VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Niklas Granö, Simo Niemelä, Olli Salmu

# REAALIAIKAISEN TAPAHTUMAKALENTERIN TOTEUTTAMINEN NYKYAIKAISIN MENETELMIN

Liiketalous ja matkailu

2011

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

# TIIVISTELMÄ

| | |
|---|---|
| Tekijä | Niklas Granö, Simo Niemelä, Olli Salmu |
| Opinnäytetyön nimi | Reaaliaikaisen tapahtumakalenterin toteuttaminen nykyaikaisin menetelmin |
| Vuosi | 2011 |
| Kieli | suomi |
| Sivumäärä | 105 + 2 liitettä |
| Ohjaaja | Sirkka Hellman, Raija Tuomaala |

Opinnäytetyön tarkoituksena oli kehittää reaaliaikainen tapahtumakalenteri, joka erottuisi jo olemassa olevista vastaavanlaisista ratkaisuista. Haluttiin luoda sivusto, joka kokoaa kaikki tapahtumat yhdelle sivulle ja josta tapahtumat löytyvät vaivattomasti tarkentavan haun ja sijaintiedon käytön perusteella. Opinnäytetyön aihe valittiin niin, että siinä voitaisiin soveltaa mahdollisimman laajasti jo opittuja sekä uusia menetelmiä ja kehittää omaa osaamista. Projektimme tavoitteena oli edistää käyttäjäkokemusta luomalla intuitiivinen käyttöliittymä, jossa tiedon tuoreus, ryhmittely ja selkeys olivat ensisijaisen tärkeitä.

Käytimme teoriaa projektin toteuttamisen sekä kuvaamisen apuna. Keskeisiä käsitteitä olivat projektinhallintamenetelmät, suunnittelumallit, käyttöliittymät ja käytettävyys sekä ohjelmointikielet. Käytimme projektin toteutuksessa uusimpien ohjelmointimenetelmien lisäksi useita projektinhallintamenetelmiä. Projektinhallintamenetelmien avulla pysyimme sovelluksen kannalta oleellisissa asioissa ja sovelluksen kehitys oli johdonmukaista ja tehokasta. Teoria-aineistona hyödynsimme aihealueiden kirjallisuutta ja internetissä olevia asiantuntijoiden artikkeleita.

Onnistuimme suunnitellun aikataulun puitteissa luomaan tapahtumakalenterin sille asetettujen vaatimusten mukaisesti. Saimme toteutettua sovellukseen reaaliaikaisen käyttöliittymän ja rakenteeltaan selkeän ulkoasun. Havaitsimme haasteeksi sivuston sisällön tuottamisen ja käyttäjien mielenkiinnon luomisen ja ylläpitämisen. Totesimme, että uusimpien menetelmien käyttö ei välttämättä ratkaise sivuston jatkuvuuteen liittyviä ongelmia, vaan sisällön määrä, laatu ja saatavuus ovat sovelluksen kriittisiä menestystekijöitä.

| | |
|---|---|
| Avainsanat | Web-sovellus, reaaliaikaisuus, projektinhallinta, Ruby on Rails, Node.js |

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittelyn koulutusohjelma

# ABSTRACT

| | |
|---|---|
| Author | Niklas Granö, Simo Niemelä, Olli Salmu |
| Title | Implementation of a Real-Time Event Calendar with Modern Techniques |
| Year | 2011 |
| Language | Finnish |
| Pages | 105 + 2 attachments |
| Name of Supervisor | Sirkka Hellman, Raija Tuomaala |

The aim of this thesis was to develop a real-time event calendar application, which would stand out from similar already existing applications. This application project was chosen as the subject of the thesis so that we could implement as many of the things learned during our studies as possible. We also wanted to include a set of the newest methods available, and of course, to learn new things along the way. The aim of the project was to enhance the user-experience by creating an intuitive user interface, up-to-date, well-grouped and uncluttered content as our primary objective. The aim was also to create a website that would collect all the events on one site, and where the events would be effortlessly found, using the filtered search as well as the geolocation provided by the web browser of the user.

Theory was used in both the execution and the documentation of the project. The essential topics were project management methods, design patterns, programming languages, user-experience design and usability in web-design. We used not only the newest programming methods but also several project management methods. By using numerous project management tools, we were able to focus on the most relevant matters concerning the project and keep the development of the project consistent and efficient. Both literature and internet sources were used as theoretical reference.

We succeeded in creating an event calendar application within the set schedule and fully met the requirements set beforehand. We were able to create a real-time user interface and a structurally uncluttered layout for our application. We observed that the biggest challenge was in generating the actual content and in attracting and maintaining the interest of the users. It was also established that the problems concerning the future success of the web site are not necessarily resolved by using the newest techniques, but rather the quantity, quality and reachability of the content are the crucial success factors of the application.

Keywords: Web-Application, Real-Time, Project Management, Ruby on Rails, Node.js

# SISÄLLYS

**KUVIOLUETTELO**

## KOODIESIMERKKILUETTELO



## LIITELUETTELO

# ASIASANAT

### ActiveRecord

Rails-sovelluksen mallit periytyvät ActiveRecord-luokasta. Sen avulla päästään käsiksi sovelluksen tietokantaan. (Ruby, Thomas & Hansson 2011, 33–34)

### Asset Pipeline

Asset Pipeline tarjoaa sovelluskehyksen, jolla voidaan yhdistää komentosarja- ja tyylitiedostot yhdeksi tiedostoksi. Asset Pipeline osaa myös käsitellä CoffeeScript-tiedostoja. (Asset Pipelines 2011)

### Asynkroninen

Asynkroninen suoritus ei ole ajallisesti toisesta suorituksesta riippuvainen eli suoritukset ovat ajasta ja paikasta riippumattomia. (Asynchronous applications 2003)

### Breadcrumb

Suomeksi murupolku. Sitä voidaan kutsua toissijaiseksi navigoinniksi, josta voidaan samalla nähdä käyttäjän sen hetkinen sijainti sivustolla. (Smashing Magazine 2009)

### Callback

Callback on tavallinen metodi, jonka muistiosoitetta voidaan hyödyntää eri puolilla ohjelmaa. Metodi, joka käyttää muistiosoitetta kutsuakseen Callback-metodia, ei välttämättä tiedä, mistä metodista on kyse. (Sriram Srinivasan 1997)

### CoffeeScript

CoffeeScript on pieni ohjelmointikieli, jonka tavoitteena on helpottaa JavaScript-ohjelmointikielen hyvien omaisuuksien hyödyntämistä. CoffeeScript-koodi muunnetaan JavaScript-koodiksi CoffeeScriptin JavaScript-kääntäjän avulla, jotta ohjelmakoodia voitaisiin ajaa selaimessa. (Burnham 2011, xvi–xviii)

**Comet**

Comet on web-sovellusmalli, joka mahdollistaa tiedon lähettämisen palvelimelta selaimelle ilman erikseen lähetettyä pyyntöä. Cometia voidaan käyttää esimerkiksi tapahtumalähtöisten reaaliaikaisten sovellusten kehittämiseen. (Gravelle 2011)

**CSS3**

CSS, Cascading Style Sheets - suomeksi tyylitiedosto, määrittää sivuston tyylit erillään koodista. CSS3 on CSS:n uusin standardi. (CSS3 Tutorial 2011)

**Doctype**

Doctype on HTML-tiedoston ensimmäinen määritys, joka kertoo selaimelle, mitä versiota merkintäkielessä käytetään, Doctype-merkintä viittaa dokumenttityyppimääritykseen, joka määrittelee merkintäkielessä käytettävän säännöstön niin, että selain osaisi näyttää sivuston oikein. (HTML <!DOCTYPE> Declaration 2011)

**DRY**

Lyhenne sanoista “Don’t Repeat Yourself”. (Dont Repeat Yourself 2011)

**Front-end**

Front-end on sovelluksen käyttäjälle näkyvä osa eli käyttöliittymä. (Frontend 2011)

**Font-stack**

Font-stack on suomeksi fonttipino. Fonttipino on tapa latoa fonttimääritykset järjestettyyn listaan tyylitiedostossa niin, että ensimmäisenä on ensisijainen fontti. Ensimmäisen fontin jälkeen fontit ovat järjestyksessä niin, että viimeiset fontit toimivat kaikissa käyttöjärjestelmissä ja selaimissa niin sanotusti “varmasti”. (Chapman 2009)

**F-Pattern**

F-Pattern on käyttäjän silmän liikettä tutkimalla muodostettu standardi käyttäjän käyttäytymisestä web-sisällön lukemisessa. (Nielsen 2006)

**GitHub**

GitHub on verkkosivusto, jonka tarkoituksena on helpottaa lähdekoodin jakamista. (GitHub 2011)

**Heuristinen**

Kokemukseen perustuva arviointi, jonka avulla etsitään käytettävyysongelmia sovelluksesta. (Nielsen 2005)

**HTML5**

HTML5 on Hyper Text Markup Languagen uusin spesifikaatio, joka tuo paljon uusia ominaisuuksia sisältäen muun muassa paremmin tarkoitustaan kuvaavat tagit, sivujen ja ikkunoiden välillä liikkumisen, animaatiot ja parannetun multimediatuen. (W3C 2011)

**JSON**

Lyhenne sanoista JavaScript Object Notation. Yksinkertainen tiedonsiirtomuoto, jota sekä ihmisen että tietokoneen on helppo lukea ja kirjoittaa. (JSON 2011)

**KISS**

Lyhenne sanoista “Keep It Simple, Stupid!” (KISS Principle 2001)

**Message Queue**

Menetelmän tarkoituksena on parantaa prosessin kontrollia lisäämällä ja poimimalla pinosta suoritettavia tehtäviä. Kaikki suoritettavat tehtävät odottavat pinossa kunnes ne suoritetaan. (Microsoft 2011)

**Middleware**

Väliohjelmistoiksi tai middleware-ohjelmistoiksi kutsutaan palvelinkoneella ajettavia ohjelmistoja, jotka toimivat selaimen ja palvelimella olevan sovelluksen välissä. (Puhakka 2004)

**MongoDB**

MongoDB on dokumenttipohjainen C++ kielellä kirjoitettu avoimen lähdekoodin tietokantajärjestelmä. (MongoDB 2011)

**MVC**

Model-View-Controller on sovellusarkkitehtuuri, joka erottaa sovelluslogiikan ja sovelluksen sisältämän datan käyttäjälle näytettävästä käyttöliittymästä. (MVC Architecture 2008)

**MySQL**

MySQL on suosituin vapaan lähdekoodin tietokanta. Sen etuihin kuuluu nopeus, luotettavuus ja sen käytön luvataan olevan helppoa. Toimii useilla eri alustoilla, kuten Linux, Windows ja Mac OS. (MySQL 2011)

**Observer**

Observer on tarkkailija, joka tilaa itselleen tiettyjä viestejä kohteelta. Observer-mallia käytetään kun halutaan jakaa kohteen tietoa monelle eri tarkkailijalle. Hyvä esimerkki on sääasemasovellus, jossa tarkkailtava kohde on lämpömittari ja tarkkailija on näyttö. (Freeman, Freeman, Sierra & Bates 2004, 56; Douglas 2003, 370–371)

**Pilvipalvelu**

Palvelu, joka skaalautuu asiakkaan tarpeiden ja käytön mukaan. Sen etuihin lukeutuu myös kustannustehokkuus ja se on nopea ottaa käyttöön. Julkiset pilvipalvelut vaativat aina internetyhteyden toimiakseen. (IBM 2011)

**Prototyyppi**

Prototyypiksi voidaan kutsua sovelluksen ensimmäistä testiversiota. Prototyyppi voi olla myös sovelluksen testiversio, jossa kokeillaan jotain uutta. (Hunt & Thoms 1999, 77)

**PHP**

PHP on komentosarjakieli, joka sopii hyvin web-sovellusten toteuttamiseen. PHP-koodia voidaan kirjoittaa HTML-koodin sekaan. (PHP 2011)

**Repository**

Repository on versionhallintajärjestelmässä oleva projektin tiedostojen säilytyspaikka. Käytämme jatkossa nimeä säilytyspaikka. (Swicegood 2008)

**Ruby on Rails**

MVC-mallin mukainen sovelluskehys, jota käytetään pääasiassa web-pohjaisten sovelluksien kehittämisessä. Käytämme jatkossa lyhennettä “Rails” sanasta “Ruby on Rails”. (Ruby on Rails Documentation 2011)

**Regressiotestaus**

Kun ohjelmistoon tehdään parannuksia tai muutoksia, jo aiemmin testattuja toimintoja testataan uudelleen sen varmistamiseksi, etteivät uudet ominaisuudet ole aiheuttaneet uusia ongelmia. (Kosonen 2011)

**Ruby**

Ruby on oliopohjainen ohjelmointikieli. (Thomas, Hunt & Fowler 2009, 13–15)

**Serialisointi**

Serialisoinnilla voidaan muuntaa luettavan datan tila sellaiseen muotoon, joka voidaan tallentaa tiedostoon, välimuistiin tai lähettää verkon yli. Datan tila voidaan myöhemmin palauttaa alkuperäiseen luettavaan muotoon. (McGowan 2010)

**Skype**

Skype on verkkopuheluihin erikoistunut ilmainen sovellus. Internetin kautta pystytään keskustelemaan ilmaiseksi ja maksusta voidaan soittaa puheluita yleisiin puhelinverkon puhelimiin. (Skype 2011)

**Sovelluskehys**

Sovelluskehys on runko tai alusta, jonka päälle sovellus rakennetaan. Sovelluskehys tarjoaa valmiita ohjelmakomponentteja, jotta varsinaisen sovelluksen tekeminen nopeutuisi. (Ruby ym. 2011)

**Symfony**

Symfony on PHP-pohjainen modulaarinen sovelluskehys. Symfony koostuu komponenteista, ja halutessaan voi jättää ylimääräiset toiminnallisuudet pois pienentääkseen sovelluksen kuormaa. (Symfony 2011)

**Syntaksi**

Syntaksi eli lauseoppi tarkastelee luonnollisen tai virallisen kielen merkkiyhdistelmiä. (Savolainen 2001)

**Tagi**

Tagi on termi, joka yleensä liitetään tietoon. Tageja käytetään tiedon kuvaamiseen ja niiden tarkoituksena on helpottaa tiedon etsimistä ja jaottelua. (Getting 2007)

**Type foundry**

“Fonttilähde”, Palvelu tai yritys, joka luo tekstityyppejä (fontteja) ja tarjoaa niitä muiden käytettäväksi. (Typeface Foundries 2011)

**Typografia**

Tarkoittaa nykyisin mitä tahansa tekstiin, tekstityyppiin, kirjainten asetteluun, väritykseen ja muuhun liittyvää suunnittelua ja korjausta. (Juselius 2004)

# 1 JOHDANTO

Työn tarkoituksena oli toteuttaa reaaliaikainen tapahtumakalenteri hyödyntäen nykyaikaisia menetelmiä, kuten ketterää ohjelmistokehitystä. Tapahtumakalenterin perimmäisenä ideana oli, että kaikki tapahtumat löytyisivät vaivattomasti yhdeltä sivustolta. Tapahtumien vaivaton haku oli sovelluksen tärkeimpiä osia. Käyttäjälle tarjotaan mahdollisimman laaja valikoima vaihtoehtoisia suodattamismahdollisuuksia, joilla löytää tapahtumia muun muassa kaupungin, maan, kategorian, viimeksi lisättyjen tapahtumien ja suosituimpien tapahtumien mukaan. Tapahtumakalenterissa käytettiin ulkoisten palveluntarjoajien kirjautumispalveluja, hyödyntämään käyttäjän jo olemassa olevia tunnuksia.

Tapahtumakalenteriin haluttiin tuoda mukaan reaaliaikaisuutta, jolloin vuorovaikutus käyttäjän kanssa saadaan syntymään mahdollisimman käyttäjäystävällisesti. Reaaliaikaisuuden mukaan ottamisella haluttiin erottua muista samankaltaisista sovelluksista. Reaaliaikaisuuden mukaan ottamisen syynä oli myös sen käytännöllisyys.

Kirjautuneet käyttäjät voivat keskustella tapahtumista kommentoimalla tapahtumia. Facebook- ja Google+-palveluja hyödynnettiin niiden “tykkää”-painikkeilla, joilla voidaan jakaa tietoa tapahtumista kyseisissä palveluissa. Suunnitteluvaiheessa tapahtumakalenteriin mietittiin ominaisuutta, joka voisi tuoda taloudellista tuottoa. Liikeidea muistuttaa hyvin pitkälle Googlen ja Facebookin mallia eli sivustolle tarjotaan mahdollisuutta ostaa mainostilaa. Tapahtumakalenterin käyttäjille tarjotaan mahdollisuus luoda asiakkuuksia ja asiakkuuksilla on mahdollisuus lisätä mainoksia. Mainoksen hinta määräytyy päivien ja tagien mukaan.

Projektin tarkoitus oli luoda konkreettinen käytännön sovellus hyödyntäen moderneja tekniikoita ja menetelmiä, kuten Node.js:ää, CoffeeScriptia ja Ruby on Railsia. Sovelluksen arkkitehtuuria ja ideaa suunniteltiin niin pitkälle, että sitä voidaan helposti jatkokehittää tulevaisuudessa.

Tapahtumakalenterin toteuttamisessa haluttiin kokeilla uusia menetelmiä, joista muun muassa HTML5 ja CSS3 ovat mielenkiintoisia tekniikoita kokeiltavaksi.

# 2 TAPAHTUMAKALENTERIN TOIMINTA

Tapahtumakalentereita on olemassa useampia, mutta halusimme tehdä meidän projektistamme hieman muista tapahtumakalentereista poikkeavan. Löysimme muutamia sisällöltään ja käytettävyydeltään hyviä sovelluksia. Eventful on monipuolisin tapahtumia sisältävä palvelu, joka vaikutti tarjoavan eniten sisältöä. City.fi:n Go -tapahtumapalvelun käytettävyys oli näppärää sen tapahtumasuodatin-toiminnallisuuden ansiosta, mutta sisältöä ei ollut yhtä paljon kuin Eventful-palvelussa, ainoastaan kotimaan tapahtumia. Halusimme yhdistää näiden kahden sovellusten hyvät ominaisuudet keskenään eli sisällön, selkeyden ja käytettävyyden.

