frontend on tietojärjestelmissä käyttäjälle näkyvä osa, tarkoittaen verkkosivuilla selaimelle ladattua sivua (Rouse 2019a).

HTML on kuvauskieli, jolla rakennetaan verkkosivuja (developer.mozilla.org 2019c).

JavaScript on kevyt, käytönaikaisesti käännettävä moniparadigmainen ohjelmointikieli, joka tunnetaan parhaiten käytöstä verkkosivujen frontendissä (developer.mozilla.org 2019d).

jQuery on JavaScript-kirjasto, joka sisältää verkkosivujen ohjelmointia helpottavia funktiota, jotka kykenevät manipuloimaan DOMia, CSSää ja tarjoavat AJAX-toiminnallisuuksia (w3schools.com 2019).

JSON - JavaScript Object Notation. JavaScriptin syntaksia muistuttava tekstimuotoinen formaatti, jolla voidaan esittää JavaScript-objekteja tekstimuodossa. Käytetään mm. tiedonsiirrossa kuljettamaan dataa (developer.mozilla.org 2019e).

JSX on JavaScriptin syntaksilaajennos. Syntaksiltaan muistuttaa HTML:ää, ja sitä käytetään yleensä Reactin kanssa verkkosivujen ulkoasujen rakentamisessa (reactjs.org 2019a).

Komponentti tarkoittaa tässä raportissa uudelleenkäytettävää verkkosivun elementtiä (esimerkiksi kuva, tekstialue tai verkkolomake), joka voidaan lisätä työstettävälle verkkosivulle helposti.

Node.js on JavaScriptille tehty ajoympäristö, jota käytetään pääasiassa verkkosovelluskehityksessä. Sisältää laajan pakettivalikoiman (>1 000 000 kirjastoa) ja on avoimen lähdekoodin projekti, jota voidaan käyttää alustariippumattomasti (nodejs.dev 2019).

Normalisoitu data on datan muoto, jossa datarakenteena käytetään viittauksia datan kopioinnin sijasta. Datataulut pysyvät pieninä, mutta niiden käyttö voi viedä resursseja. (Rouse 2017).

ODM - Object Document Mapper. Muokkaa dataobjektin sopimaan dokumenttipohjaiseen tietokantaan, kuten MongoDB:hen (Tyler 2018).

Polyfill on koodi, joka tarjoaa jonkin ohjelmallisen ominaisuuden selaimelle, jossa ominaisuutta ei ole. Käytetään esimerkiksi tarjoamaan HTMLn, CSSn ja JavaScriptin uusimpia ominaisuuksia Internet Explorerille (developer.mozilla.org 2019f).

Puurakenne on hierarkkinen tietorakenne. Se alkaa juurisolmusta, jolla voi olla useampia lapsisolmuja. Näillä voi myös olla lapsisolmuja jne. Tietorakenteen visuaalinen esitys muistuttaa puun oksistoa (Rouse 2005).

Relaatiotietokanta on tietokantatyyppi, johon data on tallennettu tietokantatauluihin. Tietokantataulujen välissä on viittauksia muihin tauluihin, joita hyödynnetään dataa haettaessa. Tietokantakielenä käytetään yleensä jotain SQL-kielistä (Rouse 2019b).

Renderöidä -termiä käytetään tässä raportissa kuvaamaan tietokoneen suorittamaa verkkosivun tai sen osan piirtämistä näytölle.

SPA - Single Page Application tarkoittaa verkkosivua, joka ei ohjaudu uudelle sivulle käyttäjän painaessa linkkejä tai painikkeita, vaan JavaScript generoi tarvittavan näkymän. Yleensä tarvittava data ja JavaScript -kirjastot latautuvat käyttäjän selaimeen sivulle tultaessa (Halme 2018).

Sisältöalue tarkoittaa tässä raportissa HTML:llä muodostettua aluetta, johon voidaan lisätä sivun rakentamisen aikana komponentteja. Käytetään alueen rajaamiseen muusta sivusta ja sisällön rakenteelliseen ryhmittelyyn.

Sisällönhallintajärjestelmä (*CMS - Content Management System*) on sovellus, jolla hallitaan digitaalista sisältöä, kuten esimerkiksi verkkosivujen sisältöä (Rouse 2019c).

Tietokantaskeema (*Database Schema*) on malli, joka kuvastaa tietokannan tai siinä olevan tietokantataulun sisältöä. Määrittelee kentät, niiden tietotyypit ja niiden väliset suhteet (tutorialspoint.com 2019).

Tuotantokäyttö tarkoittaa tässä raportissa ohjelmistotuotteen ”oikean” version käyttöä esimerkiksi työntekoon. Termiä käytetään erottamaan ohjelmiston tuotantoversio esimerkiksi kehitys- ja testiversioiden käytöstä.

UX/UI -suunnittelu (*User Experience / User Interface design*) tarkoittaa sovelluksen käyttöliittymän ja ulkoasun suunnittelua. Näiden tarkoituksena on mahdollistaa sovelluksen helppokäyttöisyys ja tarkoituksenmukainen ulkoasu (Turunen 2017).

Validoida (*Validate*) tarkoittaa ohjelmakoodin tai muun datan tarkastamista, että se täyttää sille asetetut muoto- ja sisältövaatimukset. Tarkastuksella pyritään estämään esimerkiksi epäkelvon datan joutumista tietokantaan (mongodb.com 2019a).

Websovellus (*Web Application*) on webpalvelimelle tallennettu sovellus, joka lähetetään käyttäjän webselaimeen käytettäväksi (Rouse 2019d).

1 Johdanto

Opinnäytetyö on toimeksianto joensuulaiselta verkkopalveluita tuottavalta atFlow Oy:ltä. Toimeksiannon tarkoituksena on tutkia ja suunnitella heidän käyttämäänsä web-sisällönhallintajärjestelmään täysin graafisella käyttöliittymällä toimiva verkkosivunrakentamisohjelma, PageBuilder. Tämän tarkoituksena on muuttaa verkkosivujen tekoprosessia helppokäyttöisemmäksi, jotta verkkosivustoja voisi rakentaa vähäisemmällä teknisellä osaamisella. Toimeksiannon tuloksena syntyi tekninen suunnitelma, jonka perusteella verkkosivunrakentamisohjelma voidaan rakentaa, sekä suosituksia valmiin sisällönhallintajärjestelmän kehittämiseksi mahdollistamaan yhteiskäyttö PageBuilderin kanssa (Tolkki 2019).

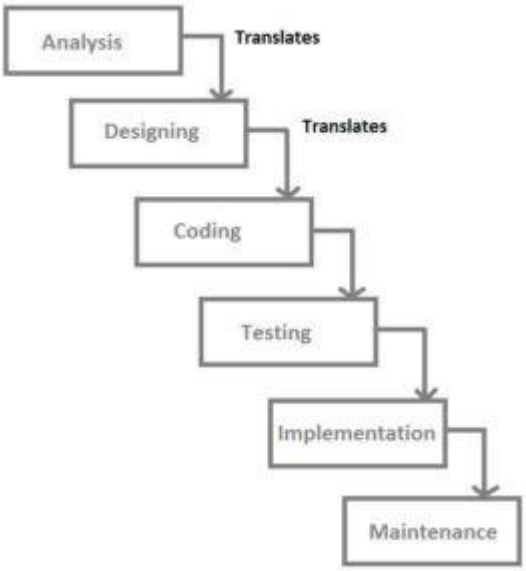
Tässä raportissa kerrotaan teoreettinen tietopohja vesiputousmallisesta ohjelmistokehityksestä, ja suunnittelussa käytetyistä ja siihen vaikuttaneista teknologioista. Tämän jälkeen raportissa käydään läpi projektin toteutusvaiheet, ja kerrotaan PageBuilderin keskeisimmistä ominaisuuksista ja niiden suunnittelusta. Lopuksi käydään läpi yhteenveto ja pohdinta koko projektista.

2 Ohjelmistojen määrittely ja suunnittelu

2.1 Ohjelmistotuotanto vesiputousmallin mukaan

Vesiputousmalli on eräs ohjelmistokehityksessä käytössä oleva vaihejakomalli, jossa ohjelmistotuotantoprosessi jaetaan useisiin, toisistaan poikkeaviin peräkkäisiin vaiheisiin. Edellinen vaihe tuottaa seuraavan vaiheen aloittamiseen tarvittavat asiat, kuten esimerkiksi arkkitehtuurisuunnitelman, jota tarvitaan ohjelmointityön aloittamisessa. Vaikka vesiputousmallissa jokainen vaihe suoritetaan loppuun ennen seuraavan aloittamista, tarvittaessa projekti tai sen osa voidaan palauttaa edelliseen vaiheeseen. Tämä on tarpeen, jos edellisen

vaiheen lopputuotos on puutteellinen. Vesiputousmalli sisältää yleensä ainakin määrittely-, suunnittelu- ja toteutusvaiheet (kuva 1). Toteutusvaihe voidaan jakaa ohjelmointi-, testaus- ja ylläpitovaiheisiin. (Haikala & Märijärvi 2004, 36-41)



Kuva 1. Vesiputousmalli (Reddy 2019).