Google+- ja Facebook-palveluista on tullut entistä reaaliaikaisempia. Käyttäjät näkevät muiden päivitykset lähes reaaliajassa. Idea reaaliaikaisesta tapahtumakalenterista lähtikin alun perin näistä sosiaalisista palveluista. Ajattelimme, että reaaliaikaisuutta voitaisiin hyödyntää ainakin etusivulla, jossa listattaisiin uusimpia tapahtumia. Havaitsimme heti muutamia ongelmia käytettävyyteen liittyen, jos tapahtumavirta olisi täysin reaaliaikainen: se vaikeuttaisi tapahtumalistan seuraamista, koska lista päivittyy ja mahdollisesti korvaa tai poistaa vanhan tiedon automaattisesti lukijan huomaamatta. Päätimme, että käyttäjällä voisi olla mahdollisuus kytkeä päälle täysin reaaliaikainen tila, jolloin uutta palvelimelta saapuvaa tietoa päivitetään automaattisesti käyttöliittymässä. Sovelluksen reaaliaikaisuustila olisi kuitenkin oletuksena manuaalinen, jolloin käyttäjä voisi halutessaan päivittää sisällön, kun uutta sisältöä on saatavilla. Ylläpitäjät kuitenkin aina saavat tiedon uusista tapahtumista reaaliaikaisesti käyttöliittymän kautta.

Pohdimme, kuinka saisimme tarpeeksi sisältöä sivuillemme, lisäävätkö käyttäjät itse vai haetaanko dataa jostain muualta. Tulimme kuitenkin siihen tulokseen, että käyttäjät ja yritykset saisivat itse lisätä tapahtumia sivustolle, eikä sisältöä tuotaisi mistään

muualta. Saisimme suunnitella tietokannan taulut juuri tapahtumakalenteriin sopiviksi. Tapahtumia ei viedä sosiaalisiin palveluihin niin, että samaa dataa olisi kahdessa paikassa.

## 2.1 Käyttöliittymä ja sovelluskehykset

Käyttöliittymästä haluttiin selkeä ja intuitiivinen, joka toimisi myös mobiililaitteilla. Käyttöliittymää mallinnettiin perinteisesti paperin ja kynän voimin, välillä myös iPadilla. Vaikutteita haettiin Google- ja Eventful-sivuilta. Halusimme Googlen kaltaisen matalan yläpalkin, jotta sisältö saisi riittävästi tilaa ja siis alkaisi sivustolla mahdollisimman korkealta. Tutkimme, mitä nykyaikaisia tekniikoita voisimme käyttää sovelluksen toteuttamiseen. Ruby on Rails oli aikaisemman kokemuksen vuoksi selkeä valinta sovelluskehykseksi. Ruby on Railsista julkaistiin projektimme alussa uusi versio, jonka mukana tuli uusia ominaisuuksia. Muun muassa SCSS, CoffeeScript ja Asset Pipeline (Hansson 2011). Oli itsestään selvää, että käyttäisimme kyseisiä tekniikoita, koska ne nopeuttaisivat ohjelmointia ja vähentäisivät fyysisen koodin määrää.

Reaaliaikaisia toimintoja varten päätimme käyttää erillistä sovelluskehystä. Railsia oli ajateltu käyttää lähinnä front-endissä eli käyttöliittymäpuolella ja tietokanta-operaatioihin. Suorituskyky olisi luultavasti kärsinyt, jos reaaliaikaiset toiminnot olisi tehty myös Railsissa. Rails sisältää middlewareja eli väliohjelmia, jotka suoritetaan jokaisen web-pyynnön saapuessa palvelimelle ennen kuin Rails pystyy käsittelemään pyynnön, eli jonkin verran viivettä on havaittavissa ennen kuin käyttäjä saa vastauksen palvelimelta (Rails on Rack 2011). Teimme päätöksen hyödyntää Node.js-sovelluskehystä reaaliaikaisen palvelinsovelluksen ympärillä. Valitsimme Node.js:n, koska totesimme sen suorituskyvyn olevan paljon parempi kuin Railsin. Node.js on uusi ja nopeasti yleistyvä tekniikka tällä hetkellä ja sen käyttö on vaivaton oppia, koska sen ohjelmat kirjoitetaan JavaScript-ohjelmointikielellä. Me kuitenkin ajattelimme käyttää projektissa CoffeeScript-ohjemointikieltä JavaScript-ohjelmointikielen sijaan.

Käyttäjä paikannetaan selaimen Geolocation-rajapinnan avulla tai IP:n perusteella. Ensisijaisesti käytetään selaimen paikannusta eli Geolocationia, koska se antaa tarkemman tuloksen kuin IP. Geolocation antaa sijainnin katutarkkuudella, kun taas IP-osoitteeseen perustuva paikannus pystyy kertomaan sijainnin vain kaupungin tarkkuudella. (IP2Location 2011)

Geolocation-rajapinnan avulla saadaan käyttäjän sijainti koordinaatteina (Geolocation API Specification 2010). Koordinaatit ovat käytössä vain selaimessa, JavaScript-koodissa, muuttujaan tallennettuna. Koordinaatit saadaan lähetettyä Rails-sovellukselle lisäämällä koordinaattiarvot jonkin HTML-elementin arvoksi JavaScript-koodilla osaksi näkymää.

## 2.2 Toimintaympäristö

Sovelluksemme toimintaympäristöiksi määriteltiin Linux ja Mac OS X. Node.js:stä ei ole vielä julkaistu toimivaa Windows-versiota, ja Ruby-ohjelmointikieli on ollut kokemuksiemme perusteella huomattavan hidas Windows-ympäristössä, joten jouduimme jättämään Windowsin pois toimintaympäristöjen listalta. Linux ja OS X olivat hyvät valinnat ympäristöiksi, sillä projektimme kaikilla jäsenillä oli jo ennestään kokemusta Unix-sukuisista järjestelmistä.

Ensisijaiseksi tietokantajärjestelmäksi valittiin MySQL, mutta kehitysympäristössä päätimme käyttää SQLite-tietokantajärjestelmää, joka tuleekin oletuksena Railsin mukana. Rails on riippumaton tietokantajärjestelmästä, eli periaatteessa olisimme voineet käyttää mitä tahansa tietokantajärjestelmää sekä tuotepalvelimella että kehityksessä. Ajattelimme käyttää MongoDB-tietokantajärjestelmää Node.js-sovelluksessä.

## 2.3 Projektityökalut ja tiimityö

Pääasiallisiksi projektinhallintamenetelmiksi valittiin Scrum ja XP, lisäksi käytössä oli muun muassa BDD, DRY, KISS ja MVC. Scrum oli entuudestaan tuttu projektinhallintamenetelmä aikaisemmista projekteista, ja siksi se tuntui luontevalta valinnalta. XP otettiin käyttöön, koska se tukisi ryhmätyöskentelyä. XP oli menetelmänä kaikille käytännössä tuntematon, mutta halusimme tutustua siihen konkreettisesti.

Opinnäytetyö aikataulutettiin niin, että ensimmäisenä toteutetaan sovellus ja raportti kirjoitetaan sen pohjalta. Näin saisimme keskittyä täysin sovelluksen toteutukseen, jonka jälkeen tiedetään, mitä kaikkea raportti tulee sisältämään. Ensimmäisenä prosessina listattiin sovelluksen vaatimukset, josta seurasi toinen prosessi, käyttötapausten kuvaus. Käyttöliittymän, tietokannan ja arkkitehtuurin mallinnusprosessi aloitettiin rinnakkain. Tavoitteena oli, että jokainen prosessi saataisiin vietyä loppuun saakka korkeintaan kahdessa viikossa. Jos kuitenkin vaikutti siltä, että prosessi kuluttaa enemmän aikaa kuin oli arvioitu, aloitimme toisen prosessin edellisen ohella kuluttamatta liikaa aikaa. Teknisen toteutuksen loppuun määriteltiin raportti-, lopputestaus-, ylläpito- ja käyttöönottoprosessit, joista ylläpitoprosessi jatkui koko sovelluksemme elinajan.

Työnjako tehtiin henkilökohtaisten osaamisalueiden ja kiinnostusten mukaan. MVC-mallin ansiosta tehtävät saatiin eristettyä selvästi toisistaan, esimerkiksi yksi teki näkymiä, toinen ohjaimia ja kolmas tietokannan malleja. MVC-malli vähensi huomattavasti tiedostojen konfliktien eli ristiriitaisuuksien määrää.

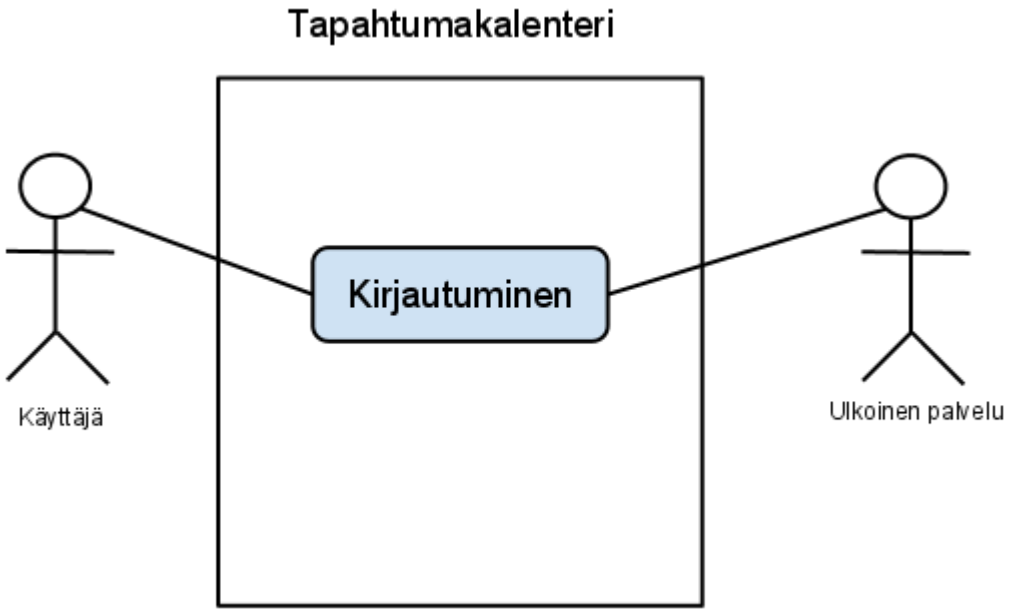
## 2.4 Käyttötapaukset

Käyttötapauksilla kuvataan käyttäjän tai toimijan ja sovelluksen välistä vuorovaikutusta, jonka avulla käyttäjä saavuttaa jotakin hyödyllistä. Käyttäjä tai toimija voi olla henkilö tai palvelu. Käyttötapaukset ovat sovelluksen mallintamisessa käytettävä tekniikka, jolla voidaan havainnollistaa toteutettavia ominaisuuksia. Hyvin kirjoitettu tai kuvattu käyttötapaus on helppo lukea ja ymmärtää. Käyttötapauskaaviosta saadaan

yleiskuva sovelluksesta ja sen toiminnallisuudesta. Sen avulla on helppo käydä keskustelua loppukäyttäjän kanssa. (Cockburn 2001, 1–3)

### 2.4.1 Käyttäjän kirjautuminen tapahtumakalenteriin

Tapahtumakalenteriin ei toteutettu käyttäjien rekisteröintimahdollisuutta. Haluttiin välttää uusien tunnusten luomista käyttäjille, koska käyttäjät luultavasti omistavat ennestään useampia tunnuksia sosiaalisiin palveluihin. Käyttäjän on luotava käyttäjätili johonkin sovelluksen tarjoamista palveluista, mikäli hänellä ei sellaista jo ole. Sovelluksessa ohjataan käyttäjää luomaan MyOpenID-tunnus tai GoogleID-tunnus, joita voidaan käyttää monissa muissakin palveluissa. Käyttäjä voi halutessaan myös luoda tilin johonkin muista tarjottavista palveluista. Kuviossa 1 on esitetty käyttötapauskaavio käyttäjän kirjautumisesta sovellukseen kirjautumispalvelun tarjoajan välityksellä.




**Kuvio 1.** Käyttäjän kirjautuminen sovellukseen

**Kirjautuminen**

**Päätoimija:** Käyttäjä

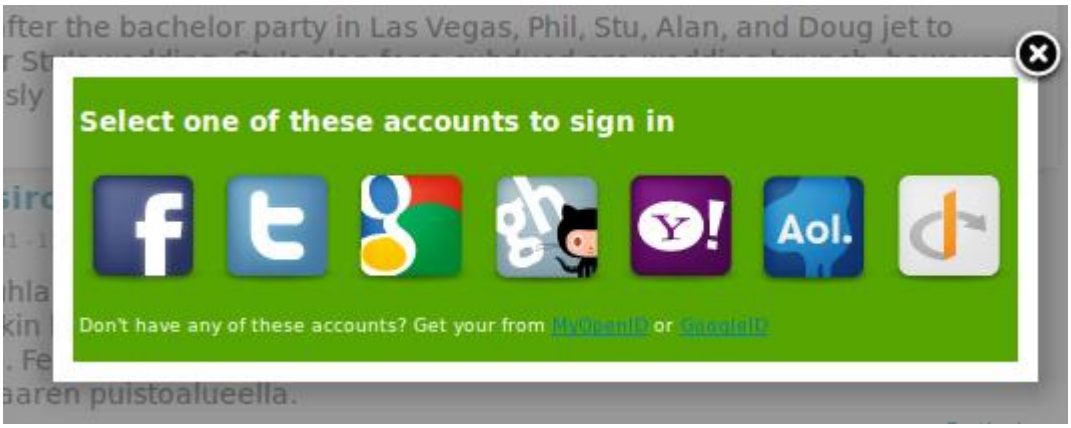
**Tehtävät:** Käyttäjä haluaa kirjautua tapahtumakalenteriin. Kirjautumispalvelun tarjoaja tarkistaa käyttäjän tunnuksen ja salasanan.

**Esiehto:** Käyttäjä on tapahtumakalenterin etusivulla.

**Tavoite:** Käyttäjällä on oltava voimassaoleva tunnus jossakin tarjottavassa palvelussa. Palveluntarjoajan on tunnistettava käyttäjä kirjautumistietojen perusteella.

**Skenaario:**

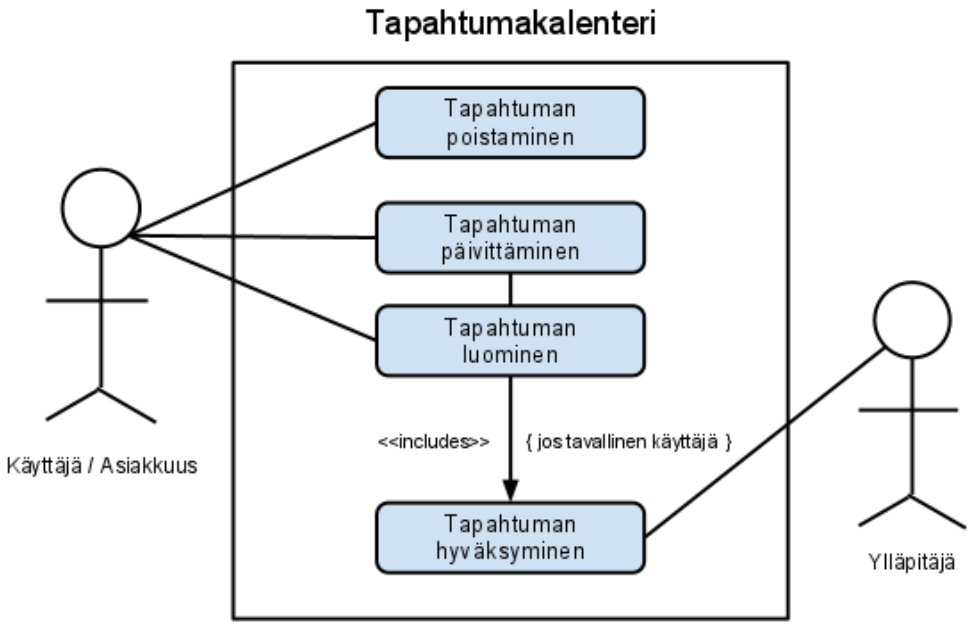
1. Käyttäjä napsauttaa kirjaudu sisään -linkkiä.
2. Käyttäjälle aukeaa kuvion 2 mukainen ikkuna, jossa näytetään kirjautumisvaihtoehdot.
3. Käyttäjä napsauttaa haluamaansa palveluntarjoajaa.
4. Sovellus ohjaa käyttäjän palveluntarjoajan sivustolle.
5. 
    a. Käyttäjä syöttää käyttäjätunnuksen ja salasanan tunnistusta varten.
    b. Käyttäjä ei omista tunnuksia palveluun:
        i. Käyttäjä luo kyseisellä sivulla itselleen tunnuksen.
        ii. Käyttäjä syöttää käyttäjätunnuksen ja salasanan tunnistusta varten.
6. Palveluntarjoaja ohjaa tunnistuksen jälkeen käyttäjän takaisin tapahtumakalenterin etusivulle.
7. Tapahtumakalenteri ilmoittaa käyttäjälle, että kirjautuminen onnistui.




**Kuvio 2.** Näkymä kirjautumissivusta

### 2.4.2 Tapahtuman lisääminen

Tapahtumakalenterin tärkeimpänä ominaisuutena pidetään tapahtumien lisäämistä, sillä tapahtumakalenterin toiminta perustuu siihen, että se sisältää paljon tapahtumia. Tapahtumia voivat lisätä sekä niin sanotut “tavalliset käyttäjät” että käyttäjät, jotka kuuluvat jollekin asiakkuudelle. Näiden kahden erona on se, että asiakkuuden ilmoittamat tapahtumat eivät mene ylläpitäjälle tarkistettavaksi, vaan ne julkaistaan heti. Asiakkuudet ovat luotettuja tapahtumien lisääjiä, joiden uskotaan ilmoittavan asiallisista tapahtumista. Ylläpito saa asiakkuuksien lisäämistä ja muokkaamista tapahtumista sähköpostiviestin. Ylläpito voi myös seurata asiakkuuksien lisäämiä tai muokkaamia tapahtumia mahdollisten asiattomuuksien varalta. Käyttöliittymässä näytetään ylläpidolle vain tarkistettavaksi menevät tapahtumat, jotta tavallisten käyttäjien ja asiakkuuksien ilmoittamat tapahtumat saadaan pidettyä selvemmin erillään toisistaan. Kuviossa 3 on esitetty käyttötapauskaavio tapahtumien hallinnasta.




**Kuvio 3.** Tapahtumien hallinta käyttäjän ja ylläpitäjän näkökulmasta

**Tapahtuman lisääminen**

**Päätoimija:** Käyttäjä

**Tehtävät:** Käyttäjä lisää uuden tapahtuman tapahtumakalenteriin.

**Esiehto:** Käyttäjän tulee ensin olla kirjautuneena tapahtumakalenteriin (katso käyttäjän kirjautuminen tapahtumakalenteriin).