Nykyisin vesiputousmalli on harvoin käytössä muiden vaihejakomallien ja ohjelmistosuunnittelumenetelmien korvattua sen. Nykyiset menetelmät kuitenkin perustuvat suurilta osin vesiputousmalliin, jonka vuoksi sitä voidaan käyttää periaatteellisena pohjana kuvaamaan ohjelmistokehityksen eri vaiheita. Vesiputousmallin hyviä puolia ovat selkeät tehtävät jokaiselle eri vaiheelle, ja jokaisesta vaiheesta syntyy selkeitä tuloksia. Mallin huonoja puolia ovat sen jäykkyys ja siitä johtuva huono virhesietoisuus. Parhaimmillaan vesiputousmalli on pienen mittakaavan projekteissa, joissa projektien vaatimukset ymmärretään ja hallitaan hyvin. (Pal 2019)

2.2 Määrittelyvaihe

Määrittelyvaiheessa (*analysis*, kuvassa 1) ohjelmistoprojektille asetetaan tavoitteet ja vaatimukset. Vaihe aloitetaan suorittamalla esitutkimus, jossa kartoitetaan projektin tilaajan tarpeet. Kartoitetuista tarpeista muodostetaan asiakasvaatimukset, jotka kuvaavat, miksi tilaaja tarvitsee projektista syntyvää ohjelmistoa. Asiakasvaatimusten pohjalta muodostetaan ohjelmistovaatimukset.

Ohjelmistovaatimuksissa asiakasvaatimukset on muutettu kuvaamaan toteutettavalle järjestelmälle asetettavia vaatimuksia. (Haikala & Märijärvi 2004, 37-39)

Vaatimusten pohjalta määritellään ohjelmiston toiminnot, rajoitukset, yhteydet muihin järjestelmiin ja ei-toiminnalliset vaatimukset, kuten käytettävyys. Näistä muodostettavaa dokumenttia kutsutaan toiminnalliseksi määrittelyksi. Tämä sisältää vaatimukset ja niiden täyttävän järjestelmän kuvauksen toimintoineen. Dokumentissa tilaajalta tulleet vaatimukset on muutettu ohjelmistokehitykseen sopiviksi määrittelyiksi. Tämä dokumentti on määrittelyvaiheen lopputulos, jota käytetään suunnitteluvaiheessa. (Haikala & Märijärvi 2004, 37-39, 78-81)

2.3 Suunnitteluvaihe

Suunnitteluvaiheessa (*design* kuvassa 1) tehdään toiminnallisen määrittelyn pohjalta projektissa käytettävä tekninen määrittely. Se vastaa *miten*-kysymykseen, jos toiminnallisen määrittelyn ajatellaan vastaavan *mitä*-kysymykseen. Suunnitteluvaiheessa tehtävä työ voidaan jakaa kahteen tasoon: arkkitehtuurisuunnitteluun ja moduulisuunnitteluun. (Haikala & Märijärvi 2004, 40, 81)

Arkkitehtuurisuunnittelun tarkoituksena on jakaa tuotettava järjestelmä pienempiin kokonaisuuksiin, moduuleihin. Moduuleiden on tarkoitus olla pieniä itsenäisiä kokonaisuuksia, joita voidaan suunnitella ja kehittää toisistaan riippumatta. Arkkitehtuurisuunnittelu on moduulien tehtävien ja keskinäisten rajapintojen suunnittelua. Projektin jako moduuleihin pienentää projektin monimutkaisuutta ja lisää sen hallittavuutta. (Haikala & Märijärvi 2004, 81-83)

Moduulisuunnittelu keskittyy moduulien sisäisen rakenteen ja toiminnallisuuden suunnitteluun arkkitehtuurisuunnitelman tarjoamissa puitteissa. Moduuli voidaan tarvittaessa jakaa vielä pienempiin kokonaisuuksiin, alimoduuleihin, jotta suurempi moduuli pysyisi hallittavampana. Moduulisuunnittelu voidaan myös

yhdistää moduulien ohjelmoinnin ja testaamisen kanssa samaksi vaiheeksi. (Haikala & Märijärvi 2004, 81-83)

Suunnitteluvaiheen lopputuloksena syntyy tekninen määrittely, jonka pohjalta voidaan aloittaa varsinainen ohjelmointi. Määrittely pitää sisällään arkkitehtuuri- ja moduulisuunnittelusta syntyneet kuvaukset ja suunnitelmat. Näitä ovat mm. järjestelmän kuvaus, moduulien tehtävä- ja rajapintakuvaukset, tietokanta-arkkitehtuuri sekä yleiset tekniset toimintaperiaatteet. (Haikala & Märijärvi 2004, 40, 81-83)

2.4 Prototyyppimallinen suunnittelu

Prototyyppimallinen lähestymistapa ohjelmistokehitysprosessiin on keino toteuttaa projektista kokeiluversio, ennen varsinaisen projektin määrittelyn ja suunnittelun aloittamista. Tätä lähestymistapaa käytetään, jos jonkin teknisen ratkaisun tai käyttöliittymän käyttökelpoisuutta tulee testata, tai epäselvää asiakasvaatimusta tulee selventää käytännön kokeilun kautta. Projektin prototyyppivaihe sisältää samat vesiputousmallin vaiheet kuin itse varsinainen projekti, mutta lopputulos on yleensä huomattavasti pienempi kuin varsinaisen projektin lopputulos. (Haikala & Märijärvi 2004, 42-43)

Prototyyppikehitys jaetaan yleensä neljään eri tyyppiin. Nämä ovat:

- Kertakäyttöprototyypitys (*throwaway prototyping*), jossa prototyyppi tehdään nopeasti teknisen ratkaisun tai käyttöliittymän käyttökelpoisuuden testaamista varten. Tämä prototyyppimalli tuotetaan nopeasti. Ohjelmakoodi ei välttämättä ole tuotantokäyttöön soveltuvaa, jonka vuoksi sitä ei jatkokäytetä varsinaisessa projektissa.
- Evolutiivinen prototyypitys (*evolutionary prototyping*), jossa prototyyppi tehdään huolellisesti varsinaisen projektin ohjelmiston ytimeksi. Projektia jatketaan lisäämällä prototyypin ympärille muita tarvittavia ominaisuuksia.
- Inkrementaalinen prototyypitys (*incremental prototyping*), jossa tehdään useita eri prototyyppejä vastaamaan järjestelmän eri osia. Nämä osat

yhdistellään yhdeksi toimivaksi tuotantoon sopivaksi kokonaisuudeksi projektin edetessä.

- Äärimmäinen prototyypitys (*extreme prototyping*). Tätä käytetään yleensä websovelluksia kehitettäessä. Tässä mallissa tuotetaan ensin pelkkä frontend, sitten backend simuloidaan tarjoamaan mock-vastauksia kutsuihin, ja lopulta backend kehitetään toimivaksi. Tätä prototyyppimallia käytetään sovelluksissa, joissa ulkoasun ja frontendin toimintoja tulee pystyä tarkastelemaan sovellusta kehitettäessä. (Hurbans 2017)

Prototyyppimallin hyvinä puolina voidaan pitää nopeasti saatavaa näkyvää tulosta asiakkaalle. Sen vuoksi kehittäjät saavat nopeasti palautetta järjestelmästä ja ongelmalliset kohdat järjestelmässä voidaan paikallistaa nopeasti. Siksi ohjelmiston toimintaa voidaan muuttaa pienehköin muutoksin. Mallin huonoina puolina pidetään sen hitautta, kertakäyttöprototyypin kohdalla kalleutta ja nopean palautesyklin vuoksi projekti voi muuttua paljon lyhyessä ajassa, mikä voi aiheuttaa vaikeuksia projektin ylläpidettävyydessä. (guru99.com 2019a)

3 MongoDB-tietokanta

3.1 Yleistä tietokannoista

Tietokanta on järjestelty kokoelma dataa. Tietokantoja käytetään tiedon tallentamiseen, etsimiseen ja käsittelyyn. (guru99.com 2019b) Tietokantoja on monia erilaisia tyyppejä, kehitettyinä eri käyttötarkoituksiin. Yksi yleinen tapa luokitella tietokantoja on jakaa ne kahteen eri tyyppiin: SQL-kieliä käyttäviin ja ei-SQL-kieliä käyttäviin (*NoSQL*) tietokantoihin. (Smallcombe 2019)

SQL-tietokannat käyttävät muodoltaan ja sisällöltään esimääriteltyjä tietokantatauluja, ja tiedon tallentamiseen ja hakemiseen SQL -kieliä. SQL-tietokannoissa data tallennetaan taulukkomuodossa, joten samassa tietokantataulussa olevilla riveillä tulee olla samat tietokentät. Tietokantataulun

muotoa kutsutaan kaavioksi (*database schema*), joka määritellään tietokantataulua luotaessa. (Smallcombe 2019) SQL-kyselyjä suorittavat komennot ajetaan SQL-kielen komennoista ja arvoista koostuvilla hakulausekkeilla. Arvoja ovat esimerkiksi tietokantataulun ja tietokentän nimet. SQL-kielessä komennot ovat pääosin englannin kielen verbejä ja kenttien avainten nimet substantiiveja, joten hakulausekkeet voivat muistuttaa jossain määrin luonnollista kieltä: *SELECT name FROM students WHERE age > 18;* (geeksforgeeks.org 2019)