**Tavoite:** Käyttäjän tulee täyttää kaikki lomakkeen vaatimat kentät.

**Skenaario:**

1. Käyttäjä napsauttaa tapahtumakalenterin etusivulla ”lisää tapahtuma”-linkkiä.
2. Käyttäjälle aukeaa lomake-näkymä.
3. Käyttäjä antaa tapahtumalle nimen.
4. Käyttäjä kirjoittaa tapahtumalle kuvauksen. Siitä tulee käydä ilmi tapahtuman hinta, mikäli kyseessä on maksullinen tapahtuma.

5. Käyttäjä valitsee tapahtumalle sopivat kategoriat napsauttamalla kategorian nimen vieressä olevaa laatikkoa. Käyttäjä voi siis valita useamman kuin yhden kategorian. Ylläpitäjä tarkistaa, ettei tapahtumaa ole laitettu kategoriaan, johon se ei ehdottomasti kuulu.

6. Käyttäjä valitsee kalenterista tapahtuman alkamisajan ja mahdollisesti päättymisajan, mikäli tapahtuman on tarkoitus päättyä tiettyyn aikaan mennessä. Kalenteri tulee näkyville, kun napsauttaa teksti-kenttää. Ajat voi myös syöttää käsin muodossa "YYYY-mm-dd HH:mm". Mikäli tapahtumalla on useampia esitysaikoja, niin käyttäjä napsauttaa "lisää esitysaika"-linkkiä, jolloin näytetään uudet kentät alkamisajalle ja päättymisajalle. Uusia kenttiä koskevat samat säännöt kuin edellisen esitysajan kenttiä.

7. Käyttäjä voi halutessaan lisätä tapahtumalle kuvan "Tapahtuman kuva"-otsikon alla olevasta "Choose file"-painikkeesta. Kuvia voi myös lisätä myöhemmin tapahtuman galleriaan.

8. Käyttäjä valitsee alasvetovalikosta maan, missä tapahtuma sijaitsee.

9. Käyttäjä syöttää tapahtumapaikan katuosoitteen ja kaupungin pilkulla eroteltuna. Tapahtumapaikka näytetään tapahtumasivulla Google Maps -karttana.

10. Käyttäjän tulee asettaa tapahtumalle julkaisuaika ja piilotusaika kalenterista. Julkaisuaika määrittelee milloin tapahtuma tulee kaikkien nähtäville ja piilotusaika taas määrittelee milloin tapahtuma ei ole enää nähtävillä. Tapahtumaa ei kuitenkaan poisteta tietokannasta. Julkaisuajan ja piilotusajan voi myös syöttää käsin samassa muodossa kuin esitysajat, "YYYY-mm-dd HH:mm".

11. Käyttäjä painaa Tallenna-painiketta.

12.

    a. Käyttäjää pyydetään valitsemaan lisäämästään tapahtuman kuvasta haluamansa kohdan.

    b. Käyttäjä ei ole lisännyt tapahtumalle kuvaa.

        i. Käyttäjä siirtyy kohtaan 14.

13. Käyttäjä painaa "Crop"-painiketta kun valinta on tehty.

14.

    a. Lisääjänä toimii "tavallinen käyttäjä".

i. Tapahtumakalenteri ohjaa käyttäjän tapahtumasivulle ja ilmoittaa, että tapahtuma on lähetetty ylläpidolle tarkistettavaksi. Lisäksi käyttäjälle kerrotaan sähköpostitse, että tapahtuma on tarkistuksessa.

b. Lisääjänä toimii asiakkuus.

i. Tapahtumakalenteri ohjaa käyttäjän tapahtumasivulle ja ilmoittaa, että tapahtuma on lisätty. Lisäksi käyttäjä saa sähköpostiviestin, josta käy ilmi, että tapahtuma on julkaistu.

### 2.4.3 Tapahtuman hyväksyminen

Tapahtumakalenterin ylläpitäjän rooli on tärkeä siinä mielessä, että tapahtumat pysyvät asiallisina ja niiden kuvaukset ovat paikkansapitäviä ja riittävän tarkkoja. Tapahtuman hyväksymistä ylläpitäjän näkökulmasta on hyvä selkeyttää myös käyttäjälle. Tapahtuman hyväksyminen selviää kuviosta 3.

**Tapahtuman hyväksyminen**

**Päätoimija:** Ylläpitäjä

**Tehtävät:** Ylläpitäjä tarkistaa käyttäjän lisäämän tapahtuman ja joko hyväksyy tapahtuman tai pyytää asiakasta muokkaamaan tapahtumaa. Selvästi asiattomat tapahtumat hylätään ylläpitäjän toimesta.

**Esiehto:** Ylläpitäjän tulee olla kirjautuneena tapahtumakalenteriin.

**Tavoite:** Ylläpitäjän täytyy oman harkintakykynsä mukaan joko hyväksyä tai hylätä tapahtuma.

**Skenaario:**

1. Ylläpitäjä napsauttaa hallintapaneelin vahvistuspyynnöt-otsikon alta tapahtumat-linkkiä.

2. Tapahtumakalenteri listaa kaikki tapahtumat, joita ei ole vielä hyväksytty, vanhimmasta uusimpaan.

3.

    a. Ylläpitäjä voi jo tässä vaiheessa hyväksyä tapahtuman sen nimen ja kuvauksen perusteella, jotka ovat näkyvillä.

    b. Ylläpitäjä napsauttaa ensimmäisenä olevan tapahtuman nimeä.

4. Tapahtuman näkymä aukeaa ylläpitäjälle.

5. Ylläpitäjä tarkistaa tapahtuman asiallisuuden.

6.

    a. Ylläpitäjä hyväksyy tapahtuman.

        i. Tapahtuma julkaistaan kaikkien nähtäville.

        ii. Käyttäjä saa sähköpostiviestin, josta käy ilmi, että tapahtuma on julkaistu.

        iii. Tapahtuma tulee näkyville käyttäjän omiin tapahtumiin.

    b. Ylläpitäjä ei hyväksy tapahtumaa.

        i. Ylläpitäjä pyytää sähköpostitse käyttäjää tekemään muutoksia tapahtumaan.

        ii. Käyttäjä käy muokkaamassa tapahtumaa ylläpitäjän tekemien huomautusten mukaisesti.

        iii. Palataan kohtaan 5.

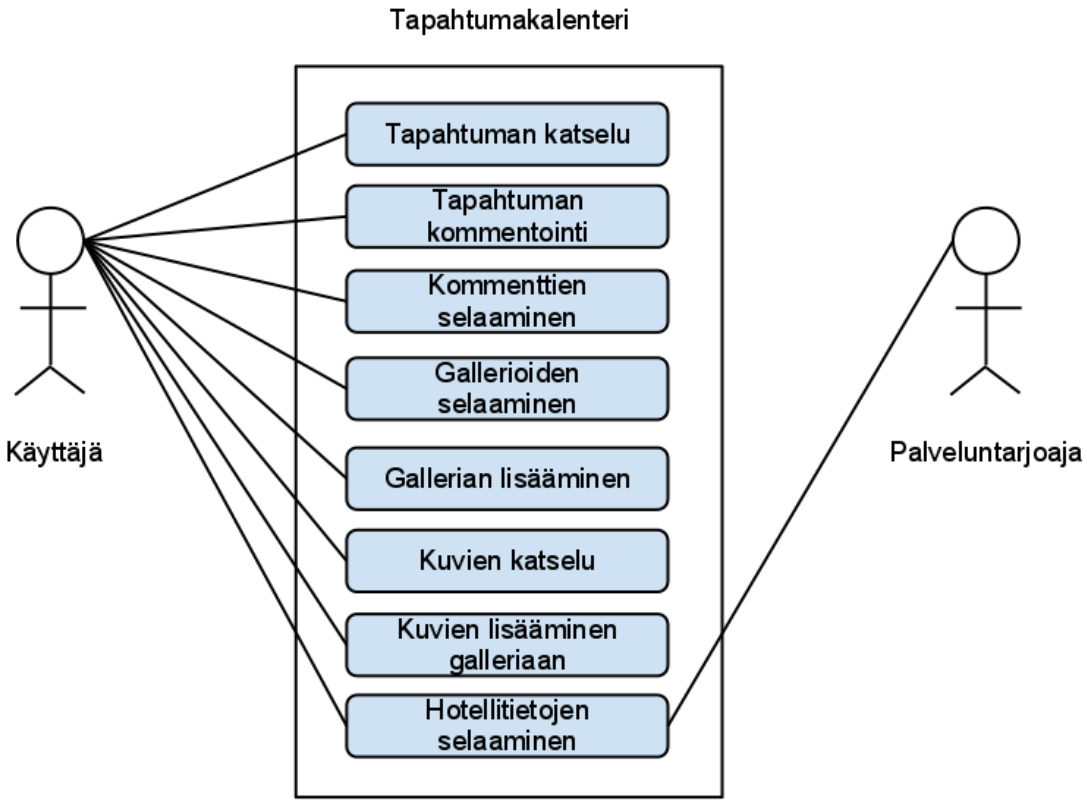
    c. Ylläpitäjä hylkää tapahtuman.

        i. Käyttäjä saa sähköpostiviestin, josta käy ilmi, että tapahtuma on hylätty, koska tapahtuma on ylläpitäjän toimesta havaittu selvästi asiattomaksi.

### 2.4.4 Tapahtuman tarkastelu

Käyttäjät voivat selata tapahtumia listana, jolloin tapahtumista näytetään vain tärkeimmät tiedot. Tarkempaa tietoa tapahtumasta saadaan avaamalla tapahtumanäkymä. Tapahtumasivulla käyttäjä näkee ensimmäisenä tapahtuman tietoja, muun muassa

kuvauksen, esitysajat ja tapahtumapaikan. Kuviossa 4 on käyttötapauskaavio tapahtuman tarkastelusta.




**Kuvio 4.** Tapahtuman tarkastelu.

**Tapahtuman tarkastelu**

**Päätoimija:** Käyttäjä

**Tehtävät:** Käyttäjä saa selville tapahtuman tiedot. Käyttäjä voi kommentoida ja lukea muiden käyttäjien kommentteja tapahtumasta. Käyttäjä voi katsella tapahtumien kuvia. Käyttäjä voi lisätä omiin tapahtumiin gallerioita ja lisätä niihin kuvia tapahtumasta. Käyttäjä voi varata hotellin tapahtumakaupungista.

**Esiehto:** Käyttäjä on tapahtumakalenteri-sovelluksessa.

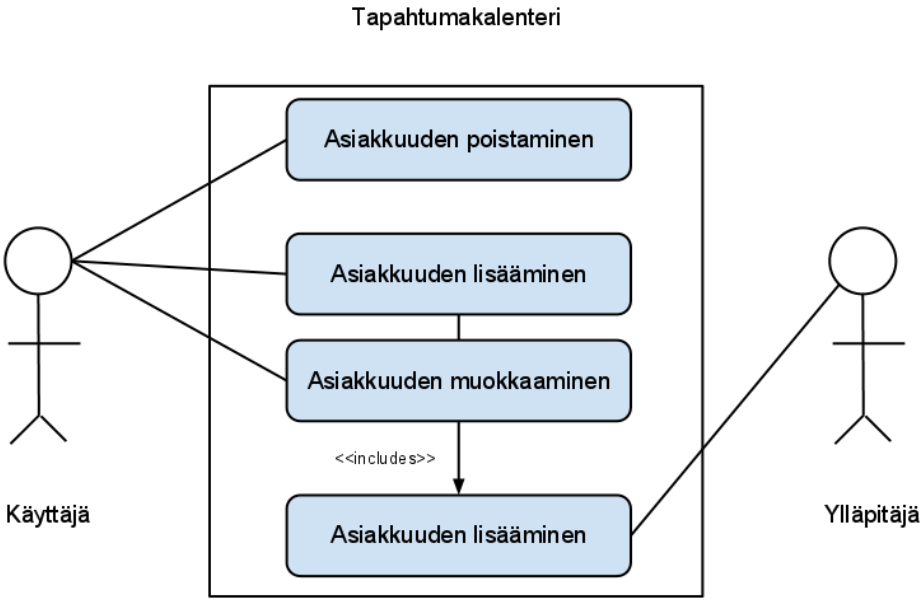
**Tavoite:** Tapahtumakalenterissa täytyy olla ainakin yksi tapahtuma.

**Skenaario:**

1. Käyttäjä napsauttaa tapahtumaa.
2. Käyttäjälle aukeaa tapahtumasivu.
3. Käyttäjälle näytetään tapahtuman info-välilehti, josta käy ilmi tapahtuman nimi, kuvaus, tapahtuman kuva, esitysajat, sosiaalisen median toiminnot, tapahtumapaikka ja tapahtuman kategoriat.
4. Käyttäjä voi valita napsauttamalla mille välilehdelle siirtyy:
    a. Kommentit
        i. Käyttäjä voi lukea muiden käyttäjien kommentteja.
        ii. Kirjautunut käyttäjä voi:
            1. kommentoida tapahtumaa,
            2. muokata kirjoittamiaan kommentteja,
            3. poistaa kirjoittamiaan kommentteja.
    b. Hotellit
        i. Käyttäjä napsauttaa ”Hotels.com”-linkkiä, joka siirtää käyttäjän Hotels.com-sivustolle ja näyttää tapahtumapaikkakunnan vapaat hotellit.
    c. Galleria
        i. Käyttäjä voi selailla gallerioita ja niissä olevia kuvia.
        ii. Kirjautunut käyttäjä voi:
            1. lisätä omiin tapahtumiin gallerioita ja kuvia,
            2. muokata omien tapahtumien gallerioita ja kuvia,
            3. poistaa omien tapahtumien gallerioita ja kuvia.
    d. Info
        i. Käyttäjä näkee tapahtuman tiedot.

### 2.4.5 Asiakkuuden lisääminen

Tapahtumakalenterissa on mahdollista luoda asiakkuus. Asiakkuudella tarkoitetaan yritystä, organisaatiota tai instituuttia. Asiakkuuksien avulla saadaan tapahtumakalenteriin tunnettuja tapahtumia ja paikallisten yritysten järjestämiä tapahtumia, esimerkiksi elokuvateatterin elokuvatarjonta. Ylläpitäjä tarkistaa ja hyväksyy asiakkuuden ennen kuin se julkaistaan. Asiakkuuden luomisessa täytetään vaadittavat kentät, jonka jälkeen lomake tallennetaan, ja asiakkuus menee ylläpidolle hyväksyttäväksi. Ylläpito joko hyväksyy, pyytää tekemään muutoksia tai hylkää asiakkuuden. Asiakkuuden hyväksyminen tapahtuu myös ”vahvistuspyynnöt”-otsikon alta samalla tavoin kuin tapahtuman hyväksyminen. Kuviossa 5 on käyttötapauskaavio asiakkuuden lisäämisestä.




**Kuvio 5.** Asiakkuuden lisääminen

**Asiakkuuden lisääminen**

**Päätoimija:** Käyttäjä

**Tehtävät:** Käyttäjä luo asiakkuuden.

**Esiehto:** Käyttäjä on kirjautuneena tapahtumakalenteriin.

**Tavoite:** Käyttäjän täytyy täyttää kaikki lomakkeen kentät.

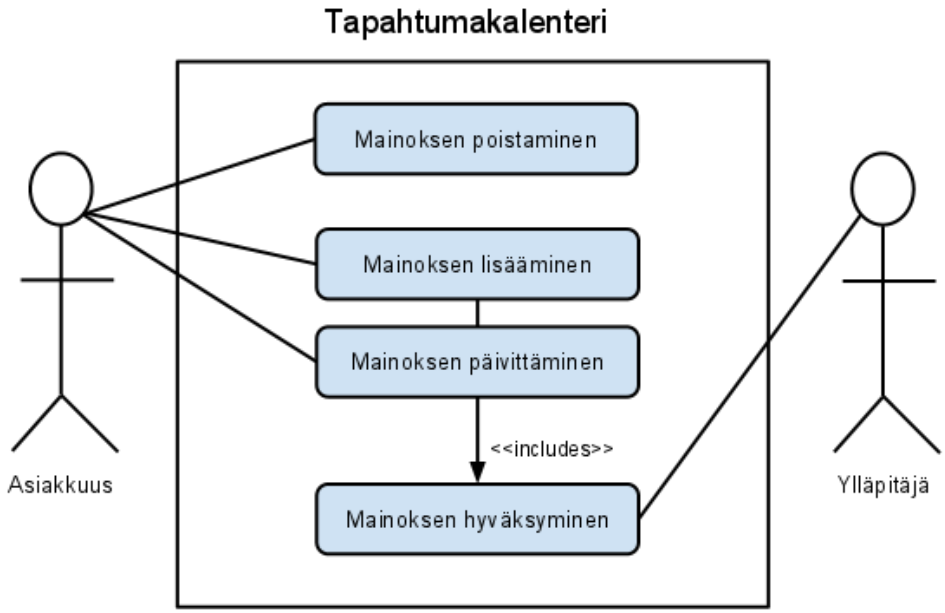
**Skenaario:**

1. Käyttäjä napsauttaa ”Omat linkit”-otsikon alta ”Uusi asiakkuus”-linkkiä.
2. Käyttäjälle aukeaa uuden asiakkuuden lisäämiseen tarkoitettu lomake.
3. Käyttäjä kirjoittaa asiakkuuden nimen.
4. Käyttäjä kirjoittaa asiakkuuden kuvauksen.
5. Käyttäjä valitsee alasvetovalikosta asiakkuuden tyypin.
6. Käyttäjä kirjoittaa asiakkuuden osoitteen.
7. Käyttäjä valitsee alasvetovalikosta asiakkuuden maan.
8. Käyttäjä napsauttaa ”Tallenna”-painiketta, jolloin asiakkuus menee ylläpidon tarkastettavaksi.