NoSQL-tietokantatyyppiin taas kuuluu monia erilaisia tietokantoja. Näitä yhdistää pääasiassa se, että ne eivät käytä SQL-kieltä, eikä muita SQL-tietokantojen piirteitä. Tietorakenteeltaan NoSQL-kannat ovat yleensä joustavia, eivätkä aina pakota tallennettavaa dataa olemaan muotoiltuna tiettyyn kaavioon. NoSQL-tietokannoissa tietokannan tarkempi rakenne ja datan hakemisen ja tallentamisen tavat riippuvat käytössä olevan tietokannan ominaisuuksista. (Smallcombe 2019)

3.2 MongoDBn käyttö

MongoDB on NoSQL-ryhmään kuuluva tietokanta. Sen tarkoitus on tallentaa JSON-muotoista dataa dokumentteihin, jotka voivat olla rakenteeltaan joustavia ja moniulotteisia. (mongodb.com 2019b) SQL:n tietokantataulua vastaa MongoDB:ssä kokoelma (*collection*), joka sisältää dokumentteja. Samassa kokoelmassa olevien dokumenttien rakenne voi poiketa toisistaan. Lisäksi dokumentit voivat sisältää alidokumentteja, sekä viittauksia muissa kokoelmissa oleviin dokumentteihin. (mongodb.com 2019c)

Toisin kuin SQL-tietokannoissa, MongoDB:ssä tietokantakyselyjen syntaksi koostetaan valitsemalla tietokantaobjektista halutun toiminnon toteuttava metodi, jolle asetetaan argumentiksi JSON-muodossa olevat hakuparametrit: *db.cardatabase.find({ model: "Volvo", mileage: { $lt: 200000 } }).* (mongodb.com 2019d)

3.3 MongoDBn vahvuudet ja heikkoudet

MongoDBn vahvuutena on sen hyvä yhteistoiminta erilaisten JavaScriptillä toteutettujen järjestelmien kanssa, koska molempien tietorakenteet voidaan muuttaa suoraan JSON:ksi. Lisäksi MongoDB:n joustavuus dokumenttien rakenteessa mahdollistaa sen käyttämisen mielivaltaisten dokumenttien tallentamiseen, joiden datan rakenteissa voi olla keskinäisiä eroja. (Noworyta 2018)

MongoDB tukee hyvin dokumentteja, joissa data ei sisällä juurikaan viittauksia muihin dokumentteihin tai sitten viittaukset voidaan korvata datan kopioinnilla alidokumentiksi (Mei 2013). Lisäksi MongoDB sisältää nykyisin SQL:n *left outer join* -hakuominaisuutta vastaavan *$lookup* -komennon (mongodb.com 2019e). Nämä eivät kuitenkaan tarjoa SQL-tietokantojen kaltaista tukea relaatiomalliselle tiedolle, joka sisältää paljon viittauksia muihin dokumentteihin. MongoDB:ssä viittausten liika käyttö useampaan kokoelmaan johtaa helposti sekavaan koodiin, jota on hankala ylläpitää ja joka on altis virheille. (Cookson 2019)

3.4 Mongoose

Mongoose on node.js-kirjasto, joka toimii tietokannan yläpuolisessa sovelluskerroksessa (Parsons 2017). Se toimii ODM-komponenttina bisneslogiikan ja MongoDB:n välissä mahdollistaen vahvasti tyypitettyjen kaavioiden (*schema*) käytön tietokannan dokumenteissa, ja muita toiminnallisuuksia, joita tavataan SQL-tyyppisissä tietokannoissa. Kaavioiden käytöllä on tarkoitus parantaa MongoDB:n kykyä tallentaa olio-ohjelmoinnissa käytettyjä objekteja tietokantaan lähes sellaisenaan. Tällöin olio-ohjelmoinnin käsitettä *luokka* vastaa Mongoosessa oleva malli (*model*), joka sisältää tietorakennetta hallitsevan kaavion. (Munro 2017)

Vaikka Mongoose käyttääkin kaavioita tietorakenteen hallitsemiseen, mallin tietorakenne toimii joustavasti. Samalla se mahdollistaa mallissa olevat kaaviot

sisältämään muita kaavioita sisäkkäisenä tietorakenteena (*nesting*) (Parsons 2017). Mallit voivat sisältää viittauksia muihin malleihin (Munro 2017).

Mongoose mahdollistaa omien validointifunktioiden lisäämisen malliin. Validointifunktioiden tarkoituksena on varmistaa mallin avulla tallennettavan datan oikeellisuus ja estää väärän datan tallennus tietokantaan. (Munro 2017) Vaikka Mongoose tarjoaakin paljon MongoDB:n ominaisuuksia lisääviä ja muuttavia toimintoja, niin tämä tapahtuu suorituskyvyn kustannuksella johtuen sen ajamisesta tietokannan yläpuolisessa sovelluskerroksessa (bugwheels94 2018).

4 Tilanhallinta

Tilanhallinta (*state management*) on olennainen osa nykyaikaisia interaktiivisia websovelluksia. Tila websovelluksessa tarkoittaa, että sovelluksella on oma sisäinen tilansa, joka voi koostua useasta eri muuttujasta. Usein tila on riippuvainen käyttäjän tekemistä toimista. Tilanhallintajärjestelmiä käytetään pitämään sovelluksen tila keskitettynä kontrolloituihin tietorakenteisiin, joiden sisällön muuttaminen voi olla rajoitettu tapahtumaan vain muutamalla metodilla, jotka noudattavat tarkkoja muotosääntöjä. Nämä tekevät tilasta helpomman hallita ja hyödyntää. Tilanhallinta pitää sisällään websovellukselle relevanttia dataa, joka voi vaikuttaa sovelluksen ulkoasuun tai siitä voidaan renderöidä käyttäjälle sisältödataa. (Moss 2019)

Websovelluksessa tila vaikuttaa lähes aina sovelluksen ulkoasuun. Kun tila muuttuu, sovellus uudelleenrenderöidään tarvittavin osin (Schulz 2018). Esimerkiksi kun Redux-tilanhallintajärjestelmää käytetään React-ulkoasukirjaston kanssa, niin React-komponentit kytketään kuuntelemaan haluttuja osia Reduxin hallitsemasta tilasta. Kun React-komponentin kuuntelema osa tilasta päivittyy, komponentti saa tiedon muutoksesta ja renderöi ulkoasunsa uudelleen vastaamaan saamaansa dataa. Komponentti voi myös päivittää tilaa käyttäjän toiminnon seurauksena. Tällöin komponentti lähettää tilanhallintajärjestelmälle ennalta määritellyn toiminnon sekä toiminnon kohteen.

Tämä johtaa tilan muuttumiseen, sekä mahdollisesti jonkin toisen komponentin ulkoasun päivittymiseen muualla sovelluksessa. (redux.js.org 2019)

Tilanhallintajärjestelmiä on monta erilaista, ja niillä on eri filosofioita tilanhallinnan toteutuksessa. Esimerkiksi Redux on järjestelmä, joka sisältää yksisuuntaisen ja tarkasti kontrolloidun datan kulun tietovarastoon ja siitä muualle. Sen käyttämän tietovaraston rakenne on muodoltaan litteä ja yksiuloitteinen. Tarvittaessa moniuloitteiset tilarakenteet voidaan muuttaa yksiuloitteisiksi ns. normalisoimalla data. (redux.js.org 2018) Toinen yleisesti käytetty tilanhallintajärjestelmä on MobX. Se voi sisältää useamman datavaraston, ja tietorakenteiden muoto voi olla mielivaltainen. Lisäksi MobX ei sisällä tarkkaa ennaltamääriteltyä datan kulkua, jonka mukaan järjestelmän tilaa päivitetään. (Ravichandran 2019)

5 Frontendin ohjelmistokehykset

5.1 Yleistä

JavaScriptillä frontendiin tehtyjä ohjelmistokehyksiä (*software framework*) käytetään nykyaikaisessa webkehityksessä. Ne ovat suosittuja, koska ne tarjoavat toimintoja, jotka vähentävät tarvittavan koodin määrää nopeuttaen ja tehostaen websovellusten kehitystyötä. (Andras 2017) Ennen ohjelmistokehyksiä websovellusten frontend kehitettiin kirjoittamalla puhdasta HTML:ää, CSS:ää ja JS:ää. Suurissa projekteissa tämä kuitenkin johti ohjelmakoodin vaikeaan ylläpidettävyyteen. Tähän kehitettiin avuksi web-ohjelmistokehykset, joita kehitettiin eri painotuksilla ja toimintafilosofioilla. Yhteistä useimmille niistä ovat seuraavat ominaisuudet:

- Sovelluksen tilan muutosten ja näkymien synkronointi
- Linkkien luonnin automatisointi reitittämällä
- Uudelleenkäytettävät komponentit
- Valmiit komponentti- ja sivupohjat (Joliat 2019)

5.2 React

React on yksi suosituimmista web-ohjelmistokehyksistä (Hannah 2018), vaikka teknisessä mielessä se ei täytä ohjelmistokehyksen vaatimuksia. Se on vain JS-kirjasto. (Kryzhanovska 2019) Reactia käytetään käyttöliittymien toteutukseen. Tilanhallinta ja reititys voidaan toteuttaa kolmannen osapuolen kirjastoilla. Tämä on johtanut siihen, että Reactille on saatavilla monipuolisesti erilaisia lisäkirjastoja, joilla Reactia käyttävän sovelluksen ominaisuuksia lisätään, ja sovellus voidaan toteuttaa käyttämällä erilaisia ohjelmointityylejä. (Hannah 2018)

Reactia käytettäessä websovellus voidaan jakaa pieniin, kevyisiin ja uudelleenkäytettäviin komponentteihin, joita voidaan sisällyttää toisiin komponentteihin. Tällöin voidaan käyttää JSX-syntaksia, joka mahdollistaa React-komponenttien käyttämisen HTML-koodissa käyttämällä itse luotuja HTML-tageja (reactjs.org 2019b), Joustavalla ohjelmointityylillä ja hyvällä suunnittelulla komponenteista voidaan tehdä yleiskäyttöisiä, ja niitä voidaan käyttää useassa eri sovelluksessa. Tämä mahdollistaa erilaisten valmiiden komponenttikirjastojen tekemisen tehostamaan sovelluskehitystyötä (Capozzi 2019).

6 Projektin määritysvaihe

6.1 Vaatimusmääritelmän luonti

Määritysvaihe alkoi, kun toimeksiantajalta saatu kuvaus toivotuista ominaisuuksista tarvitsi suunnittelua ja toteutusta varten lisämäärityksiä. Tätä varten suoritettiin esitutkimus määritysten selvittämiseksi. Tutkimus aloitettiin haastattelemalla toimeksiantajan sisällönhallintajärjestelmän parissa työskenteleviä henkilöitä. Haastattelulla selvitettiin mitkä ominaisuudet ja toiminnot olisivat PageBuilderissa hyödyllisiä heidän kokemuksensa perusteella. Tämän lisäksi toimeksiantajan kanssa käytiin läpi tutkimuksen aikana

esiinnousseita ideoita ja niiden jatkojalostamista tai hylkäämistä. Tutkimuksen tuloksena syntyi vapaamuotoinen vaatimusmääritelmä, joka kuvasi ihanteellista kaikenkattavilla toiminnoilla varustettua verkkosivunrakenninta (Tolkki 2016).

Vaatimusmääritelmän ytimenä oli tuottaa graafisella käyttöliittymällä toimiva webselaimella käytettävä sovellus, jolla voitaisiin toteuttaa kaikki normaalit sivustonluonnissa ja -kehityksessä tarvittavat toimenpiteet. Näistä suurimpina tarpeina olivat uudelleenkäytettävien komponenttien asettelu verkkosivulle raahaamalla ja pudottamalla (*drag and drop*) ja komponenttien toimintaan ja ulkoasuun vaikuttavien asetusten muuttaminen graafisesta käyttöliittymästä. (Tolkki 2016)

Sovellusta ajavan palvelimen määrittely jätettiin pois projektista toimeksiantajan itse suunniteltavaksi, koska se olisi laajentanut projektissa suunniteltavaa asiaa liikaa.

PageBuilderin ulkoasuvaatimuksia ei juurikaan määritelty tässä vaiheessa kuin karkealla tasolla. Sovellus on ns. Single Page Application, jossa näkyy sama näkymä koko ajan. Käyttäjälle näkyy keskellä työtilaelementti, jossa rakennettava verkkosivu näkyy. Vasemmalla on komponenttiluettelo, josta voidaan raahata työtilaelementtiin komponentteja. Ylhäällä on työkaluvalikot, joista voidaan hallita esimerkiksi käyttöön valittua teemaa. Kuvassa 2 näkyy kuvakaappaus sovelluksen prototyypistä, mikä havainnollistaa ulkoasun rakennetta.

6.2 Vaatimusanalyysi

Tutkimuksen tuloksena syntynyt vaatimusmääritelmä oli laaja ja se piti sisällään paljon kehitysideoita (Tolkki 2016). Koska käytettävissä oleva aika oli rajallista, niin vaatimusmäärittelystä tekniseen suunnitteluun päätyivät ainoastaan tärkeimmät toiminnallisuudet. Sivuun jääneet toiminnallisuudet voidaan suunnitella ja toteuttaa ydintoiminnallisuuksien ympärille tulevaisuudessa PageBuilderin jatkokehityksen yhteydessä.

Ydintoiminnallisuuksiin kuuluu verkkosivustojen luontitoiminnallisuus, jota käytetään graafisen käyttöliittymän kautta yhdistelemällä ja konfiguroimalla valmiiksi ohjelmoituja komponentteja. PageBuilderilla on myös tarkoitus editoida sillä tehtävien verkkosivustojen ulkoasua ja teemaa.

Muita tutkimuksessa ja vaatimusanalyysissä esiin nousseita asioita ovat olleet työprosessien muuttaminen PageBuilderin kanssa paremmin yhteensopivimmiksi ja PageBuilderiin tarvittavien valmiiden komponenttien rakenne ja konfiguroitavuus. (Tolkki 2016)

7 Projektin suunnitteluvaihe

7.1 Suunnittelun aloitus

Määritysvaiheen perusteella päätettiin, että suunnitelman ytimenä olisi valmiiden komponenttien lisääminen tyhjälle sivulle, ja niiden editointi niihin liitetyillä asetuksilla. Tällä tavoin tehty verkkosivu tulisi vielä tallentaa tietokantaan, ja olla lopulta muutettavissa oikeaksi verkkosivuksi toimeksiantajan sisällönhallintajärjestelmään. Nämä toiminnallisuudet muodostavat koko projektin ytimen, ja projektin arkkitehtuurin ensisijainen tavoite olisi mahdollistaa nämä toiminnot. Muut toiminnot tuli suunnitella näiden toiminnallisuuksien toteutuksen ehdoilla.

Suunnitteluvaiheen alussa päätettiin soveltaa prototyypittämistä vaiheen aikana, koska valmiiden komponenttien lisäämiseen sivulle oli useita mahdollisia keinoja, joiden toimivuudesta ei ollut täyttä varmuutta. Prototyyppiä käytettiin vain asioiden teknisen toteutettavuuden testaamiseen alusta alkaen, eikä ohjelmakoodia ollut tarkoitus tehdä varsinaiseen tuotantokäyttöön sopivaksi. Tämän vuoksi kertakäyttöprototyypitys valittiin prototyypin kehitystavaksi.

7.2 Arkkitehtuurisuunnittelu

Arkkitehtuurisuunnittelu aloitettiin tarkastelemalla PageBuilderin ydintoiminnallisuuden toteutettavuutta käytännössä. Sivunrakennuksesta syntyvä data on muodoltaan puurakenteista. Se koostuu useasta sisäkkäisestä tasosta, joita ovat *sivusto*, *sivu*, *sisältöalue* ja *komponentti*. Sivusto-taso voi sisältää useita sivuja, joissa voi olla useita sisältöalueita, joissa voi olla useita komponentteja. Tähän hierarkiaan perustuu datan puurakenteinen muoto, jota PageBuilderilla syntyy sivurakennuksen tuloksena. Puurakenteinen data haluttiin pitää yksinkertaisuuden vuoksi denormalisoituna. Tämä tarkoittaa sitä, että kaikissa vaiheissa oli mukana mahdollisesti syvälle ulottuvia tietorakenteita.

Backendissä puurakenteisen datan käsittely tarkoitti MongoDB:n käyttöönottoa tietokannaksi, koska se kykenee tallentamaan puurakenteisia tietorakenteita. Koska backendin toiminnallisuudet ovat toteutettavissa pitkälti valmiilla node.js:lle tehdyillä kirjastoilla, sinne tehtävää koodia ei tarvitse jakaa moduuleihin.

Frontendin arkkitehtuurin suunnittelun aikana nojauduttiin päätöksiin, jotka tehtiin prototyypittämisessä esiin tulleiden havaintojen perusteella. Frontend jaettiin kolmeen eri moduuliin, jotka vastaavat sivurakenteen datan tilapuun ylläpidosta, työnäkymän päivittämisestä ja työkalujen ja muun kokonaisuuden ylläpidosta.

Koko PageBuilderin arkkitehtuurin perusideana voidaan pitää puurakenteisen datan rakentamisen, kuljettamisen, tallentamisen ja ulosviennin mahdollistaminen eri kerroksissa ilman että datan rakennetta muokataan paljoakaan missään vaiheessa, lukuunottamatta luontivaiheessa tapahtuvia muutoksia. Liitteen 1 kaavio havainnollistaa valmiin järjestelmän rakennetta datan liikumisen näkökulmasta.

7.3 Prototyyppi

Prototyypin ensisijaisena tarkoituksena oli testata PageBuilderin komponenttien raahaa ja pudota -toiminnon toteutettavuutta erilaisilla JavaScript-kirjastoilla. Tätä varten tehtiin nopeasti jQuery-kirjastolla pelkkä frontend ilman backend -toiminnallisuuksia.

Prototyyppiä laajennettiin tarpeen mukaan kattamaan muut toteuttavuuden tutkimista tarvitsevat ominaisuudet. Näitä olivat teeman vaihtaminen, komponenttien asetusten ja tilanhallintajärjestelmän toteutettavuus. Komponenttien asetusten ja tilanhallintajärjestelmän tutkimisen yhteydessä prototyypille luotiin backend. Tämän yhteydessä testattiin suunnitellun tietokantarakenteen toimivuus.