### 2.4.6 Mainoksen lisääminen

Asiakkuudet voivat lisätä tapahtumakalenteriin omia mainoksiaan. Mainoksista on tehty maksullinen palvelu, jolla halutaan antaa asiakkuuksille mahdollisuus mainostaa itseään ja kerätä tapahtumakalenterille varoja jatkokehitystä varten. Vain asiakkuuksilla on mahdollisuus lisätä mainoksia. Kun asiakkuus on lisännyt mainoksen, niin vain ylläpitäjä voi sitä enää muokata. Asiakkuus voi lisätä haluamansa määrän tageja mainokseen ja valita, kuinka monta päivää mainos on näkyvillä. Mitä enemmän lisätään tageja mainokseen, niin sen näkyvämpi siitä tulee tapahtumakalenterin eri näkymissä. Mainoksen hinta määräytyy päivien lukumäärän ja tagien määrän mu-

kaan. Hinta päivittyy ruudulla automaattisesti, kun asiakkuus lisää mainokselle päiviä ja tageja. Yksi päivä mainoksen esitysaikaa maksaa viisi euroa, myös tagin hinnaksi on määritelty viisi euroa. Mainoksen hyväksyminen tapahtuu myös vahvistuspyynnöt otsikon alta samalla tavoin kuin tapahtuman tai asiakkuuden hyväksyminen. Kuviossa 6 on käyttötapauskaavio mainoksen lisäämisestä.




**Kuvio 6.** Mainoksen hallinta

**Mainoksen lisääminen**

**Päätoimija:** Asiakkuus

**Tehtävät:** Asiakkuus lisää mainoksen.

**Esiehto:** Käyttäjän on täytynyt luoda asiakkuus. Käyttäjä on kirjautuneena tapahtumakalenteriin.

**Tavoite:** Käyttäjän täytyy täyttää kaikki lomakkeen kentät.

**Skenaario:**

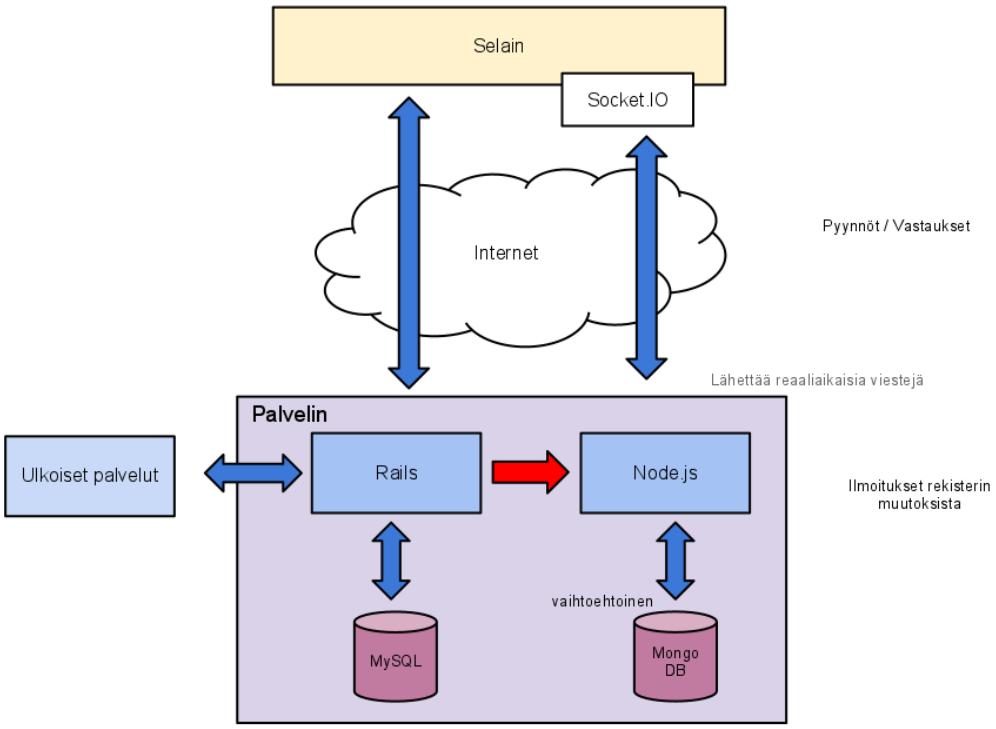
1. Asiakkuus napsauttaa ”Omat linkit”-otsikon alta “Uusi mainos”-linkkiä.
2. Asiakkuudelle aukeaa uuden mainoksen lisäämiseen tarkoitettu lomake.
3. Asiakkuus valitsee liukuvalitsimella, montako päivää mainos on näkyvissä.
4. Asiakkuus kirjoittaa ”Tagit”-otsikon alle halutut tagit pilkulla eroteltuna.
5. Asiakkuus valitsee mainoksen alkamisajankohdan kalenterista, joka aukeaa, kun napsauttaa alkamisajan teksti-kenttää.
6. Asiakkuus valitsee omista tiedostoistaan mainokselle kuvan napsauttamalla “Choose file”-painiketta.
7. Asiakkuus napsauttaa” Tallenna”-painiketta, jolloin mainos menee ylläpidon tarkastettavaksi.

## 2.5 Arkkitehtuuri

Tapahtumakalenterissa toimii kaksi sovellusta ja selain. Selain on keskeisessä osassa tapahtumakalenterin toiminnan kannalta. Selaimen kautta ohjataan tapahtumakalenterin toimintaa, ja sen lisäksi selaimessa nähdään sovellusten tuottamat tulokset. Ruby on Rails -sovellus on ensimmäisenä selaimen pyyntöä vastassa. Jos pyyntö on tavallinen lukutoiminta, niin Ruby on Rails -sovelluksesta palautetaan pelkkä näkymä. Node.js-sovellusta huomautetaan operaation jälkeen, mikäli pyyntöä seurasi tietokanta-operaatio. Node.js-sovellus on vastuussa tiedon välittämisestä selaimille, saatuaan Ruby on Rails -sovellukselta huomautuksen muuttuneesta tiedosta.

Kuviosta 7 voidaan nähdä, että selain lähettää ja vastaanottaa pyyntöjä. Kun käyttäjä tulee sivulle, selain lähettää pyynnön Ruby on Rails -sovellukselle, joka puolestaan lähettää näkymän selaimelle. Selain saa näkymän mukana tyyli- ja komentosarjatiedostot. Rails lähettää kaikki komentotiedostot selaimelle lukuun ottamatta Socket.IO-rajapintaa. Socket.IO-rajapinta ladataan Node.js-sovelluksesta sen jälkeen, kun selain on vastaanottanut näkymän. Kun selain on ladannut Socket.IO-rajapinnan, niin se

muodostaa pysyvän yhteyden Node.js-sovellukseen Socket.IO-rajapinnan tarjoamien palveluiden avulla. Tämän jälkeen selain on valmis vastaanottamaan viestejä Node.js-sovellukselta.




**Kuvio 7.** Yleiskuva tapahtumakalenterin arkkitehtuurista.

### 2.5.1 Ulkoiset palvelut ja Rails

Rails-pohjainen sovellus toimii näkymien jakajana ja ohjaimena tietokantaoperaatioihin MVC-mallin mukaisesti. Rails-sovelluksessa ylläpidetään myös istuntoja. Käyttäjillä on mahdollisuus kirjautua sovellukseemme jollain ulkoisella tilillä, esimerkiksi Facebook- tai Google-tileillä istuntotyyppisesti. Tapahtumakalenteri rekisteröitiin edellä mainittujen sivustojen rajapintapalveluiden käyttäjäksi, jotta ulkoisten palve-

luiden käyttö tapahtumakalenterissa olisi mahdollista. Rekisteröitymisen jälkeen saimme kummastakin palvelusta merkkijono muotoisen avaimen, jolla ulkoinen palvelu tunnistaa tapahtumakalenterin. Kun käyttäjä kirjautuu sisään, hän valitsee kirjautumistilin, ja pyyntö tapahtumasta lähetetään tapahtumakalenterille. Rails-sovelluksen sisällä rakennetaan uusi pyyntö ulkoiselle palvelulle käyttäjän valitseman tilin mukaan. Sisäiseen pyyntöön liitetään käyttäjän valitseman kirjautumispalvelun avain ja tapahtumakalenterin osoite. Kun pyyntö on suoritettu ulkoisessa palvelussa, selain ohjataan tapahtumakalenteriin.

Onnistuneen kirjautumisen jälkeen tietokantaan tallennetaan käyttäjän valitseman kirjautumispalvelun tiedot. Tällä tavalla käyttäjä on tunnistettavissa, esimerkiksi kommenteissa. Käyttäjän ei aina ole pakko valita samaa kirjautumispalvelua, vaan sama käyttäjä voi kirjautua sovellukseen usealla eri tilillä.

### 2.5.2 Sovelluskehysten välinen kommunikointi

Kuviossa 7 Railsin ja Node.js:n välinen nuoli tarkoittaa, että kahden sovelluksen välillä on silta, ja data kulkee siltaa pitkin vasemmalta oikealle eli Rails-sovelluksesta Node.js-sovellukseen. Kuljetettava data sisältää tiedon siitä, mitä on tapahtunut Rails-sovelluksessa, esimerkiksi “tapahtuma on lisätty” tai “tapahtuma on päivitetty”. Rails-sovelluksessa tietokantaoperaatiot saadaan kiinni käyttämällä Railsin Active-Record Callback -kutsuja, kuten after_destroy, after_create ja after_update (Ruby, Thomas & Hansson 2011, 289–290). Näiden kutsujen sisällä lähetämme datan siltaa pitkin Node.js-sovellukselle. Koodiesimerkissä 1 on Callback-kutsujen ja tiedon lähetyksen toteutus tapahtumakalenterissa.

```
require 'nodejs_notifier'

class Event < ActiveRecord::Base
  after_create :notify_created
  after_update :notify_updated
  after_destroy :notify_removed

  private
    def notify_updated
      NodejsNotifier.notify('/event_updated', to_json)
    end

    def notify_created
      NodejsNotifier.notify('/event_added', to_json)
    end

    def notify_removed
      NodejsNotifier.notify('/event_removed', to_json)
    end
end
```

**Koodiesimerkki 1.** Callback-kutsut ja datan lähetys siltaa pitkin.

Silta Rails-sovelluksessa on tavallinen luokka, jolla on yksi metodi. Tämän metodin avulla lähetetään HTTP POST -pyyntö Node.js-sovellukselle. Metodi saa kaksi parametria, joista ensimmäinen kertoo, mihin osoitteeseen pyyntö Node.js-sovelluksessa lähetetään ja toinen parametri sisältää lähetettävän datan. POST-pyyntöön sisällytetään muuttuneen tietueen data JSON-muodossa. Jokainen datamalliluokka perii metodin to_json “ActiveRecord::Base”-luokalta. Sen ansiosta tietueen data voidaan serialisoida, ja data saadaan kätevästi esitettyä JSON-muodossa (Thomas, Fowler & Hunt 2009, 420–421; to_json 2008). Node.js-sovellusta ajetaan Rails-sovelluksen kanssa samalla palvelimella, koska silloin tiedon lähetys on nopeaa eikä viivettä ole juuri lainkaan. Koodiesimerkki 2 kuvaa, kuinka silta on toteutettu Railsin ja Node.js:n välillä.

```ruby
require 'net/http'

class NodejsNotifier
  HOST = 'localhost'
  PORT = 1337

  def self.notify(url, params)
    request = Net::HTTP::Post.new(url, 'Content-Type' => 'application/json')
    request.body = params

    begin
      Net::HTTP.new(HOST, PORT).start { |http| http.request(request) }
    rescue
      Rails.logger.debug("Node.js sovellus on alhaalla")
    end
  end
end
```

**Koodiesimerkki 2.** NodejsNotifierin eli sillan implementaatio.

Node.js-sovelluksessa käytetään Observer-mallia selvittämään, mille selaimille viesti tapahtumasta välitetään. Node.js-sovellus tarkkailee osoitteita, joihin Rails-sovellus lähettää viestejä. Viestin saapuessa yhteen määritellyistä osoitteista tullaan tarkkailijoita huomauttamaan Observer-mallin mukaisesti. Koodiesimerkissä 3 on kuvattu uusien tarkkailijoiden lisääminen ja niiden huomauttaminen.

```
# Muuttuja, johon tallennetaan kaikki tarkkailijat
observers = {}

# Rekisteröi uuden tarkkailijan
addObserver = (subject, observer) ->
  observers[subject] = [] unless observers[subject]
  observers[subject].push observer

# Lähettää tiedon ja viestin tapahtumasta tarkkailijoille
notifyObservers = (subject, action, data) ->
  subjectObservers = observers[subject] || []

  for observer in subjectObserver
    observer.onAction action, data

# Alustetaan tapahtumien tarkkailijaolio
eventFeed = new feed.EventFeed(io.of('/events'))
eventFeed.init()

# Rekisteröidään tarkkailijaolio tarkkailemaan event-tyyppisiä viestejä.
addListener('event', eventFeed)

# Rekisteröidään Node.js-sovelluksen osoitteet.
#
# Kun uusi pyyntö saapuu johonkin osoitteeseen,
# ilmoitetaan tarkkailijoille tapahtumasta.
app.post '/event_added', (req, res) ->
  notifyObservers 'event', 'add', req.body
  res.send 200

app.post '/event_updated', (req, res) ->
  notifyObservers 'event', 'update', req.body
  res.send 200

app.post '/event_removed', (req, res) ->
  notifyObservers 'event', 'remove', req.body
  res.send 200
```

**Koodiesimerkki 3.** CoffeeScript-ohjelmaesimerkki, kuinka Node.js-sovellus on toteutettu Observer-mallin mukaisesti.

### 2.5.3 Selaimen ja Node.js:n välinen kommunikointi

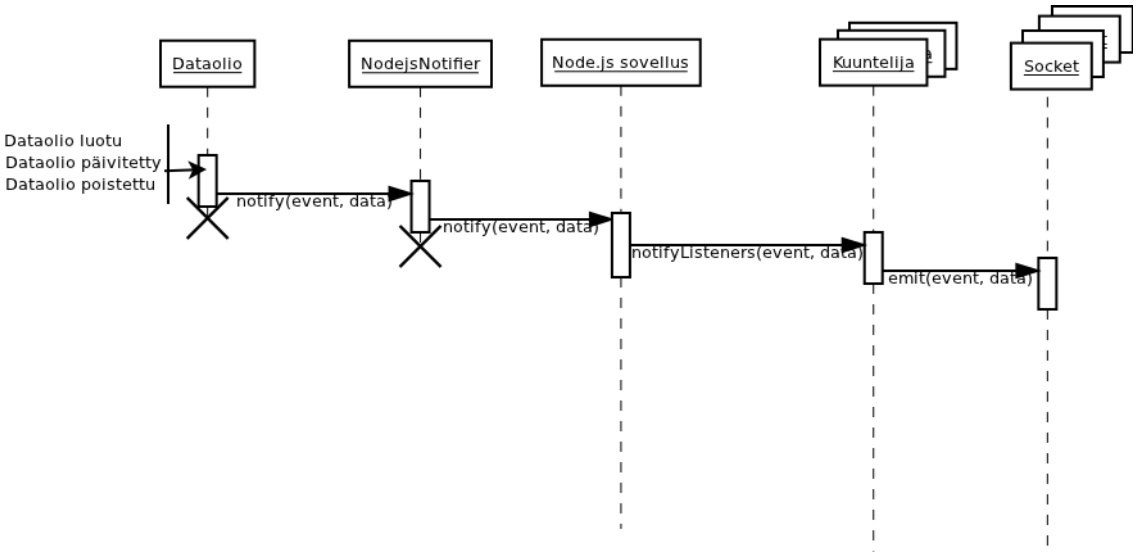
Tapahtumien tarkkailijat pitävät sisällään tiedon yhteysinstansseista, jotka haluavat saada vain tietyn tapahtuman viestejä. Tarkkailijaluokka implementoi onAction-metodin, jota kutsutaan, kun tarkkailijan kohteelle on tapahtunut jotain. onAction-

metodi ottaa vastaan kaksi parametria, action-parametrin ja data-parametrin. Action-parametri ilmaisee tapahtuman tyypin: lisäys, päivitys tai poisto. Data sisältää tietueen muuttuneen datan. Parametrit välitetään vain niille yhteysinstansseille, jotka ovat tallennettu kyseiseen luokkaan. Asiakaspään ohjelmakoodi pystyy muokkaamaan käyttöliittymää selaimen vastaanottamien viesti- ja dataparametrien perusteella. Käyttäjälle voidaan esimerkiksi ilmoittaa uudesta tapahtumasta. Koodiesimerkissä 4 on malli kuinka tarkkailijaluokka ja datan välitys selaimille on toteutettu.

```
# EventFeed-luokkaan tallennetaan selaimen ja
# Node.js välisen yhteyden instanssin. EventFeed-luokkaan
# tallennetaan vain ne yhteysinstanssit, jotka tarkkailevat events-tyyppisiä
# tapahtumia.
#
# Luokka rekisteröidään tarkkailijaksi, ja tapahtuman sattuessa,
# onAction-metodia kutsutaan.
class EventFeed
  constructor: (@sockets) ->

  init: ->
    @sockets.on 'connection', (socket) ->
      # älä tee mitään

  # Lähettää viestin kaikille selaimille
  onAction: (action, data) ->
    @sockets.emit action, data
```

**Koodiesimerkki 4.** Ohjelmaesimerkki yksinkertaisesta kuuntelijaluokasta, joka välittää tiedon selaimille, kun jotain on tapahtunut.

Data välitetään kaikille selaimille, jotka tarkkailevat kyseistä tapahtumaa. Kuviossa 8 on malli, kuinka selain vastaanottaa tiedon ja datan Node.js-sovellukselta. Asiakaspään Socket.IO-ohjelma tarkkailee saapuvia viestejä ja ilmoittaa kuuntelijoille, kun viesti on vastaanotettu. Socket.IO Callback-metodin sisällä kutsutaan tapahtumakalenterin metodia, joka muokkaa käyttöliittymää tai ohjaa kutsun muille toimintametodeille.