Tietokantaa lukuunottamatta prototyypin front- ja backend toteutettiin kokonaan JavaScriptillä ja sillä luoduilla valmiilla kirjastoilla niiden laajan kattavuuden ja nopean käyttöönoton vuoksi. Koska jQuery-kirjaston havaittiin toimivan parhaiten kaikissa toteutettavuustestauksissa, se päätettiin käyttöönotettavaksi myös varsinaisessa PageBuilderissa. Prototyyppiä varten tehty ohjelmakoodi ei kuitenkaan ole tuotantokäyttöön soveltuva, koska kattavuuden lisäksi siinä käytetty ohjelmakoodin taso on huonolaatuista ja testaamatonta.

Prototyypin frontendin arkkitehtuurin periaate on kuitenkin sama kuin mikä on tarkoitettu tuotantoversioon, joten prototyypin koodia voidaan tarvittaessa käyttää lisätestaukseen, mikäli suunnitelmasta syntynyt ohjelmistosuunnitelma ei ole riittävän kattava.

Kuva 2. Kuvakaappaus prototyypistä

7.4 Frontendin arkkitehtuurin suunnittelu

Koska suurin osa toiminnallisuudesta on frontendissa, komponenttien lisääminen ja sivun muokkaaminen vaatii hyvän keinon hallita ja renderöidä dataa. Siksi päätettiin soveltaa jo olemassa olevaa mallia, nk. Flux-arkkitehtuuria, jota yleensä käytetään Reactin ja Fluxin/Reduxin yhdistelmän kanssa (Wheeler 2016). Mallia yksinkertaistettiin sisältämään pelkästään työstettävän sivun rakentamiseen tarvittavat toiminnot. Tässä tapauksessa tilanhallinnan tehtäväksi jäi työkalussa työstettävän sivun tilan ylläpito.

Komponentit ja sisältöalueet oli suunnitteluvaiheen alussa tarkoitus tuottaa Reactilla johtuen sen hyvästä tuesta verkkosivun koostamiselle valmiista komponenteista. Prototyypityksessä tätä varten luotiin muutama React-komponentti toteutettavuustestausta varten. Näitä testattiin raahaamalla jQueryn raahaustoiminnolla PageBuilderin työtilaelementtiin, missä yhteydessä huomattiin, ettei niiden renderöinti ollut ongelmatonta. Reactilla luotuun sisältöalueeseen ei voitu lisätä sisältöä raahaamalla, eikä raahaustoiminnon vaatimia JavaScript-kuuntelijoita saatu toimimaan React-komponenteissa. Syy tähän on Reactin ohjelmointifilosofiassa, joka pyrkii päivittämään DOMia eri tavalla kuin jQuery (Shoemaker 2016). Tämän johdosta Reactin käytöstä luovuttiin, ja komponenttien rakenne ja päivitys hoidetaan HTML:llä ja jQuery:llä.

Frontendin ja backendin välinen tiedonsiirto suunniteltiin toteutettavaksi JSON:lla. Kyseinen tekstinotaatio on muutettavissa helposti JavaScript-dataksi ja toisinpäin, ja kielessä on olemassa siihen tarvittavat työkalut valmiina (developer.mozilla.org 2019g).

Frontend koostuu kolmesta moduulista, jotka vastaavat PageBuilderin työpöydällä olevan rakennettavan sivun päivittämisestä, tilapuun ylläpitämisestä ja muun PageBuilderin ylläpitämisestä. Tilapuusta ja rakennettavan sivun ylläpitämisestä kerrotaan lisää luvussa 8 Frontendin toiminnallisuudet ja niiden suunnittelu.

7.5 Backendin rakenne

Tietokannaksi valittiin MongoDB, koska sen havaittiin soveltuvan vapaamuotoisten dokumenttien tallentamiseen muita yleisesti käytettyjä tietokantoja paremmin. Lisäksi siihen on helppo tallentaa JSON-muotoista dataa (mongodb.com 2019f), joka mahdollistaa kevyen backend-ratkaisun. Tietokannan tarkoituksena on toimia pelkästään tietovarastona, johon kohdistetaan vain yksinkertaisia luku- ja kirjoitusoperaatioita, joilla uudelleenkirjoitetaan tai luetaan projektikohtainen data kerralla.

MongoDBn ja bisneslogiikan väliin päätettiin ottaa käyttöön Mongoose-kirjasto. Sen tarkoituksena on yhdistää MongoDB ja Node.js:n päällä oleva bisneslogiikka. Prototyypissä suoritetussa toteutettavuuskokeilussa havaittiin sen tarjoavan helpon sovellusrajapinnan MongoDB-tietokantaan verrattuna pelkkään MongoDB:n Node.js-ajuriin. Heikomman suorituskyvyn ei katsottu haittaavan Mongoosen käyttöä, koska sillä ei ole tarkoitus ajaa monimutkaisia ja suorituskyvyllisesti hankalia hakuja. Mongoosen valintaa tuki sen kyky validoida tietokantaan tallennettava sivurakennusdata. Datan rakenteen ja sisällön on tärkeää olla oikeanlaista, ettei myöhemmässä kehitysvaiheessa tai datan siirrossa sisällönhallinnanjärjestelmään tule ongelmia mahdollisesti viallisen datan vuoksi. Mongooselle tehdyn tietokannan rakenne on vapaamuotoisena UML-kuvana liitteessä 2.

REST-rajapinta toteutetaan Express -kirjastolla, joka on node.js:lle tehty valmis palvelinkirjasto. Se muodostaa rajapinnan backendin ja frontendin väliselle kommunikaatiolle. Kirjasto automatisoi matalan tason protokollakutsut ja tukee REST-arkkitehtuuria. (Wieruch 2019)

Datansiirto toimeksiantajan käyttämään sisällönhallintajärjestelmään rajattiin tämän suunnitteluprojektin ulkopuolelle, ettei projekti laajene liikaa. Datansiirto tapahtunee karkeasti suunniteltuna ottamalla tietokannasta sivustoprojektin data ulos JSON:na, ja ajamalla se sisällönhallintajärjestelmään liitettyyn ohjelmaan, joka luo järjestelmään sisällöt ja rakenteet PageBuilderista tulleen datan perusteella.

7.6 Suunnitteluvaiheen havainnot

Suunnitteluvaiheessa arkkitehtuuri- ja moduulisuunnittelu tapahtuivat suurilta osin samanaikaisesti johtuen prototyyppimäisestä kehitystavasta, ja arkkitehtuuritason valinnat perustuivat prototyypittämisessä saatuihin havaintoihin ja ideoihin. Tämä ei vastannut täysin vesiputousmallin suunnitteluvaiheen jakoa arkkitehtuuri- ja moduulisuunnitelmiin itsenäisinä kokonaisuuksina (Haikala & Märijärvi 2004, 40, 81). Tämä oli kuitenkin tarpeen toimivan kokonaisuuden aikaansaamiseksi.

8 Frontendin toiminnallisuudet ja niiden suunnittelu

8.1 Tilanhallinta

Kun verkkosivua rakennetaan PageBuilderissa, teemaa vaihdetaan tai asetuksia muutetaan, niin muutos tallennetaan työkalussa olevaan tilanhallintaan. Tilanhallinta toimii puurakenteisesti, jotta siihen saadaan tallennettua helposti hierarkkisesti asettuvia sivuja, sisältöalueita ja komponentteja asetuksineen.

Tämän muotoista rakennetta on helppo ylläpitää. Tilapuuhun tallennettavan datan määrä pyritään ylläpitämään pienenä tiedon hallittavuuden ja siirrettävyyden parantamiseksi. Esimerkiksi komponentteihin viitataan pelkästään id-numeroilla, jolloin mahdollinen komponentin HTML-koodin muutos ei sotkisi datan yhdenmukaisuutta. Suunnitelmassa ja prototyypissä tilanhallintajärjestelmää käytetään pelkästään PageBuilderissa työstettävän sivuston datan tallentamiseen. Liite 3 havainnollistaa tilanhallintajärjestelmän tietorakennetta.

Tilapuu rakennettiin prototyypissä puhtaalla JavaScriptillä tyhjästä, johtuen muiden yleisesti käytössä olevien tilanhallintakirjastojen puutteista, kuten tuen puute puurakenteelle tai hankala yhdistäminen itsetehtyyn komponenttijärjestelmään, jota käytetään suoraan DOM:a muokkaamalla. Prototyypin alkuvaiheessa kokeiltiin käyttää MobX-tilahallintajärjestelmää osana prototyyppiä, mutta sen projektiin yhdistämiseen olisi kulunut liikaa aikaa saavutettaviin hyötyihin nähden, kun samat toiminnot oli mahdollista ottaa käyttöön puhtaalla JavaScriptillä. Mikäli PageBuilderia ohjelmoitaessa tai myöhemmin laajennettaessa tulee tarve toisenlaiselle tilanhallinnalle, kuten vaikka versionhallinnan vuoksi, niin tällöin on suositeltavaa implementoida tuotteeseen jokin jo olemassa oleva tilanhallintajärjestelmä, kuten MobX hallitsemaan muita tiloja.

8.2 Raahaustoiminto

Raahaustoiminnon (*drag and drop*) on tarkoitus mahdollistaa komponentin lisääminen sivulle raahaamalla komponentti hiirellä valikosta ja tiputtamalla se haluttuun kohtaan. Loppukäyttäjälle tämä on yksinkertainen ja helposti ymmärrettävä toimenpide, mutta työkalun suunnittelussa tämä vaati monen asian huomioonottamista ja erilaisten ratkaisujen toimivuuden testaamista prototyypissä.

Raahattava komponentti työkalussa on vain ”esikatselukuva” sivulle ilmestyvästä komponentista. Kun raahattava komponentti tiputetaan sisältöalueeseen,

sisältöaluetta tilapuussa edustavaan solmuun lisätään uusi lapsisolmu, johon raahatun komponentin ID lisätään. Tämän jälkeen työtilaelementti uudelleenrenderöidään ja komponentti ilmestyy ruudulle.

Lisäksi raahaustoiminto vaatii, että käyttöön valitussa sivuasetelmassa on komponenteille sopivia paikkoja, joihin ne voidaan asettaa. Tämän tarkoituksena on estää komponenttien lisääminen alueille, jotka myöhemmissä käyttövaiheissa estäisivät oikean verkkosivun luomisen ohjelmallisesti.

8.3 Työtilaelementin ylläpito

Kun PageBuilder ladataan selaimeen, työtilaelementtiin renderöidään Bootstrapin Container -elementin sisältävä pohjasisältöalue. Tämä luo pohjan sivun työstämiselle. Kun tähän raahataan komponentteja, niin ne lisätään pohjasisältöalueen alle tilapuuhun luvussa 8.2 Raahaustoiminto kuvatulla tavalla. Kun tilapuu päivitetään, samalla työtilaelementin DOM tyhjennetään sinne lisätyistä komponenteista. Tämän jälkeen tilapuuta käydään läpi rekursiivisesti. Jokainen solmu sisältää komponentin ID-numeron. Sillä haetaan selaimen muistissa olevasta taulukosta komponentin HTML-koodi, joka lisätään ylätason DOM-elementtiin. Kun tilapuu on käyty läpi, sivuun tehdyt muokkaukset näkyvät käyttäjälle. Myös komponentin tai sivun asetusten muokkaaminen päivittää tilapuuta, joka johtaa työstettävän sivun uudelleenrenderöintiin.

8.4 Teemat

Ulkoasukehitystä nopeuttaa, jos verkkosivun tekijä voi valita sivuston ensisijaisen, toissijaisen ja muut värit valitsemalla, ja ne automaattisesti ilmestyvät komponentteihin ennalta määritellyille paikoilleen. Samalla menetelmällä voidaan määrittää käytettävä fontti, ja sen koko ja paksuus. PageBuilderissa on tämä keskeinen ominaisuus, vaikka se rajoittaakin ulkoasusuunnittelun käyttämään komponenttien tekovaiheessa määriteltyjä ratkaisuja.

Teknisesti tämä toteutetaan CSS-muuttujilla ja ennalta määritellyllä CSS-luokkien nimeämiskäytännöillä. Jokaista teemaa varten kirjoitetaan oma muuttujalistansa, jota voidaan PageBuilderissa vaihtaa työn aikana.

Teemaa ei käytetä sivun asettelun ja komponenttien koon hallinnassa, koska teeman tarkoituksena ei ole muuttaa sivun rakennetta. Rakennetta hallitaan muilla keinoin, kuten sivuasettelua tai komponentin ulkoasua vaihtamalla. Rakenne toteutetaan Bootstrap -CSS-kirjastolla.

Teeman valintamahdollisuutta testattiin prototyypissä. CSS-muuttujien vaihtoehtona oli SASS/SCSS -node.js-kirjasto. Jälkimmäinen ei kuitenkaan sovellu selaimella käytettäväksi suoraan, ja sen muuttaminen selainversioksi tutkimusta varten ei olisi ollut tarkoituksenmukaista. CSS-muuttujien huonona puolena on sen toimimattomuus suurimpiin selaimiin kuuluvalla Internet Explorer 11 -selaimella (caniuse.com 2019), johon on kuitenkin saatavilla polyfill-versio.

8.5 Komponenttien asetukset

Komponenttien tulee olla mahdollisimman monikäyttöisiä, jottei jokaiseen projektiin tarvitse tehdä omia komponentteja. Siksi niiden toiminnallisuutta tulee hallita projektikohtaisten asetusten avulla.

Komponenttien asetukset tallennetaan JSON-muodossa tietokantaan. Tarkempi rakenne määritellään komponenttikohtaisesti. Tässä suunnitelmassa paneudutaan asetusnäkymän tarvitseman järjestelmän tekniseen toteutukseen ja sen tarvitsemien rajapintojen suunnitteluun.

Koska asetusnäkymässä pitää voida toteuttaa tarvittaessa monimutkaisia ja erilaisia konfiguraatioita (kuten rakentaa HTML-lomake), niin ei ole mielekästä tehdä yleiskäyttöistä, konfiguroitavaa asetusnäkymää. Sen sijaan jokaiselle komponentille tulee ohjelmoida oma asetusnäkymänsä ja sen tarvitsemat JavaScript-toiminnallisuudet. Näitä varten määriteltiin oma rajapinta, joiden avulla asetusnäkymää ajavat funktiot ovat yhteydessä muuhun järjestelmään.

Kun komponentin asetuksia muutetaan, järjestelmä kutsuu rajapinnan avulla asetusnäkymän luovaa funktiota. Kun asetusnäkymä on luotu, ja käyttäjä on tehnyt muutoksensa, asetusnäkymä kutsuu rajapinnan sille tarjoaman funktion avulla muuta järjestelmää, joka tallentaa muutokset ja tarvittaessa uudelleenrenderöi työstettävän verkkosivun.

Komponenteille voidaan tehdä useampi mahdollinen ulkoasutiedosto. Niitä hallitaan myös komponenttien asetusnäkymästä. PageBuilder vaihtaa komponentille uuden ulkoasun, kun sellainen on vaihdettu asetuksista. Suurin osa muista asetuksista tulee todennäköisesti koskemaan komponentin backend-toiminnallisuutta toimeksiantajan sisällönhallintajärjestelmässä, joten niiden vaihtaminen ei enimmäkseen muuta mitään näkyvää PageBuilderissa.

9 Projektin lopputuotos

Suunnitteluprojektin lopputuloksena toimeksiantaja sai perusteet PageBuilderin ohjelmointivaiheen aloittamiseen. Nämä perusteet pitivät sisällään dokumentin, joka pitää sisällään tekniset ja toiminnalliset määrittely- ja suunnitteludokumentit sanallisessa muodossa. Dokumentti sisältää ohjelman arkkitehtuuria, moduuleja, tietorakenteita ja tietovirtoja kuvaavat UML-kaaviot ja prototyypin lähdekoodin (Tolkki 2019). Jos jokin asia jää suunnitelmissa epäselväksi, prototyypin koodista voidaan tarkistaa, kuinka asiaa on testattu. Lisäksi prototyyppiä voidaan tarvittaessa muokata ja uudelleenkäyttää epäselväksi jääneen asian selvittämiseksi.

Osa UML-kaavioista on piirretty vapaamuotoisesti, pelkästään selventämään tai tukemaan dokumentissa mainittua toiminnallisuutta. Pääpaino suunnitelman toiminnallisuuksien teknisestä toiminnasta kertomisella on dokumentissa olevalla sanallisella selostuksella.

10 Pohdinta

Valittu aihealue paljastui jälkikäteen laajaksi, ja tässä projektissa keskityttiin ydintoiminnallisuuksien tutkimiseen ja suunnitteluun. Määrittelyvaiheen alussa tehdyssä alkuperäisessä vaatimusmäärittelyiden keräämisessä kirjatut ominaisuudet olisivat kaksin- tai kolminkertaistaneet suunnitteluprojektiin kuluneen ajan. Kun tämä paljastui myöhemmin vaiheen aikana, päätin keskittyä ydintoiminnallisuuksien tarkempaan määrittelyyn ja suunnitteluun. Suunnitteluvaiheessa toissijaiset toiminnallisuudet (kuten versionhallinta) pelkästään tiedostettiin olevan mahdollisesti tulossa tulevaisuudessa, joten tekninen toimivuus suunniteltiin mahdollistamaan niiden lisättävyys järjestelmään.

Määritysvaiheessa tein yhteistyötä toimeksiantajan työntekijöiden kanssa saadakseni vaatimusmäärityksen tuotettua. Suunnitteluvaiheessa toimin pääosin itsenäisesti vaatimusmääritykseen nojautuen (Tolkki 2016). Suunnitteluvaiheella ei ollut itseni lisäksi muita siihen nimettyjä tekijöitä. Tiedossani ei ole kuinka projektia tullaan jatkamaan suunnitelman palautuksen jälkeen.

Suunnitelmassa olleiden ydintoiminnallisuuksien ulkopuolelle jää PageBuilderin UX/UI-selvitys- ja suunnittelutyö, jonka näkisin tärkeänä osana graafisen työkalun ja sen käytettävyyden suunnittelussa.