**Kuvio 8.** Sekvenssikaavio tiedon välityksestä selaimille.

### 2.5.4 Sisällön päivittyminen käyttöliittymässä

Käyttöliittymässä tehdään muutokset joko automaattisesti tai manuaalisesti. Oletuksena käytössä on manuaalinen päivitys. Se tarkoittaa, että käyttäjä itse päivittää sivun, kun uutta sisältöä on saatavilla. Käyttäjä pystyy myös kytkemään automaattisen päivityksen päälle käyttöliittymän kautta. Manuaalisessa päivityksessä Node.js-sovellukselta tulevasta datasta analysoidaan, minkä tyyppinen toiminta on kyseessä. Toiminnan ollessa esimerkiksi lisäys, tallennetaan lisättävä data Callback-metodiin, joka hoitaa lisäyksen käyttöliittymään käyttäjän halutessa. Koodiesimerkissä 5 esitetään etusivun tapahtumien sisällön päivitys.

```
# Yhdistetään Node.js-sovellukseen ja aloitetaan
# tarkkailemaan events-tyyppisiä tapahtumia
eventFeed = io.connect 'http://' + node_app_host + ':1337/events'
eventDataBuffer = new $.DataBuffer

# Suorittaa kaikki puskurissa olevat
# callback-metodit.
applyUpdates = ->
  for data in eventDataBuffer.getBuffer()
    callback = data.method
    callback(data.param)
  eventDataBuffer.reset()

# Tarkkaillaan lisäys-tapahtumia.
#
# Kun tätä metodia kutsutaan,
# lisätään puskuriin tiedot, mitä callback-metodia,
# minkä tyyppistä ja millä parametrillä metodia kutsutaan.
eventFeed.on 'add', (event) ->
  eventDataBuffer.push(action: 'add', method: addEventToList, param: event)

# Tarkkaillaan päivitys-tapahtumia.
eventFeed.on 'update', (event) ->
  eventDataBuffer.push(action: 'update', method: updateEventDom, param: event)

# Callback-metodi osaa lisätä uuden tapahtuman listaan
#
# TODO: Implementoi logiikka!
addEventToList = (event) ->
  console.log 'added'

# Callback-metodi osaa päivittää tapahtuman tiedot
updateEventDom = (event) ->
  eventDiv = $('#event-' + event.id)

  if eventDiv
    updateEvent(event.id, 'name', event.name)
    updateEvent(event.id, 'description', event.description)

# Näytetään käyttäjälle ilmoitus, että
# uutta tietoa on saatavilla. Callback-metodit
# suoritetaan kun käyttäjä haluaa päivittää sisällön.
showUpdatesAvailableMessage = (data) ->
  event = data.param
  return if not event.approved

  $('.available_updates').slideDown('fast')
  $('.update_content_link').attr('id', 'update_event_content')
  $('#update_event_content').click (e) ->
    e.preventDefault()
    applyUpdates()
    $('.update_content_link').attr('id', '')
    $('.available_updates').slideUp('fast')

# Tarkkailaan mitä tahansa saapuvaa dataa, ja näytetään heti
# viesti uuden datan vastaanotettua.
eventDataBuffer.onData(showUpdatesAvailableMessage)
```

**Koodiesimerkki 5.** Callback-metodien suoritus ja käyttöliittymän muokkaus.

Datan sisältävä Callback-metodi varastoidaan puskuriin ja metodi jää odottamaan käyttäjän suoritusta. Puskuri tyhjennetään Callback-metodeista, kun muutokset on suoritettu. Automaattisessa päivityksessä ei käytetä puskuria ollenkaan. Automaattisessa päivityksessä data tallennetaan myös Callback-metodiin, mutta se suoritetaan saman tien ja muutokset näkyvät käyttöliittymässä heti. Koodiesimerkissä 6 on malli puskurin toteutuksesta.

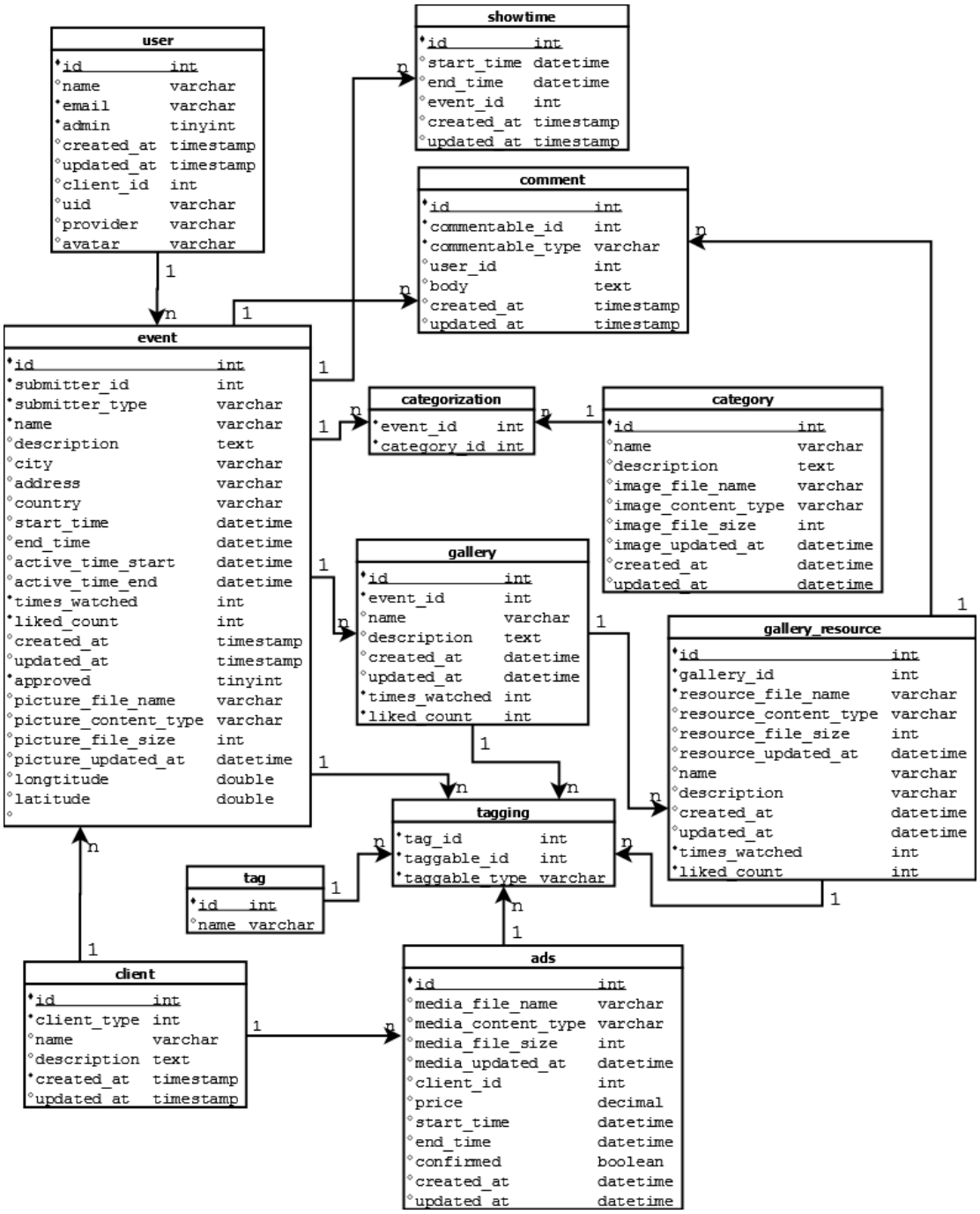
```
$.DataBuffer = class DataBuffer
  construct: ->

  # Kun uutta dataa saapuu,
  # huomautetaan tarkkalijoita asiasta
  push: (data) ->
    @getBuffer().push(data)
    @notifyObservers(data)

  getBuffer: ->
    if not @buffer
      @buffer = []
    return @buffer

  notifyObservers: (data) ->
    for observer in @getObservers()
      observer(data)

  reset: ->
    @buffer = []

  getObservers: ->
    if not @observers
      @observers = []
    return @observers

  # Rekisteröidään uusi tarkkailija-metodi (callback).
  # Callback-metodi saa parameterina datan.
  onData: (fn) ->
    @getObservers().push fn
```

**Koodiesimerkki 6.** Puskurin toteutus.

## 2.6 Tietokannan malli

Tapahtumakalenterin tietokanta koostuu useista tauluista. Tietokannan rakenteessa on hyödynnetty monimuotoisuutta vähentämään ylimääräisiä tauluja. Kuviossa 9 tietokannan taulut ”comments” ja ”tagging” ovat monimuotoisia. Railsin ActiveRecord tukee oletuksena monimuotoisia malleja, ja ominaisuuden hyödyntäminen on yksinkertaista. (A Guide to Active Record Associations 2011)

Kaikkien tietokannan taulujen viiteavaimet ovat indeksoitu, jotta haut olisivat nopeampia. Nopeus on havaittavissa varsinkin kyselyissä, joissa haetaan tietoa liitostaulujen kautta (Performance Tuning SQL Server Joins 2005). Monimuotoisten taulujen ”*_id” ja ”*_type” -kentät ovat myös indeksoitu, koska niitä tarvitaan toistuvasti.




**Kuvio 9.** Tapahtumakalenterin tietokantataulujen ja suhteiden toteutus.

## 2.7 Ruby on Rails sovelluskehyksenä

Ruby on Rails, on Ruby-kielellä toteutettu Model-View-Controllerin eli MVC-mallin mukainen sovelluskehys, jota käytetään pääasiassa web-pohjaisten sovelluksien kehittämisessä. Ruby on Railsin ensimmäinen versio julkaistiin vuonna 2003 ja sen alkuperäinen suunnittelija ja kehittäjä on tanskalainen David Heinemeier Hansson. (Getting Started with Rails 2011; Ruby on Rails 2011) Monet suuret yritykset käyttävät Railsia (Fernandez 2008).

Rails on herättänyt kiinnostusta ohjelmistokehittäjien keskuudessa juuri sen helppouden ansiosta. Mielenkiintoa Railsia kohtaan herättää myös sen ketteryys prototyyppien ja sovellusten kehityksessä. Railsin vaivattomuus ja ohjelmoijaystävällisyys on Ruby-kielen ansiota. Ruby-kielen yhtenä tavoitteena onkin parantaa ohjelmoijan ilmaisukykyä ohjelmakoodissa. Ruby-kielen syntaksi muistuttaa luonnollista kieltä ja se tekeekin kielestä vaivatonta lukea ja kirjoittaa. Rails hyödyntää myös paljon Ruby-kielen dynaamisuutta ominaisuuksissaan, kuten ActiveRecord-mallissa ja tietokantakyselyjen muodostamisessa. (Ruby ym. 2011, xvii–xx)

Railsin huonona puolena on joskus sen hyvä puoli eli dynaamisuus. Liiallinen dynaamisuus piilottaa sen, mitä sovelluksen toiminnassa oikeasti tapahtuu, joka muun muassa vaikeuttaa virheiden etsintää.

### 2.7.1 MVC -malli

Vuonna 1979 Trygve Reenskaug kehitti sovellusarkkitehtuurin, joka erotti sovelluslogiikan ja sovelluksen sisältämän datan käyttäjälle näytettävästä käyttöliittymästä. Sen ansiosta koodista saatiin joustavampaa ja sitä oli helpompi ylläpitää. (Ruby ym. 2011, 29–32)

Mallin tehtävänä on pitää huolta sovelluksen tilasta, sovelluksen datasta ja niihin kohdistuvista operaatioista. Joissakin tilanteissa tila elää vain muutaman vuorovaikutuksen käyttäjän kanssa. Tilan muutos voi olla myös pysyvä, jolloin sen hetkinen tila

tallennetaan sovelluksen tietokantaan. Malli pitää sisällään sääntöjä ja vaatimuksia, joilla dataa voidaan ylläpitää ja käsitellä tietokannassa. Malli ei ole pelkästään dataa, vaan mallissa käsitellään myös liiketoimintalogiikkaa, jota sovelletaan dataan. (Ruby ym. 2011, 29). Sovelluksen toteutuksesta riippuen, malli voi sisältää operaatioita. Operaatioilla voidaan rekisteröidä tarkkailijaolioita seuraamaan mallissa tapahtuvia muutoksia (Koskimies & Mikkonen 2005, 142). Näkymä on tyypillisesti tarkkailija, joka seuraa tietokannan mallin tapahtumia.

Tapahtumakalenterin Event-mallin logiikasta kertova esimerkki voisi olla “katsotuimmat tapahtumat”. Mallissa rajataan tapahtumat niin, että valitaan vain tapahtumat, joita on katsottu vähintään 100 kertaa, ja näytetään niistä 50 ensimmäistä.

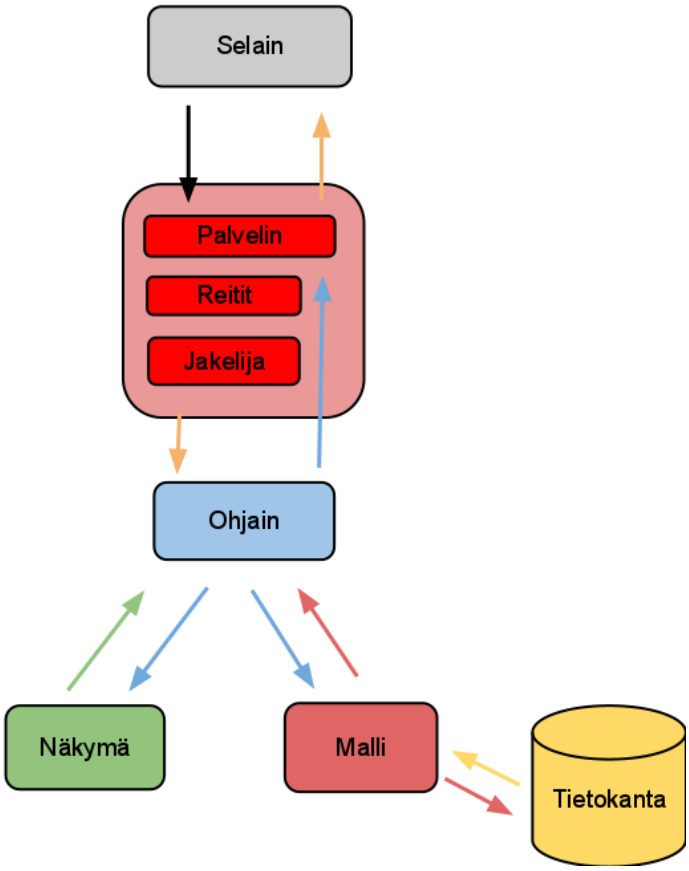
Näkymän tehtävä on näkymän eli käyttöliittymän luominen käyttäjälle. Näkymässä voidaan esittää käyttäjälle syöttölomakkeita, mutta näkymä ei koskaan kuitenkaan itse käsittele dataa. Tapahtumakalenterin tietokantaan on tallennettu kaikki tapahtumat, jotka näytetään käyttöliittymän tapahtumasivulla. Tapahtumiin päästään käsiksi tapahtumamallin kautta, josta saadaan tapahtumien tiedot näkymään. Näkymä muodostetaan mallin dataan perustuen. Eri näkymät voivat käyttää saman mallin dataa eri tarkoituksiin, eli tapahtumia voidaan näyttää myös kategoriasivulla. Näkymiä voidaan myös luoda erinäköisiksi eri käyttäjätyypeille. Ylläpitäjälle voitaisiin näyttää tapahtuman hallintaan liittyviä hallintatyökaluja.

Ohjain ottaa vastaan käyttäjän antaman GET- tai POST-komennon käyttöliittymästä ja ilmoittaa mallille käyttäjän komennosta, joka luultavasti tulee muuttamaan mallin tilaa jollakin tapaa. Mallin tilaa muuttaa esimerkiksi tapahtuman lisääminen sovellukseen. Mallille tehdyn toimenpiteen jälkeen ohjain lähettää käyttäjälle näkymän, joka vastaa käyttäjän antamaa komentoa. Ohjaimen tehtävänä on siis ohjata sovelluksen toimintaa käyttäjän komentoihin perustuen. (Ruby ym. 2011, 31–32)

Ruby on Rails, on myös sovelluskehys, joka pohjautuu MVC-arkkitehtuuriin. Malleja, näkymiä ja ohjaimia kehitetään erillään toisistaan, jotka yhdistyvät yhdeksi koko-

naisuudeksi kun sovellus suoritetaan. Railsissa yhdistämisprosessi tapahtuu automaattisesti ilman, että ohjelmoijan tulisi tehdä ylimääräisiä muutoksia sovelluskoodiin. (Ruby ym. 2011, 30)

Railsin MVC-mallia voidaan kuvata kuvion 10 esittämällä tavalla, joka on myös käytössä tapahtumakalenterissa. Kuvasta käy ilmi, kuinka käyttäjän selaimesta antama komento kulkee MVC-mallissa. Käyttäjä antaa sovelluksessa komennon avata sivun “localhost:3000/events/1”, jolloin selain tekee pyynnön palvelimelle kyseisestä osoitteesta. Palvelin vastaanottaa pyynnön ja Rails tutkii reititystiedostosta, mitä ohjainta ja toimintoa pyynnössä tarvitaan. Esimerkissä ohjaimena toimii ”events” ja toimintona näytetään tapahtuma ”1”. Jakelijan tehtävänä on alustaa ohjainolio ja välittää parametrit oikealle toiminnolle. Events-ohjaimen "Näytä"-metodi tietää hakea tapahtuman. Events-ohjain pyytää mallia hakemaan tapahtuman ”1” tietokannasta ja näyttämään sen käyttäjälle. Mallit ovat tietokannan taulun ilmentymiä, jotka vastaavat logiikasta. Ne ovat yhteydessä tietokantaan, jossa ylläpidetään sovelluksen dataa. Käyttäjälle näytetään tapahtuman näkymä ”Näytä”-toiminnon tuloksena yksinkertaisimmillaan HTML-koodina, mutta yleensä näkymää on tehostettu CSS-tyyleillä JavaScriptillä. Ohjain palauttaa vastauksen rungon ja metadatan serverille, jossa se yhdistetään HTTP -vastaukseksi ja lähetetään käyttäjälle. Sama sykli suoritetaan joka kerta, kun käyttäjä lähettää pyynnön sovellukselle. (Azad 2007)




**Kuvio 10.** Railsin MVC-arkkitehtuuri

(Azad 2007)

## 2.8 Testaaminen

Testaaminen on olennainen osa sovelluskehitystä. Sovelluksia testaamalla havaitaan niissä ilmeneviä vikoja ja puutteita. Testaajan ei tarvitse olla sovelluksen tehnyt yritys, vaan todennäköistä on, että jokin ulkopuolinen yritys hoitaa sovelluksen testaamisen. Sovelluksen testaamista voidaan suorittaa koko kehitysvaiheen aikana testaustavoista riippuen. Testaus voidaan suorittaa ennen sovelluskoodin kirjoittamista tai vasta ohjelmointivaiheen jälkeen. (Ambler 2011)

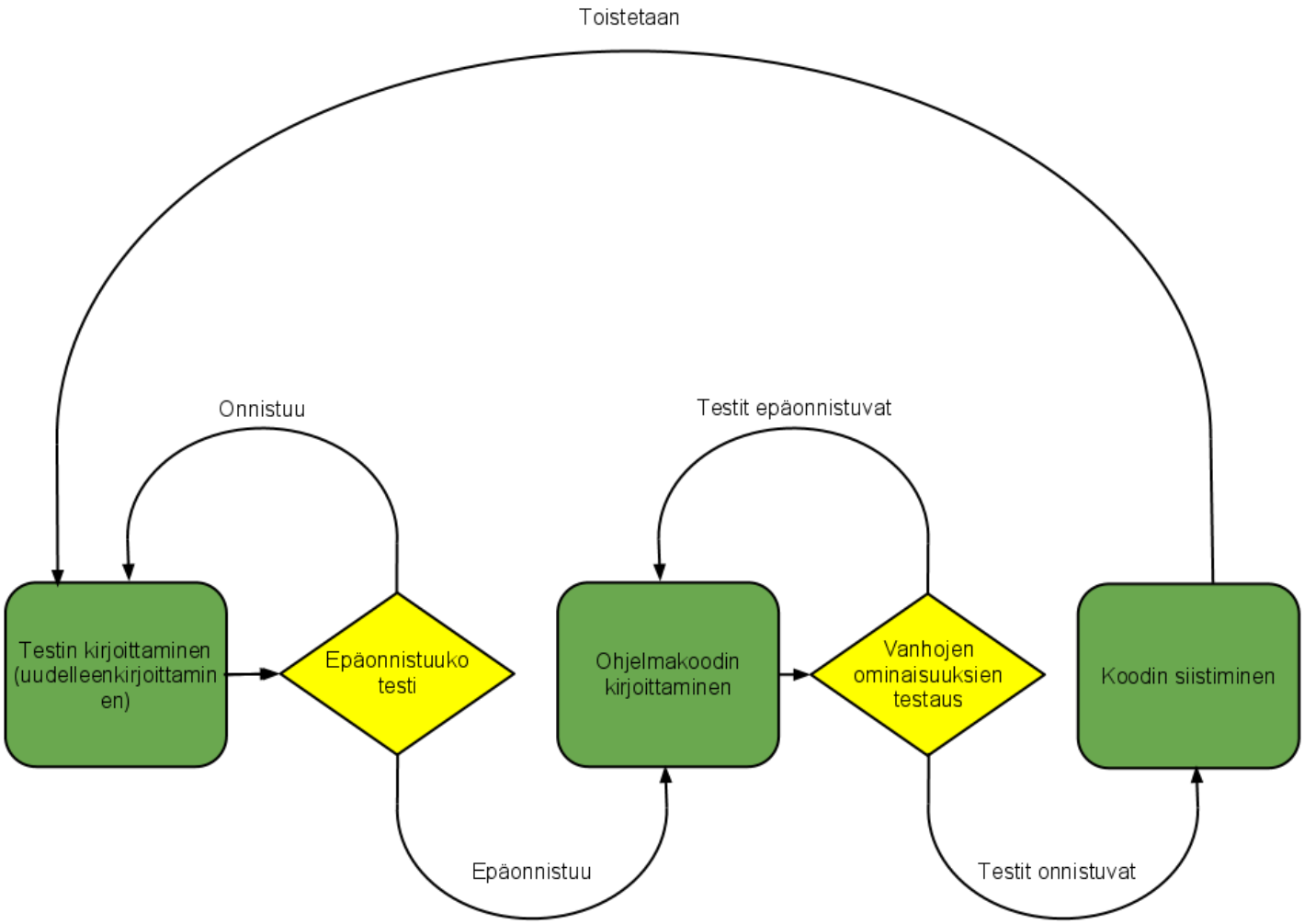
Tapahtumakalenterin toteutuksessa teimme testausta käyttöliittymässä aina uusien ominaisuuksien toteuttamisen jälkeen. Tapahtumakalenterityyppisten sovellusten toteuttamisessa testilähtöinen (TDD) tai käyttäjälähtöinen (BDD) kehitys voisi olla sopiva kehitysmalli.

### 2.8.1 TDD

Test-driven developmentin tarkoituksena on, että kirjoitetaan sovelluksen ominaisuuksille testitapaukset ennen sovelluskoodin kirjoittamista. TDD:n toiminta perustuu toistuviin lyhyisiin syklimäisiin toteutuksiin. Uudelle ominaisuudelle tai vanhan ominaisuuden parantamiselle kirjoitetaan testitapaukset olettamalla ominaisuuden toimivan halutulla tavalla, jotka palauttavat joko ”true” tai ”false”. Testien kirjoittaja tutkii käyttötapauksia ja käy yhdessä asiakkaan kanssa ominaisuuden kaikki vaatimukset läpi. Testien kirjoittajan on ymmärrettävä ominaisuuden vaatimukset tarkasti, jotta hän osaa ottaa kaikki asiat huomioon testeissä. (Ambler 2011)

Syklien avulla kehitetään ominaisuuksia pienissä osissa. Syklit tulee pitää lyhyinä, sillä jos koodi rikkoo muiden ominaisuuksien testitapaukset tai sovelluskoodi ei muuten vaan tyydytä, niin voidaan palauttaa edellinen toimiva versio. (Ambler 2011)

Uuden ominaisuuden testien ei tule suoriutua onnistuneesti ennen sovelluskoodin kirjoittamista. Mikäli testi onnistuu ennen sovelluskoodin kirjoittamista, niin ominaisuus on joko toteutettu tai testi on puutteellinen. Sovelluskoodia kirjoitetaan kunnes testit suoriutuvat onnistuneesti. Mikäli kaikki aikaisemmatkin testitapaukset suoriutuvat onnistuneesti, niin ominaisuuksien katsotaan vastaavan testitapausten vaatimuksia. Ominaisuuksien koodia voidaan refaktoroida uusissa sykleissä. Refaktorointi vaatii testitapausten uudelleen suorittamista, jotta voidaan varmistaa olemassa olevan toiminnallisuuden virheettömyyden. Kun kaikki testitapaukset suoriutuvat onnistuneesti, niin sovelluskoodi vastaa testitapausten vaatimuksia. Kuviossa 11 nähdään tarkemmin TDD:n syklin kulku. (Ambler 2011)




**Kuvio 11.** Kuvaus Test-Driven Developmentin syklistä.

(Ambler 2011)

### 2.8.2 BDD

Behavior-Driven Development on ketterän kehityksen tekniikka, jonka avulla yritetään tuoda asiakkaat mukaan testausprosesseihin. TDD:ssä kirjoitetut testitapaukset kirjoitetaan BDD:ssä luonnollisella kielellä, joita myös analyytikot ja testaajat ymmärtävät ohjelmoijien lisäksi. BDD:ssä testitapaukset kirjoitetaan kokonaisilla lauseilla. Sillä pyritään selkeyttämään testattavaa aihetta ja parantamaan testien luetta-

vuutta. BDD:ssä on oleellista ajatella, mitä tärkeitä toimintoja nykyinen sovellus ei toteuta. Testitapauksessa kuvataan, mitä toiminnolla halutaan saavuttaa. (North 2011)

BDD:n testitapauskuvauksissa hyödynnetään usein vaatimusmäärittelyssä listattuja asioita, jossa on yleensä kerrottu, mitä sovelluksen tulisi tehdä. Vaatimusmäärittelystä poimitaan vaadittu toiminto, ja toiminnon kuvaus muokataan BDD:n testitapaukseen sopivaksi. Kun BDD:n testejä suoritetaan, nähdään, mitä toimintoja on toteuttamatta ja mitkä toiminnot ovat puutteellisia. (North 2011)

Testitapausmalli sisällön reaaliaikaisesta päivittämisestä BDD:tä käyttäen:

**Sisällön reaaliaikainen päivittäminen**

- **Oletetaan,** että käyttäjä on etusivulla
- **ja** live mode on pois päältä
- **ja** sisältö on päivitetty
- **ja** ilmoitus saatavilla olevasta sisällöstä on näytetty
- **kun** päivitän sisällön napsauttamalla “Päivitä”
- **sitten** näen päivittyneen sisällön.

# 3 KÄYTETTÄVYYSAJATTELU

Käyttöliittymän suunnittelussa käytettävyys oli tärkein asia, pyrittiin minimalistiseen ja esteettiseen ulkoasuun, jossa sisältö määräisi sivuston rakenteen. Navigointi sivustolla haluttiin mahdollisimman yksinkertaiseksi, joten päädyimme yhden napin navigointiin. Halusimme myös tärkeimmät työkalut (sisäänkirjautuminen, lisää tapahtuma ja pikahaku) suoraan sivuston ylälaitaan eli headeriin, koska käyttäjät tulevat todennäköisesti etsimään ensimmäiseksi juuri näitä elementtejä. Käyttöliittymän ja elementtien asettelussa päädyttiin noudattamaan klassisia Jakob Nielsenin kirjoittamia käytettävyyden teorioita.

## 3.1 Nielsenin käytettävyysteoriat

Käyttöliittymän käytettävyyden suunnittelussa voidaan käyttää esimerkiksi alla olevan listauksen mukaisesti kymmentä Jakob Nielsenin heuristista pääperiaatetta.

1. Järjestelmän tilan näkyvyys (Nielsen 2005)

Reaaliaikaisen ja järkevän palautteen tarjoaminen järjestelmän toiminnasta ja nykytilasta. Käyttäjällä tulisi aina olla käsitys sijainnistaan sovellusta tai web-sivustoa käyttäessään: “millä sivulla olen, mistä tulin tänne ja mihin voin jatkaa tältä sivulta”. Informaatio nykyisestä tilasta tai sijainnista voidaan tarjota käyttäjälle esimerkiksi murupolun avulla, eli näytetään nykyisen sivun nimi listassa tai tekstirivinä yläsivujen kanssa josta nykyiselle sivulle on navigoitu. Informaatio voidaan myös antaa graafisin keinoin esimerkiksi korostamalla nykyisen sivun nimi sivuston ylälaidan valikossa.

Käyttäjä voi helposti turhautua tai tuntea olonsa eksyneeksi, jos hänelle ei tarjota informaatiota sovelluksen tilasta tai sijainnistaan sivustolla. Tapahtumakalenterin valikkorakenne perustuu piilotettuun päävalikkoon, joten käytimme sovelluksessa mu-

rupolkua kertomaan käyttäjälle millä sivulla hän milloinkin on. Tämä helpottaa ja nopeuttaa navigointia, vaikka ei sinänsä tarjoakaan mitään uutta päävalikkoon verrattuna.

2. Järjestelmän ja maalaisjärjen välinen yhteys, oikean kielen ja sanaston käyttö. Tieto on syytä näyttää luonnollisella tavalla ja loogisessa järjestyksessä. (Nielsen 2005)

Sisältöä luodessa on päätettävä suurpiirteinen kohdejoukko, jolle sivuston sisältö luodaan ja muokataan sopivaksi. Kaikki sisältö, tekninenkin pitäisi kuitenkin olla selkokielellä ja käyttäjän kannalta oikealla kielellä, sillä käyttäjä ei yleensä ala kääntämään sivuston tekstejä käännösohjelmien avustuksella.

Internetin käyttäjät eivät yleensä käytä riittävästi aikaa sivuston sisällön kunnolliseen lukemiseen. Tämän takia kannattaa aloittaa jokainen sivu sen tärkeimmällä sisällöllä. Tärkein sisältö tarjotaan heti ensimmäisenä, tätä kutsutaan “käänteiseksi pyramidi”-säännöksi. Käyttäjän pitäisi nopealla vilkaisulla pystyä päättelemään, mitä sivu sisältää ja mitä tarjottavaa sillä on hänelle. Usein sisältöä silmäilevät käyttäjät lukevat jokaisen kappaleen ensimmäisen lauseen; tästä voidaan päätellä, että kappaleiden otsikointi on tärkeää. On myös hyvä toimia periaatteen “yksi ajatus per kappale” mukaan. Tapahtumakalenteri päätettiin toteuttaa kahdella kielellä: suomi ja englanti. Jatkossa sivuston tekstejä voidaan kääntää muillekin kielille käyttäjäkannan kasvattamisen toivossa.

3. Käyttäjälähtöinen vapaus järjestelmän hallintaan, tarjotaan käyttäjälle vapaus hallita toimintaansa. (Nielsen 2005)

Sivustolla navigoidessaan käyttäjä käyttää erinäisiä hallintatoimintoja. Eteen- ja taaksepäin ovat jo vakiintuneet selainten perustoiminnoiksi. Mutta valintamahdollisuudet, kuten tilanteen tai toiminnan varmistaminen ja peruuttaminen tai tapahtuman muokkaaminen ja poistaminen, on aina syytä tarjota käyttäjälle. Tällaisten perustoimintojen puuttuminen sivustolta aiheuttaa helposti tilanteita, joissa käyttäjä haluaa vain sulkea

sivuston, sillä ei ole varmuutta, mitä seuraavaksi tapahtuu tai mikä toiminto tuottaa halutunlaisen tuloksen.

4. Johdonmukaisuus käyttöliittymässä. Käyttäjän ei pitäisi joutua miettimään, mitä mikäkin sana tarkoittaa tai miten tietty asia tehdään tässä sovelluksessa. Sanasto ja toiminnot tulisivat olla alanmukaisia ja standardoituja. (Nielsen 2005)

Johdonmukaisuus on elintärkeä asia käyttöliittymäsuunnittelussa. Samanarvoisten elementtien pitäisi olla samannäköisiä tai -kokoisia. Saman toiminnan suorittavien hallintatoimintojen pitäisi aina löytyä samasta paikasta. Valikoiden sijainti ja toiminta on vakiintunut niin, että samantyyppisiä valikkorakenteita näkeekin usein. On myös tärkeää että valikko, joka näyttää samalta, myös toimii samalla tavalla kuin vastaavanlaiset valikot muilla sivustoilla. Valikko on pidettävä paikoillaan lähes kaikilla sivuilla navigoinnin opittavuuden varmistamiseksi, poikkeuksina ainoastaan etusivu (joissakin tapauksissa) ja lomakesivut. (Krug 2006, 62–63)

Suurelle osalle web-sivuston sisällön osista on muodostunut standardimuotoilu ja säännöstö, esimerkiksi hakukenttä on hyvin tarkkaan määritelty elementti, jonka kaikki tunnistavat ja jolta kaikki odottavat tietynlaista toimintaa. Hakukenttää suositellaankin kaikille sivustoille, joilla on vähänkään sisältöä. Käyttäjä etsiikin usein katseellaan sivustolta ensimmäisenä hakukenttää tai jotain, joka näyttää hakukentältä. (Krug 2006, 67)

5. Virheiden vähyys (Hyvän virheviestin voittaa vain virheen ennaltaehkäiseminen.)
6. Virheiden tunnistettavuus, diagnosointi ja niistä toipuminen (Virheviestin tulisi kertoa virhe selkokielellä ja selvittää virheen syy tarkasti ja ehdottaa ratkaisuvaihtoehdot rakentavasti.)

(Nielsen 2005)

Virheiden vähentämiseksi olisi hyvä välttää tilanteita joissa virheitä helposti sattuu, tai ainakin varautua niihin. Muun muassa varmistusviesti voi toimia virheen estäjänä. Jo sovelluksen testausvaiheessa tulisi kiinnittää huomioita kohtiin, joissa saatiin aikaan virhe sovelluksen toiminnassa, ja joko korjata virheen aiheuttava osa koodia tai lisätä virheelle käyttäjän kannalta järkevä käsittely eli asiallinen virheilmoitus ja selkeät ohjeet jatkotoimista.

7. Tunnistettavuus mieluummin kuin muistettavuus (Nielsen 2005)

Sovelluksen käytettävyyden kannalta muistettavuuttakin parempi ominaisuus on tunnistettavuus. Näin minimoidaan käyttäjän muistille luotava taakka laittamalla kuvat, ikonit ja ohjetekstit näkyviin kaikkiin sellaisiin paikkoihin, joissa niitä tarvitaan, ja tarjotaan niitä myös helposti myös paikoissa, joissa niiden ei välttämättä tarvitse olla esillä. Esimerkkinä tilanne, jossa hyväksymistoiminto on toteutettu napsautettavana linkkinä; jos linkki on aikaisemmin hyväksynyt tapahtuman, käyttäjä saattaa muistaa toiminnan. Jos linkistä tehdään napin näköinen, lisätään siihen teksti “Hyväksy” ja värjätään nappi vielä vihreäksi. Niin napin tarkoitus ei jää käyttäjälle epäselväksi.

Monet internetin käyttäjät muistavat, miltä poisto tai muokkausikoni suurin piirtein näyttää nähtyään niitä muilla sivustoilla. Käytettävyyttä voidaan kuitenkin vahvistaa yhä luotettavammaksi ja arvattavammaksi antamalla tutuillekin ikoneille title-tekstit, jotka näkyvät, kun hiiren osoitin viedään ikonin päälle. Näin käyttäjä tietää, että napsauttaa varmasti oikeaa ikonia jo ennen kuin napsautus on tapahtunut.

8. Joustavuus ja käytön tehokkuus. (Sovelluksen muuttaminen käyttäjän mukaan, esimerkiksi antamaan lisäohjeita ensimmäisen kerran käyttäjälle tai jättämään ohjeistus pois kokeneemmalla käyttäjällä.) (Nielsen 2005)

Käyttäjän ja web-sovelluksen kanssakäymistä voidaan verrata kahden ihmisen väliseen kanssakäymiseen, ihmiset käyttäytyvät eri tavoin eri ihmisten kanssa, myös web-sovelluksen pitäisi pystyä muuntautumaan käyttäjän mukaan. (Ward ym. 2011, 255–256)

Ensimmäisellä käyttökerralla käyttäjältä kysytään perustietoja samalla tavalla kuin ihmiseen tutustuessa. Liian tarkkojen kysymysten esittäminen tässä vaiheessa ajaa käyttäjät pois henkilökohtaisiin asioihin menemisen takia. (Ward ym. 2011, 255–256)

Toisella käyttökerralla pitäisi sovelluksen jo muistaa käyttäjästä perustiedot samoin kuin ihminen muistaa tuntemansa ihmisen. Myös käyttöliittymän, värimaailman tai esimerkiksi fonttikoon muuttaminen jokaisen yksittäisen käyttäjän mieltymysten mukaisesti tekee käyttökokemuksesta käyttäjälle toimivamman, tutumman ja varmasti mieluisamman. (Ward ym. 2011, 255–256)

Käyttöliittymää suunnitellessa on pidettävä mielessä normaalin sosiaalisen kanssakäymisen säännöt ja tarjottava käyttäjälle mahdollisuus kertoa itsestään ja mieltymyksistään, jolloin sovellusta voidaan muuntaa hyvinkin erilaiseksi ja uniikiksi jokaista käyttäjää varten. (Ward ym. 2011, 255–256)

9. Esteettinen ja minimalistinen design (Liika turha ja asiaankuulumaton informaatio vie arvoa ja huomiota tarpeelliselta sisällöltä.) (Nielsen 2005)

Sovelluksen määrittelyvaiheessa tutkittiin muita vastaavia palveluja ja koettiin, että vaikka relevanttia sisältöä oli laajasti ja toiminnallisuus toteutettu hyvin, oli palvelut monesti “saastutettu” liialla sisällöllä. Sivustoilla oli paljon turhaakin tietoa ja kaikki sivuston tyhjä tila oli käytetty, jolloin sovelluksen pääsisältö jäi pienempään arvoon ja huomaamattomampaan asemaan. Sivuston rakennetta suunnitellessa pitäisi sisällön olla tärkein asia. Jokaisella sivulla pitäisi olla muutama päätehtävä, joita käyttäjä tavoittelee, nämä osat ja niiden sisältö pitäisi saada selkeästi erottumaan muusta sivun sisällöstä nimenomaan elementtirakenteen oikeanlaisella järjestelyllä.