Reactin ja Vue.js;n sopivuus komponenttien toteuttamistekniikaksi kokeiltiin melko alkuvaiheessa projektia. Tämän jälkeen molemmat kirjastot ovat päivittyneet ja niihin on saattanut tulla uusia ominaisuuksia, jotka mahdollistavat komponenttien rakentamisen niillä. Lisäksi projektin teon aikana Google julkaisi Polymer-kirjaston, joka tarjoaa mahdollisuuden luoda kustomoituja komponentteja websovelluksiin.

Backendiin valitut teknologiat sen sijaan ovat vieläkin käyttökelpoisia johtuen backendin yksinkertaisesta toiminnallisuudesta, toimiessaan lähinnä tietovarastona ja REST-rajapintana frontendille. Tulevia mahdollisia toiminnallisuuksia voisivat olla versionhallintajärjestelmä tai monen yhtäaikaisen

käyttäjän tuki. Jos ne lisätään, backendin toiminnallisuuksien määrä kasvaa. Siksi saattaa olla tarpeen lisätä muita teknologioita Express.js:n rinnalle, tai vaihtaa Express.js johonkin muuhun tilanteeseen paremmin sopivaan kirjastoon.

Lähteet

Andras, S. 2017. JavaScript Frameworks, why and when to use them. hello.JS. https://blog.hellojs.org/javascript-frameworks-why-and-when-to-use-them-43af33d0608d 29.10.2019

Bootstrap-team. 2019. Bootstrap. https://getbootstrap.com. 6.11.2019

bugwheels94. 2018. Performance Difference in Mongoose vs MongoDB Native Driver. medium.com. https://medium.com/@bugwheels94/performance-difference-in-mongoose-vs-mongodb-60be831c69ad. 3.11.2019

caniuse.com. 2019. CSS Variables (Custom Properties). Can I use. https://caniuse.com/#feat=css-variables. 23.9.2019

Capozzi, M. 2019. The Three Levels of Reusability in React. hackernoon.com. https://hackernoon.com/the-three-types-of-reusable-react-components-37a6bf7c2d69. 29.10.2019

Cookson, R. 2019. The Pros and Cons of MongoDB. virtual-dba.com. https://www.virtual-dba.com/pros-and-cons-of-mongodb. 2.11.2019

developer.mozilla.org. 2019a. CSS: Cascading Style Sheets. https://developer.mozilla.org/en-US/docs/Web/CSS. 6.11.2019

developer.mozilla.org. 2019b. DOM (Document Object Model). https://developer.mozilla.org/en-US/docs/Glossary/DOM. 6.11.2019

developer.mozilla.org. 2019c. HTML. https://developer.mozilla.org/en-US/docs/Glossary/HTML. 6.11.2019

developer.mozilla.org. 2019d. JavaScript. https://developer.mozilla.org/en-US/docs/Web/JavaScript. 6.11.2019

developer.mozilla.org. 2019e. Working with JSON. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. 6.11.2019

developer.mozilla.org. 2019f. Polyfill. https://developer.mozilla.org/en-US/docs/Glossary/Polyfill. 6.11.2019

developer.mozilla.org. 2019g. JSON. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON. 27.10.2019

geeksforgeeks.org. 2019. Structured Query Language (SQL). https://www.geeksforgeeks.org/structured-query-language/. 30.10.2019

guru99.com. 2019a. Prototyping Model in Software Engineering: Methodology, Process, Approach. https://www.guru99.com/software-engineering-prototyping-model.html. 24.10.2019

guru99.com. 2019b. What is Database? What is SQL? https://www.guru99.com/introduction-to-database-sql.html. 30.10.2019

Haikala, I. Märijärvi, J. 2004. Ohjelmistotuotanto. Helsinki: Talentum.

Halme, A. 2018. Mikä on Single Page App ja mihin sitä käytetään? citydevlabs.fi https://citydevlabs.fi/single-page-app/. 6.11.2019

Hannah, J. 2018. The Ultimate Guide to JavaScript Frameworks. JavaScript Report. https://jsreport.io/the-ultimate-guide-to-javascript-frameworks/. 29.10.2019

Hurbans, R. 2017. To prototype or not to prototype: that is the question. freecodecamp.org. https://www.freecodecamp.org/news/to-prototype-or-not-to-prototype-that-is-the-question-2f85c8cde2b/. 24.10.2019

Joliat, L. 2019. Do we still need JavaScript frameworks? freecodecamp.org. https://www.freecodecamp.org/news/do-we-still-need-javascript-frameworks-42576735949b/. 29.10.2019

Kryzhanovska, A. 2019. Top 10 Web Development Frameworks in 2019-2020. gearheart.io. https://gearheart.io/blog/top-10-web-development-frameworks-2019-2020/. 29.10.2019

Mei, S. 2013. Why You Should Never Use MongoDB. sarahmei.com: http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb/. 27.10.2019

mongodb.com. 2019a. Schema Validation. https://docs.mongodb.com/manual/core/schema-validation/. 6.11.2019

mongodb.com. 2019b. What Is MongoDB? https://www.mongodb.com/what-is-mongodb. 30.10.2019

mongodb.com. 2019c. Data Modeling Introduction. https://docs.mongodb.com/manual/core/data-modeling-introduction/. 30.10.2019

mongodb.com. 2019d. Query Documents. https://docs.mongodb.com/manual/tutorial/query-documents/. 30.10.2019

mongodb.com. 2019e. $lookup (aggregation). https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/. 2.11.2019

mongodb.com. 2019f. JSON and BSON. https://www.mongodb.com/json-and-bson. 27.10.2019

Moss, R. 2019. What is state? Why do I need to manage it? egghead.io. https://egghead.io/articles/what-is-state-why-do-i-need-to-manage-it. 28.10.2019

Munro, J. 2017. An Introduction to Mongoose for MongoDB and Node.js. tutsplus.com. https://code.tutsplus.com/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527. 3.11.2019

nodejs.dev. 2019. Introduction to Node.js. https://nodejs.dev/. 6.11.2019

Noworyta, T. 2018. The pros and cons of using MongoDB. medium.com. https://medium.com/selleo/the-pros-and-cons-of-using-mongodb-8e7601857a8e. 30.10.2019

Pal, S. K. 2019. Software Engineering | Classical Waterfall Model. geeksforgeeks.org. https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/. 22.10.2019

Parsons, N. 2017. Part 2: Introducing Mongoose to Your Node.js and Restify API. mongodb.com. https://www.mongodb.com/blog/post/part-2-introducing-mongoose-to-your-nodejs-and-restify-api. 3.11.2019

Ravichandran, A. 2019. A definitive guide to Redux vs. MobX. blog.logrocket.com: https://blog.logrocket.com/redux-vs-mobx/. 28.10.2019

reactjs.org. 2019a. Introducing JSX. https://reactjs.org/docs/introducing-jsx.html. 6.11.2019

reactjs.org. 2019b. Components and Props. https://reactjs.org/docs/components-and-props.html. 29.10.2019

Reddy, S. 2019. Waterfall Methodology in Project Management — Phases, Benefits. medium.com. https://medium.com/@sudarhtc/waterfall-methodology-in-project-management-phases-benefits-85393be2f1d. 22.10.2019

redux.js.org. 2018. Normalizing State Shape. https://redux.js.org/recipes/structuring-reducers/normalizing-state-shape. 28.10.2019

redux.js.org. 2019. Usage with React. https://redux.js.org/basics/usage-with-react. 28.10.2019

Rouse, M. 2005. Tree structure. searchdatamanagement.techtarget.com. https://searchdatamanagement.techtarget.com/definition/tree-structure. 6.11.2019

Rouse, M. 2013. Business Logic. whatis.com. https://whatis.techtarget.com/definition/business-logic. 6.11.2019

Rouse, M. 2017. Denormalization. searchdatamanagement.techtarget.com. https://searchdatamanagement.techtarget.com/definition/denormalization. 6.11.2019

Rouse, M. 2019a. Front end and back end. whatis.techtarget.com. https://whatis.techtarget.com/definition/front-end. 6.11.2019

Rouse, M. 2019b. Relational database. searchdatamanagement.techtarget.com. https://searchdatamanagement.techtarget.com/definition/relational-database. 6.11.2019

Rouse, M. 2019c. Content management system (CMS). searchcontentmanagement.techtarget.com. https://searchcontentmanagement.techtarget.com/definition/content-management-system-CMS. 6.11.2019

Rouse, M. 2019d. Web application (Web app). searchsoftwarequality.techtarget.com. https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app. 6.11.2019

Schulz, D. 2018. Comparison of state management solutions for Reactmedium.com. https://medium.com/dailyjs/comparison-of-state-management-solutions-for-react-2161a0b4af7b. 28.10.2019

Shoemaker, S. 2016. If you think you need JQuery in your React app you're doing it wrong. medium.com. https://medium.com/@wisecobbler/if-you-think-you-need-jquery-in-your-react-app-you-re-doing-it-wrong-77899ed7217e. 2.11.2019

Smallcombe, M. 2019. SQL vs. NoSQL: How Are They Different and What Are the Best SQL and NoSQL Database Systems? xplenty.com. https://www.xplenty.com/blog/the-sql-vs-nosql-difference/. 30.10.2019