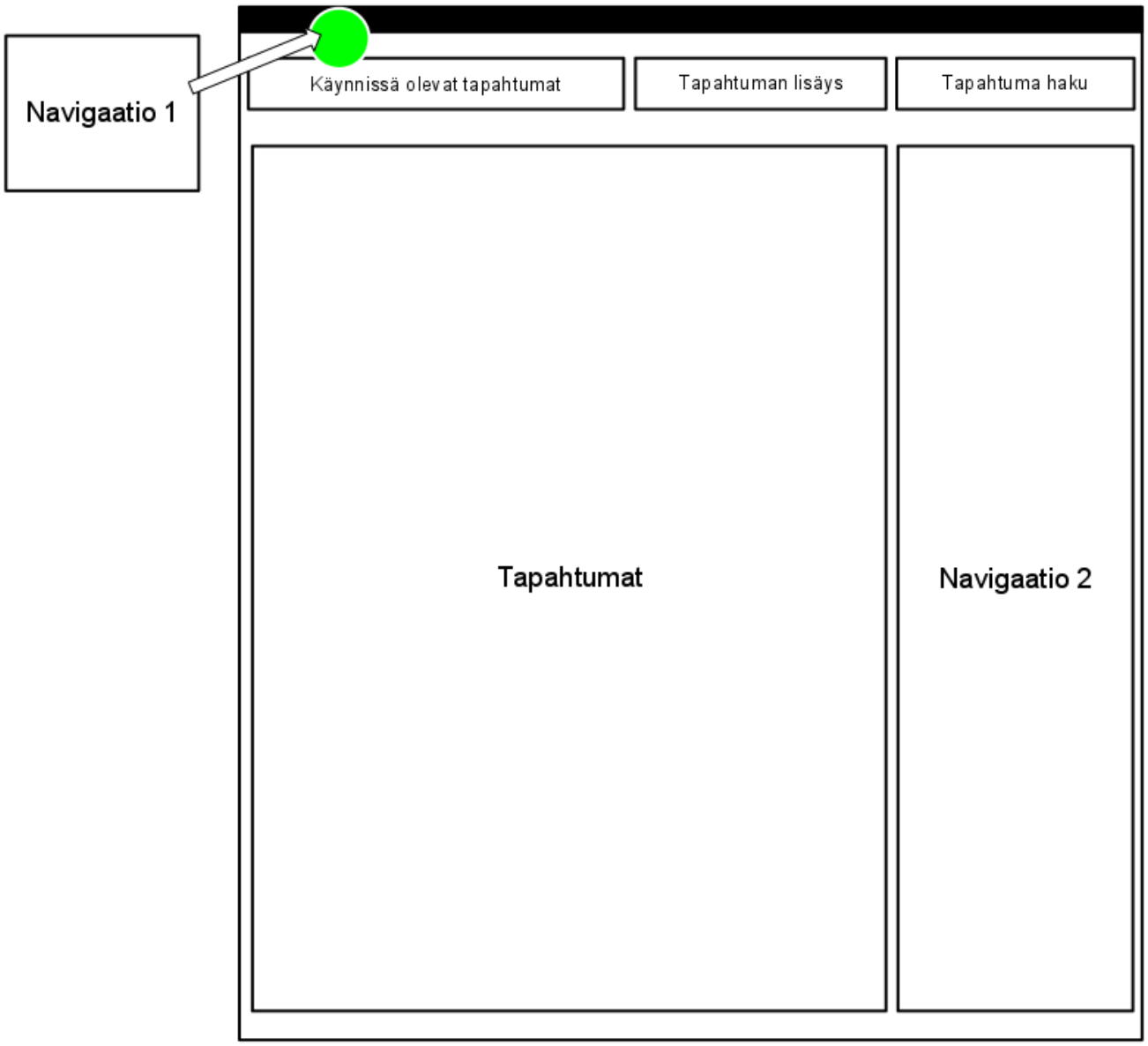
10. Avun ja dokumentoinnin tarjoaminen. (Nielsen 2005)

Vaikka järjestelmän käyttö olisikin mukavampaa ilman dokumentaation lukemista, voi olla tarpeellista tarjota apua tai mahdollisuutta lukea sovelluksen käyttöä selventävää dokumentaatiota.

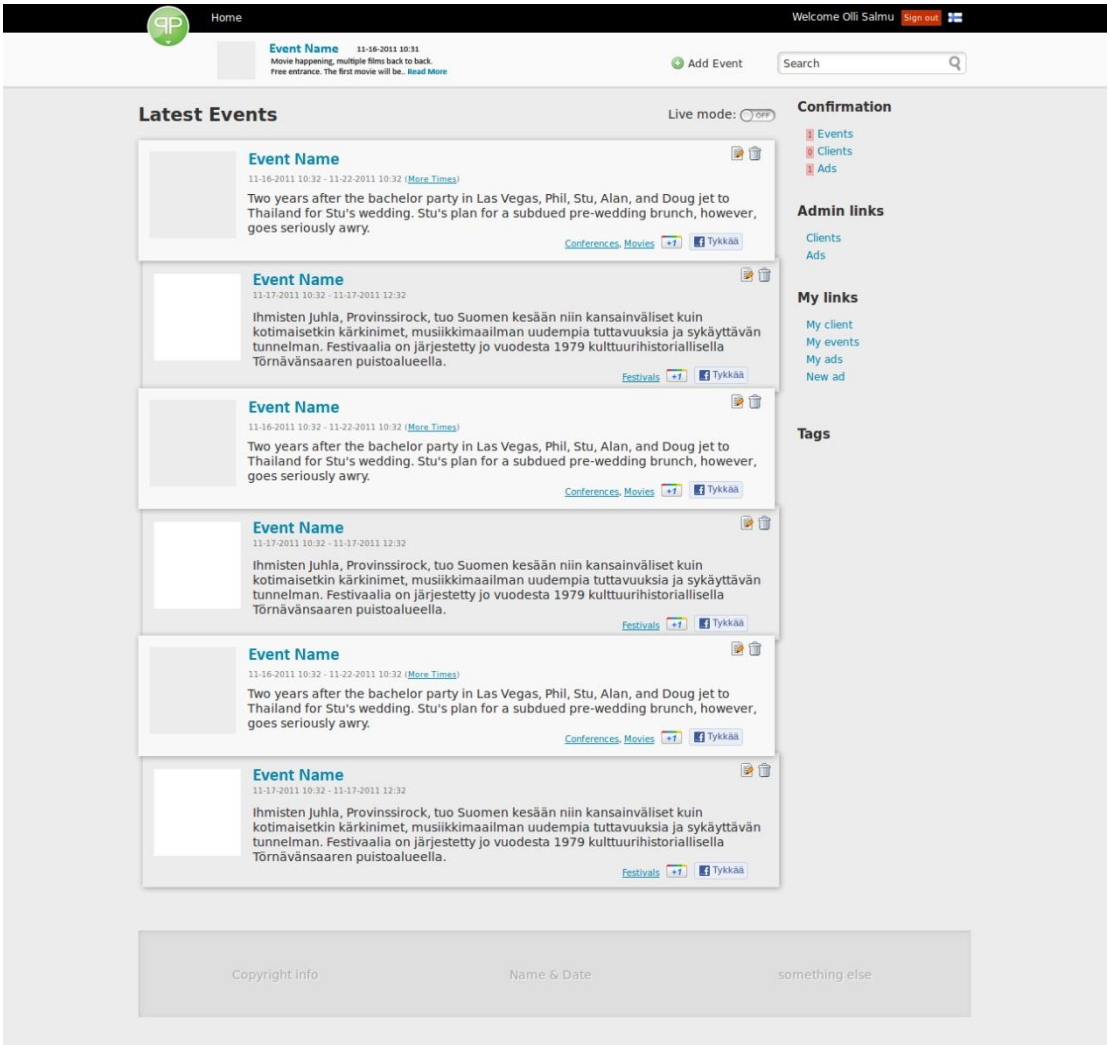
Apu tulisi olla helposti saatavilla, esimerkiksi joka sivulla näytettävän tunnistettavan ikonin takana tai jopa omana sivunaan valikossa. Avun tulisi myös olla selkeässä muodossa, josta oikea otsikko (ongelma) ja sen ratkaisu löytyisi helposti. Ohjeet ongelman ratkaisemiseen olisi syytä olla kohta-kohdalta listatussa kompaktissa muodossa ja suoraan yhteydessä käyttäjän toimintaan. Käyttäjän tulisi saada konkreettisia ohjeita juuri kokemaansa ongelmaan, eikä suurpiirteisiä järjestelmän toimintaan viittaavia ohjeita, joista ei käyttäjälle selkene selvää toimintakehotetta.

### 3.2 Lähtökohtana yksi nappi

Lähdimme rakentamaan ensimmäisiä layout-rakenteita sovelluksen kuvion 12 mukaisesti päätoimintojen ympärille: tapahtumien katselu, tapahtumien haku, tapahtumien lisäys, käynnissä olevat tapahtumat ja navigointi. Halusimme kaikki edellä mainitut toiminnot mahdollisimman helposti saataville ja sellaiseen muotoon ja paikkaan, joissa käyttäjä on ne tottunut näkemään. Kuviossa 13 on sovelluksen valmis layout.




**Kuvio 12.** Layoutin rakentuminen päätoimintojen ympärille.



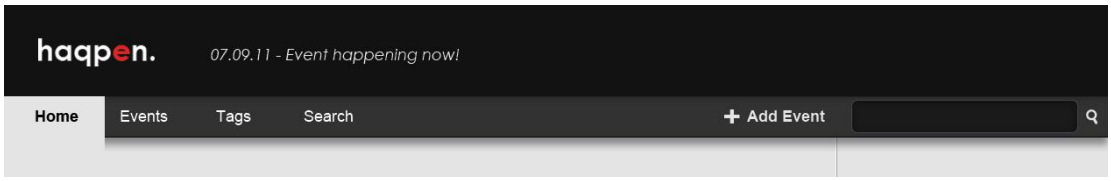

**Kuvio 13.** Valmis layoutin rakenne

Päätimme, että sovelluksemme navigoinnin lähtökohtana olisi yksi nappi, jonka kautta käyttäjä suorittaisi suurimman osan päänavigoinnistaan. Yhteen nappiin perustuva käyttöliittymä teki ulkoasusta web-sivustomaisen sijasta enemmän sovellusmaisen. Yksi päätökseen johtanut syy oli mobiililaitteiden ja taulutietokoneiden käytön yleistyminen. Navigoinnin mahduttaminen yhden napin alle, teki headerista kevyen ja layoutista yleisesti raikkaan ja siistin näköisen. Kevyt header antoi myös enemmän tilaa ja arvoa sivuston sisältö-osalle.

Kuviossa 14 näkyvä yhden napin navigointi ei yksinkertaisuudessaan tarjoa mahdollisuutta kertoa käyttäjälle millä sivustolla milloinkin ollaan, toisin kuin klassinen layout raskaalla monitasoisella menulla. Klassisessa layout-mallissa voidaan menuelementtien värityksellä ja typografialla ilmaista, mikä sivu on milloinkin aktiivinen. Toteutimme siksi sivuston headeriin murupolun, jotta käyttäjä ensinnäkin tietäisi, missä on tällä hetkellä ja toisekseen tietäisi mistä on sinne tullut. Murupolun käyttö sivustolla helpottaa navigointia eteen- ja taaksepäin verrattuna varsinaisen navigaation käyttöön. (Krug 2006, 51–78)


**Kuvio 14.** Yhden napin navigaatio

Vaihtoehtoinen ehkä klassisempi layout-tyyli olisi ollut yhteen tai kahteen tasoon järjestelty valikko headerissa, joka on esitetty kuviossa 15. Klassinen navigaatio olisi voinut olla helpommin opittava käytettävyydeltään, sillä tätä tyyliä käyttävät lukemattomat muut sivustot ja palvelut.




**Kuvio 15.** Klassinen navigaatio.

### 3.3 HTML5 ja CSS3

HTML5 on uusin HTML-spesifikaation versio, joka tuo muun muassa uusia tageja ja merkintätyylejä jo olemassa olevaan spesifikaation kirjastoon. HTML5 ja CSS3 ovat uusin kehitysaskel web-teknologiassa, ja ne on luotu helpottamaan paremman ja modernimman web-sovelluksen luomista. (Hogan 2010, 1–4)

Monet HTML:n uusista ominaisuuksista keskittyvät paremman alustan luomiseen web-sovelluksille. Paremman käyttäjäkokemuksen luomista helpottaa paremmin tarkoitustaan kuvaavat tagit, sivujen ja ikkunoiden välillä liikkuminen, animaatiot ja uuden spesifikaation parannettu multimediatuki. Jokainen uusi versio HTML:stä tuo jotain uutta merkintätyyleihin, mutta sisällön kuvaamiseen ei koskaan aikaisemmin ole suoraan liittynyt niin paljon uudistuksia kuin nyt. (Hogan 2010, 1–4)

HTML5-spesifikaation myötä ei tarvita enää Flash- tai Silverlight-liitännäisiä videon, äänen tai vektorigrafiikan toistamiseen. HTML5 tarjoaa ominaisuuksia, jotka joissain tapauksissa tekevät muiden teknologioiden käyttämisen täysin tarpeettomaksi. HTML5 tarjoaa myös WebSocket-tuen, joka mahdollistaa pysyvän yhteyden palvelimeen. Sen sijaan, että yhdistettäisiin palvelimeen tilattomalla protokollalla ja saataisiin yksittäinen vastaus pyyntöön, voi selain nyt luoda tilallisen eli avoimen yhteyden palvelimeen. Avoin yhteys mahdollistaa haluttujen muutosten suoran työntämisen kokoajan niitä odottavalle käyttäjälle reaaliajassa. Kaikissa selaimissa ei vielä ole tukea socket-yhteyksien käyttöön, mutta Adoben Flashia socket-kommunikaatiokerroksena käyttäen ja web-socket-js kirjaston avulla väliaikaisen flash-ratkaisun implementointi onnistuu. (Hogan 2010, 1–4)

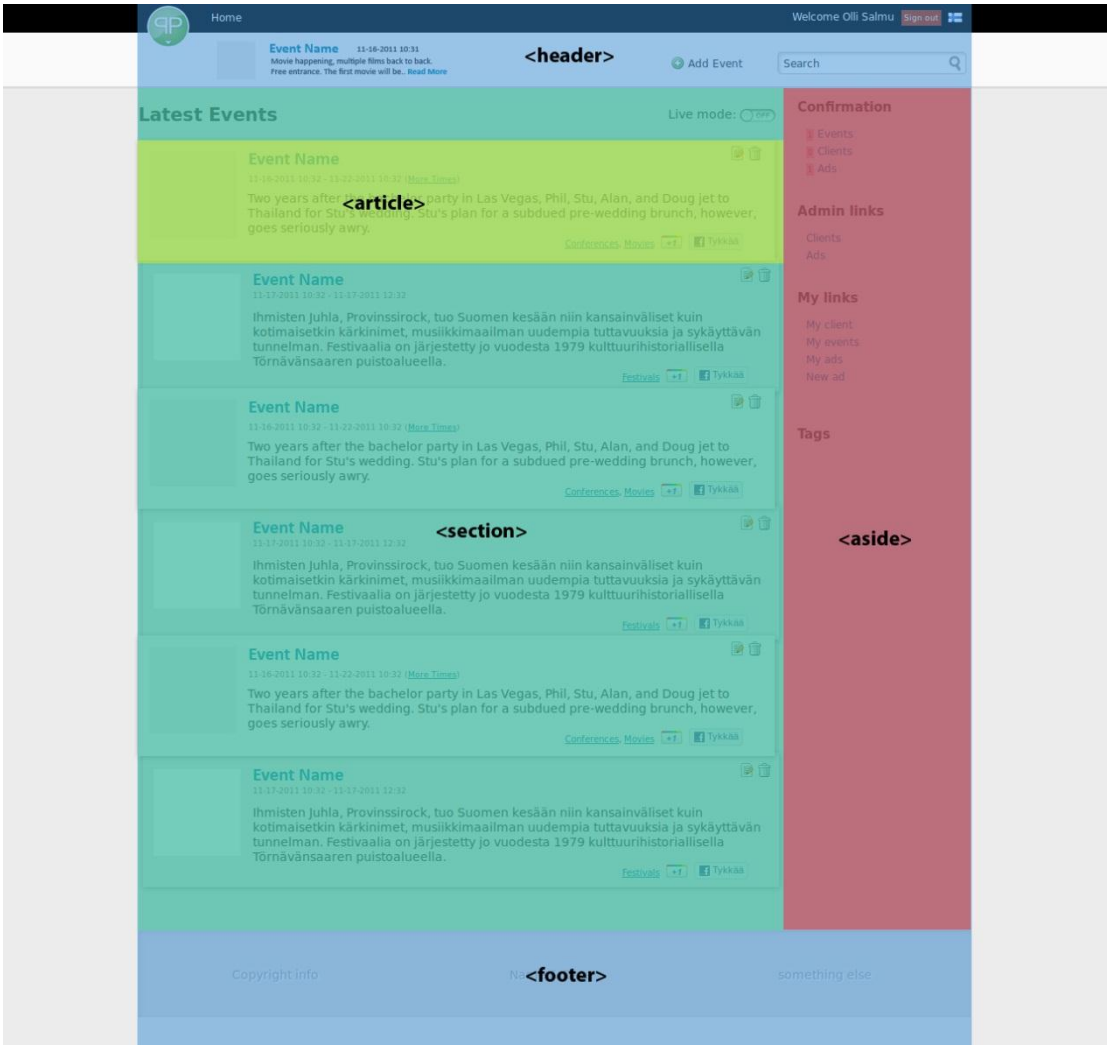
HTML5-spesifikaatiota ajatellaan yleensä web-teknologiana, mutta ”web-varastoinnin” ja webSQL-tietokantarajapintojen lisäyksen myötä voidaan web-sovellus nyt rakentaa selaimeen niin, että se käyttää dataa suoraan käyttäjän paikalliselta koneelta. (Hogan 2010, 200–206)

Käyttäjärajapinta on suuri osa web-sovellusta, ja jotta saavutettaisiin halutut tulokset selaimessa, on aikaisemmin vaadittu joko suuret määrät merkintätyylejä tai JavaScript -kirjastojen käyttöä. HTML5 ja CSS3 ovat uusilla ominaisuuksillaan luvanneet poistaa tämän ongelman. Esimerkiksi päivämäärävalitsimet ja värivalitsimet ovat nyt suoraan päteviä elementtejä HTML5:ssä, samalla tavoin kuin alasvetovalikot, valintaruudut, radionapit ja muut aikaisemmin määritetyt jo olemassa olevat elementit. (Hogan 2010, 200–206)

Uusien elementtien käyttäminen selkeästi kuvaamaan sisältöään helpottaa sivuston ymmärtämistä ja läpikäymistä lukemisohjelmistoille. Esimerkiksi sivun navigaatio on helpommin löydettävissä, kun elementin nimi on *nav* sen sijaan, että se olisi div-luokka muiden div-luokkien seassa. Lukemiseen tarpeettomien elementtien kuten header, footer ja sidebar voidaan järjestellä ohjelmistossa uudestaan tai jopa jättää kokonaan väliin. Sivuston yleinen jäsentäminen helpottuu, joka johtaa parempaan käyttäjäkokemukseen. Tämä auttaa käyttäjää luottamaan edellä mainittujen toimintojen mahdollistavien, avustavien teknologioiden käyttöön.