Tolkki, J. 2016. PageBuilderin vaatimusmääritelmä 1.0.1. Vaatimusmääritelmädokumentti

Tolkki, J. 2019. Tekninen suunnitelma. Ohjelmiston tekninen suunnitelmadokumentti

Turunen, S. 2017. Design-termistötutuksi: Näin UI-, UX- ja visuaalinen suunnittelu eroavat toisistaan. lamia.fi. https://lamia.fi/blog/design-termisto-tutuksi. 16.11.2019

tutorialspoint.com. 2019. DBMS - Data Schemas. https://www.tutorialspoint.com/dbms/dbms_data_schemas.htm. 6.11.2019

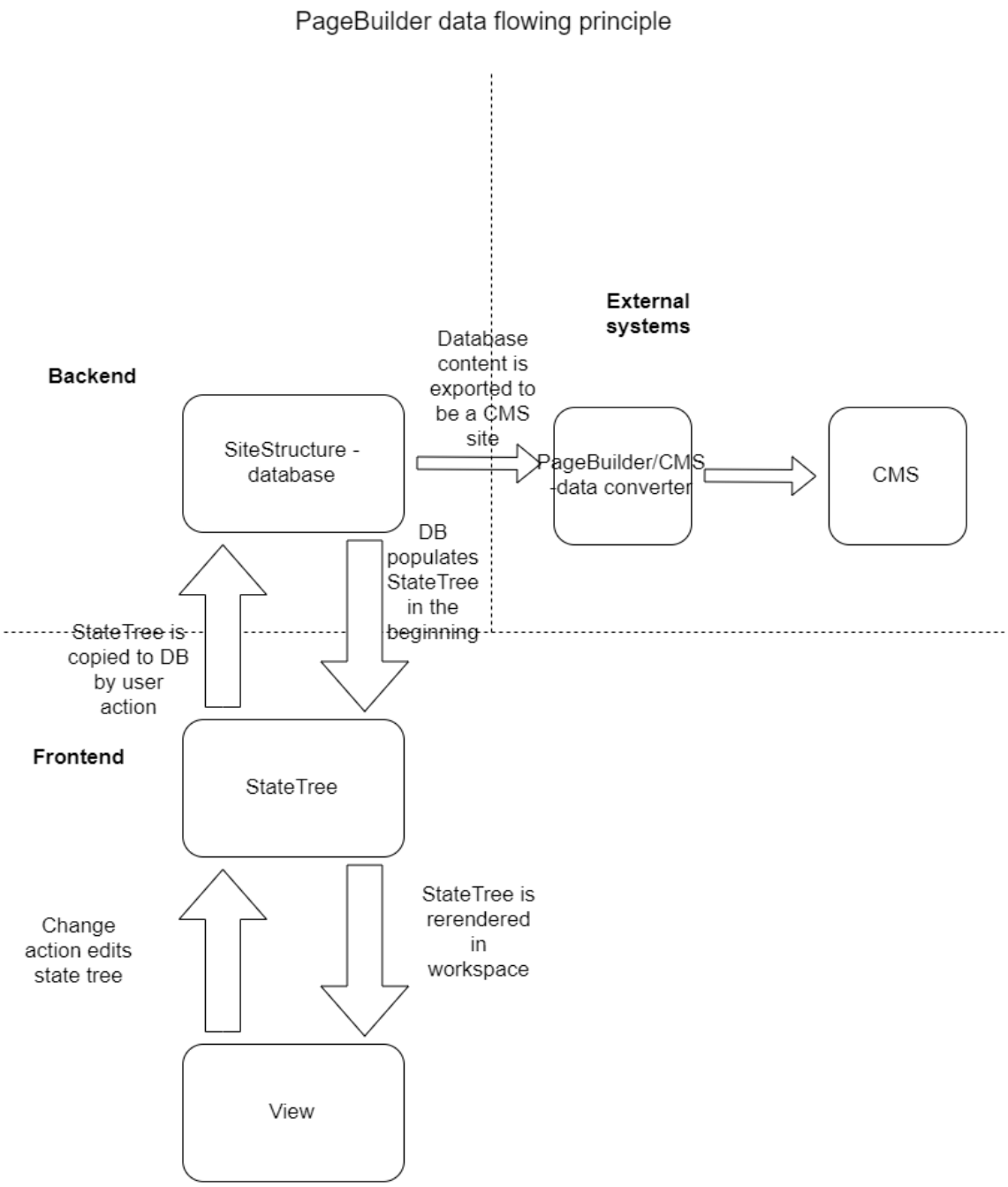
Tyler, J. 2018. What is the difference between ODM and ORM? medium.com. https://medium.com/@julianam.tyler/what-is-the-difference-between-odm-and-orm-267bbb7778b0. 6.11.2019

w3schools.com. 2019. jQuery Introduction. https://www.w3schools.com/jquery/jquery_intro.asp. 6.11.2019

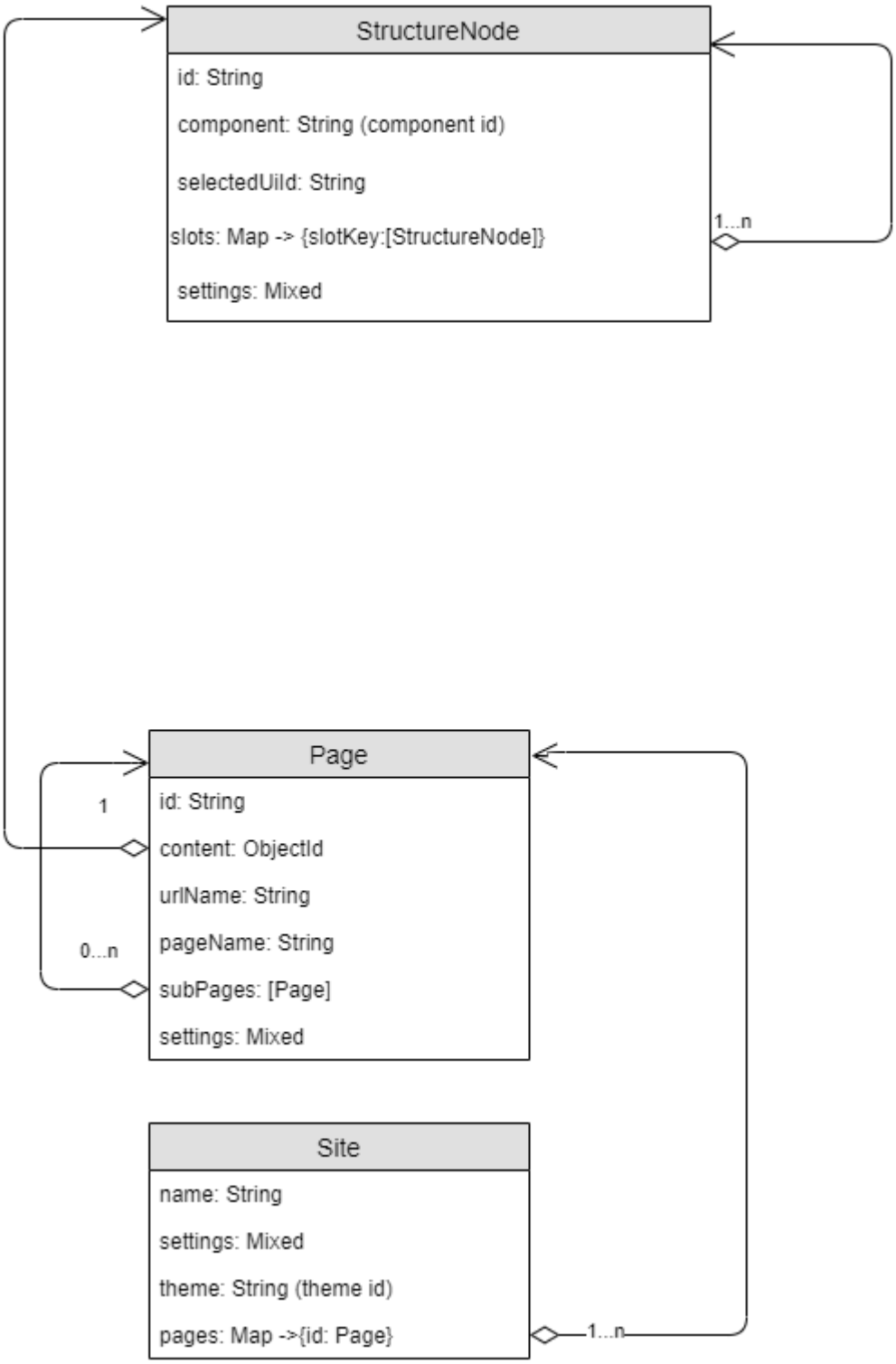
Wheeler, K. 2016. Getting To Know Flux, the React.js Architecture. scotch.io. https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture. 27.10.2019

Wieruch, R. 2019. How to create a REST API with Express.js in Node.js. robinwieruch.de: https://www.robinwieruch.de/node-express-server-rest-api. 3.11.2019

Puurakenteisen datan kulku valmiin järjestelmän eri osissa



MongoDBllä ja Mongoosella toteutetun tietokannan rakenne



Tilanhallintajärjestelmän tietorakenne