CSS3:n tuomilla uudistuksilla voidaan vähemmällä koodilla ja merkintätyylillä saavuttaa enemmän kuin ennen. Uusilla standardimäärityksillä voidaan esimerkiksi erottaa parilliset ja parittomat rivit taulukossa tai määrittää ryhmän viimeinen tekstikappale. Uudet ominaisuudet tuovat tyylien määritykseen ennen kaikkea lisää tehokkuutta. Visuaaliset tyylimääritykset kuten varjot ja liukuvärit tuovat sivustoon uudenlaista syvyyttä ja ulottuvuutta. CSS3 mahdollistaa näiden määritysten rajattoman käytön ilman kuvien tai muiden väliaikaisten korjausten käyttöä.

HTML5:n uusien ominaisuuksien käyttö tapahtumakalenterissa järkevöitti muun muassa sivuston osien nimeämistä ja asettelua. Uusien elementtinimikkeiden käyttö toi elementeille sisältöönsä viittaavan nimen, joka teki sivuston rakenteesta loogisen ja helposti ymmärrettävän. Elementtien nimeämistä on esitelty kuviossa 16.




**Kuvio 16.** Elementtien tarkoitustaan kuvaava nimeäminen

Käytimme myös HTML5-spesifikaatiossa määriteltyjä uusia lomakekenttätyyppejä kuten tiedostotyyppistä tekstikenttää ja päivämäärävalintakenttää.

HTML5 toimii jo useimmissa olemassa olevissa selaimissa, ja jopa Internet Explorer 6:n käyttäjät voivat käyttää HTML5-spesisifikaatiota vaihtamalla HTML- ja XML-tiedostojensa doctypen yksinkertaiseen: <!DOCTYPE html>-muotoon. (Hogan 2010, 200–206)

Internet Explorer on kuitenkin tällä hetkellä laajimmin käytetty selain, vaikka HTML5- ja CSS3-tuki onkin mukana vasta versiossa 9. Selaimille, jotka eivät tue HTML5:ttä ja CSS3:a, joudutaan tekemään väliaikaisia ratkaisuja tai kiertämään ongelmia esimerkiksi käyttämällä kuvia elementtien apuna. Uusimmat web-sovellukset luodaan yleensä pitäen silmällä uusimpia selaimia, ja tällöin sivusto voi näkyä vanhemmilla selaimilla eri tavalla. Sovellusta tehdessä päätettiin, että tuetaan vain tiettyjä selainversioita, jotta ylimääräinen testaus ja korjaustyö vältettäisiin. (Hogan 2010, 200–206)

HTML5 ja CSS3 eivät ole valmiita spesifikaatioita, ja niissä voi vielä tapahtua kehitystä. Vaikka Firefox-, Chrome- ja Safariselaimet tukevatkin jo HTML5:ttä melko vahvasti, spesifikaation muuttuessa on myös selainten muututtava. Tällaiset muutokset voivat johtaa rikkinäisiin sivustoihin, joissa on esimerkiksi vanhan spesifikaation merkintätyylejä. Spesifikaatiot kehittyvät ja merkintätyyliä paikataan tarvittaessa esimerkiksi JavaScriptillä tai Flashilla kunnes täydelliset CSS3- ja HTML5-spesifikaatiot saadaan standardoitua ja kaikki selaimet tukevat uusia spesifikaatioita 100 prosenttisesti. (Hogan 2010, 200–206)

### 3.4 SCSS ja SASS

SASS on laajennus CSS3:een. Se tuo uusina ominaisuuksina muun muassa: sisäkkäiset säännöt, muuttujien käytön ja valitsimien periytyvyyden. SASS perustuu kahteen erilaiseen syntaksiin

Uusi pääsyntaksi eli SCSS (SASS 3. versiosta eteenpäin) on CSS3-syntaksin ylijoukko, eli pätevä CSS3 tyylitiedosto on suoraan pätevää SCSS:ää. SCSS-tiedostot käyttävät päätettä ".scss".

Toinen, vanhempi syntaksi tunnetaan sisennettynä syntaksina tai SASS:na. SASS on tarkoitettu kehittäjille, jotka suosivat ytimekkyyttä toiston sijaan CSS:ssä. Siinä tyylit jaetaan puolipisteiden ja aaltosulkujen sijasta rivien sisennyksillä. SASS-tiedostot käyttävät päätettä ".sass". (Style with attitude 2011)

Koodiesimerkissä 7 SASS:n käyttö helpotti tyylien määrittämistä tapahtumakalenterissa tarjoamillaan uusilla ominaisuuksilla. Esimerkiksi muuttujien käyttö vähensi toistoa tyylitiedostoissa. Määrittelimme koko sivustolle perusvärit ja fontit käyttäen muuttujia. Muuttujien avulla pystyimme sitten aina tarvittaessa antamaan elementeille tyylit oikeita muuttujia käyttäen.

```
/* Layout */
$wrapper-width: 960px; /* 960px */
$aside-width: 200px;
$content-margin: 25px;
$content-width: $wrapper-width - $aside-width - $content-margin;

/* Colors */
$body-backcolor: #ececec;
$font-color: #323231;
$link-color: #0584AB;
$link-hover-color: #009CCC;

$header-background-color: #000;
$header-font-color: #fff;$¢
$header-bottomgroup-backcolor: #313131;
$header-usernav-link-color: #fff;
$header-usernav-link-hover-color: #fff;
$header-pagenav-link-color: #fff;
$header-pagenav-link-hover-color: #fff;
$header-pagenav-link-hover-bgcolor: #2B2B2B;
$header-signout-backcolor: #C53104;
$header-signout-color: #fff;
$header-signin-backcolor: #56A501;
$header-signin-color: #fff;

$aside-requestcount-color: #BA383A;
$aside-requestcount-backcolor: #E8B5B6;

/* Fonts */
$font-family: "DejaVu Sans", Calibri, Geneva, Verdana, Tahoma, "Trebuchet MS", FreeSans, Arial;
$font-size: 14px;
$user_nav_font-size: 12px;
$aside-link-font-size: 12px;
$aside-requestcount-font-size: 8px;
```

**Koodiesimerkki 7.** Muuttujien arvojen asettaminen SCSS-tiedostossa.

Myös sivuston eri osien leveydet ja korkeudet voitiin määrittää koodiesimerkin 8 mukaisesti heti rakenteen luontivaiheessa, jolloin saavutettiin järkevät mittasuhteet ja toistensa kanssa suhteessa toimivat elementtikoot. Valitsinten periytyvyys vähensi rivien määrää tilanteissa, joissa samoja tyylejä annettiin useammille elementeille.

```
#wrapper {
  width: $wrapper-width;
  margin: 0 auto;
}

#wrapper aside {
  float: right;
  width: $aside-width;
}

#content {
  float: left;
  margin-right: $content-margin;
  width: $content-width;
}

#page_nav{
  color: $header-pagenav-link-color;
}

#page_nav a{
  color: $header-pagenav-link-color;
  background-color: $header-pagenav-link-hover-bgcolor;
  cursor: hand;
}
```

**Koodiesimerkki 8.** Muuttujien käyttö tyylitiedostossa muun muassa rakenteen leveyden määrittämiseen.

## 3.5 Facebook

Facebook on tällä hetkellä yksi maailman suosituimmista nettisivuista, sillä on yli 800 miljoonaa käyttäjää ympäri maailman, joten kaiken mahdollisen Facebook tuen sisällyttäminen sivustoon tai sovellukseen on ratkaisu, joka lisää suosiota ja tuo uusia käyttäjiä sivulle. (Facebook Statistics 2011) Myös Googlen vastaava palvelu Google+, kasvaa kovaa vauhtia ja sillä oli Social Statisticsin mukaan noin neljä kuukautta julkaisunsa jälkeen (20.10.2011) jo yli 77 000 käyttäjää. (Social Statistics 2011)

Sovelluksen määrittelyvaiheessa päätettiin, että mahdollisimman paljon muissa palveluissa olevaa tietoa ja toiminnallisuutta käytettäisiin sovelluksen toteutuksessa. Pääasiallisiksi palveluiksi valitsimme Facebookin ja Google+:n.

Sovellus voisi toimia Facebookin kanssa suoraan yhteydessä niin, että tapahtumakalenterissa luodut tapahtumat luotaisiin automaattisesti ja ne näkyisivät myös Facebookissa. Tapahtumien synkronointi myös toisinpäin olisi ollut mielenkiintoinen lisä sovellukseen. Tällaisella yhteydellä voitaisiin jatkossa saavuttaa suuret määrät sekä käyttäjiä että varsinkin tapahtumia. Tapahtumakalenteriin sisällytettiin kuitenkin Facebookin ominaisuuksista ainoastaan tykkääminen ja kirjautuminen Facebook-tunnusten avulla, jolloin käyttäjän ei tarvitse tehdä sovellukseen tunnuksia erikseen. Sovellukseen voitaisiin tulevaisuudessa lisätä esimerkiksi henkilökohtainen galleria-ominaisuus, jolla käyttäjän kuvat saataisiin suoraan Facebookista.

Myös muiden vastaavien palveluiden integrointia mietittiin sovelluksen toimintaan, mutta vielä toistaiseksi muut palvelut jäivät sovelluksen ulkopuolelle. Jatkossa tapahtumakalenteriin voitaisiin kuitenkin sisällyttää muun muassa Twitter, jolla sovellus voisi saada lisää näkyvyyttä esimerkiksi Amerikassa. Twitter on siellä yleensä ensimmäisenä mukana sovellusten lisänä.

Sovelluksen tietosisällöistä voitaisiin tulevaisuudessa myös tarjota erilaisia syötteitä. Palveluun voitaisiin toteuttaa muokattava syöte-editori, jolloin käyttäjä voisi luoda sovelluksessa haluamansa syötteen esimerkiksi “Tapahtumat Vaasassa, kategoriana urheilu, vuosina 2011–2012”. Syötteen sisältöä voisi sitten lukea sitä mukaa, kun tapahtumia ilmestyy tapahtumakalenteriin erinäisistä laitteista ja sovelluksista.

Tulevaisuudessa yhteistyö muiden sosiaalisten sivustojen kanssa voisi lisääntyä, ja enemmän dataa voitaisiin synkronoida eri palveluiden kesken. Tapahtumaa luodessa voitaisiin valita, missä eri palveluissa tapahtumasta luodaan “kopio” tai että minkä palvelun sivuilla näytetään ilmoitus kyseisestä tapahtumasta. Näin tapahtumakalenteri muodostuisi luonnolliseksi osaksi sosiaalisten web-palvelujen verkostoa.

### 3.6 Sisällön ryhmittely

Käyttäjä silmäilee sisältöä tietyllä tavalla, sivuston ryhmittelyn on osoitettu vaikuttavan käyttäjän tapaan silmäillä sivustoa sisältöä lukiessaan.

Sisällön ryhmittelyn vaikutusta käyttäjän toimintaan on tutkittu laajasti. Käyttäjän katseen liikkeitä on seurattu lämpökartoilla kuviossa 17. Käyttäjä tutkii katseellaan yleensä ensin sivuston yläosaa vaakasuoralla liikkeellä, siirtyy tämän jälkeen vähän alaspäin ja tutkii sisältöä taas vaakasuoralla liikkeellä. Lopuksi käyttäjä usein silmäilee sivun vasenta reunaa katseellaan pystysuoralla hitaalla ja hyvin systemaattisella liikkeellä.

**Kuvio 17.** Sivustoa lukevan käyttäjän katseen liike kolmella eri sivustolla lämpökartoilla kuvattuna.

(Nielsen 2006)

Näistä kolmesta hyvin yleisestä katseen luomasta liikeradasta saadaan niin sanottu F-Pattern. Tietenkään käyttäjän katse ei liiku täysin näiden linjojen mukaisesti, mutta yleisesti voidaan ajatella käyttäjän katseen liikkuvan suurin piirtein näiden sääntöjen mukaan. Sivuston sisällön rakenne on syytä suunnitella F-Patternia silmällä pitäen,

koska käyttäjä tuntuu toimivan juuri edellä mainitulla tavalla. Tärkeimmät elementit sijoitetaan sivuston yläosan vaakasuoralle alueelle. Sivuston tärkein sisältö sijaitsee sivuston yläosan elementeistä hieman alaspäin, vaakasuoralla alueella. Sisällön ala-osassa tai alapuolella ei sisältö enää voi olla kovin laajalla alueella, sillä käyttäjä ei yksinkertaisesti jaksa silmäillä sisältöä katsellaan sieltä enää kovinkaan laajasti oikeaa laitaa kohti. (F-Shaped Pattern for Reading Web Content 2006)

### 3.7 Värimaailma ja typografia

Sovelluksen layout-suunnitelmissa kävi selväksi, että pääväreinä käytettäisiin mustan ja valkoisen eri sävyjä. Varjojen ja liukuvärien avulla sovelluksen ulkoasuun luotiin syvyyttä ja linjakkuutta. Tehoste väriksi valittu vihreä eroaa muista väripaletin väreistä selkeästi, mikä tuo tapahtumakalenteriin raikkaan tunnelman. Kuviossa 18 on sovellusta varten muodostettu väripaletti.


**Kuvio 18.** Tapahtumakalenterissa käyttämämme pääväripaletti.
(Adobe Kuler 2011)

Logo lähti sovelluksen hyvin ilmeisesti tarkoitustaan kuvaavasta nimestä “Happen”. Happenin kaksi P-kirjainta tuntuivat sellaisilta osilta, joista logoon saisi helposti symmetrisyyttä kääntämällä ensimmäisen P-kirjaimen väärinpäin. Layout tulisi perustumaan yhteen nappiin, joten logo haluttiin nimenomaan nappimaiseen tyyliin. Tehostevärin lisäksi nappiin lisättiin kiiltoa ja muuta tehoste-efektiä kertomaan käyttäjälle, että kyseessä on nimenomaan interaktiivinen elementti. Logoon lisättiin alas-

päin osoittava nuoli, jonka haluttiin korostavan että logoa napsauttamalla aukeaa alasvetovalikko. Kuviossa 19 on tapahtumakalenterissa käytetty lopullinen logo.


**Kuvio 19.** Happen-logo inaktiivisena ja aktiivisena, logon kaikki tuetut koot**.**

Sivuston layout-rakennetta luodessa kiinnitettiin paljon huomiota elementtien sijoitteluun, jonka lisäksi oli kiinnitettävä huomiota white-spacen eli tyhjän tilan käyttöön elementtien ympärillä. On olemassa perusväite, että sivuston elementtiä ei luoda, vaan sille kaiverretaan paikka tyhjän tilan käytöllä. Tyhjän tilan käyttö web-designissa on aivan yhtä tärkeää kuin elementtien käyttö, sen näkymättömyys antaa sille valmiudet toimia suuressa roolissa herättämättä kenenkään huomiota. Tyhjällä tilalla voidaan jakaa sivusto kahtia, työntää sivuston elementtejä lähemmäksi toisiaan tai luoda linja, jota pitkin silmä luontevasti kulkee kohti haluttua sisältöä tai lopputulosta. (Ward ym. 2011, 49–71)

Tyhjän tilan käytöllä voidaankin monesti taistella “normaaleja” web-standardeja vastaan ja pakottaa katse tiettyihin suuntiin poiketen esimerkiksi F-Patternista. Kun minimalistiselle sivustolle on aseteltu vahvat elementit ja tyhjää tilaa on käytetty oikein, silmä kulkee elementistä toiseen täysin mielenkiinnon ohjaamana. (Ward ym. 2011, 49–71)

Sovelluksessa käytettiin yleisesti modernimpia, päätteettömiä kirjasintyyppejä (Sans-serif). Lisäksi haluttiin tietysti varmistaa oikeiden fonttien käyttö useilla eri käyttöjärjestelmillä. Luotiin fonttipino, jossa on kahdeksan eri fonttia: kaksi Linuxia varten, kaksi Windowsia varten ja yksi OS X:ää varten. Lisäksi font-stackin alimmaiseksi laitettiin kolme fonttia, jotka toimivat yleisesti kaikissa käyttöjärjestelmissä.

Web-palveluina tarjottavat palvelut ovat vielä sen verran rajattuja ja epävarmojakin, että tapahtumakalenterin kirjasintyyppimääritykset päätettiin tehdä varmalla tyylitiedostoon määritettävällä usean käyttöjärjestelmän font-stackilla. On aina hyvä olla korvaava vaihtoehto, kun käytetään uusia tekniikoita tai palveluita. Jatkossa font-stack voidaan korvata SCSS:n määritettävillä muutamalla @font-face-palvelun fontilla, jotka toimivat kaikissa käyttöjärjestelmissä riippumatta siitä, mitä fontteja koneeseen on asennettuna.

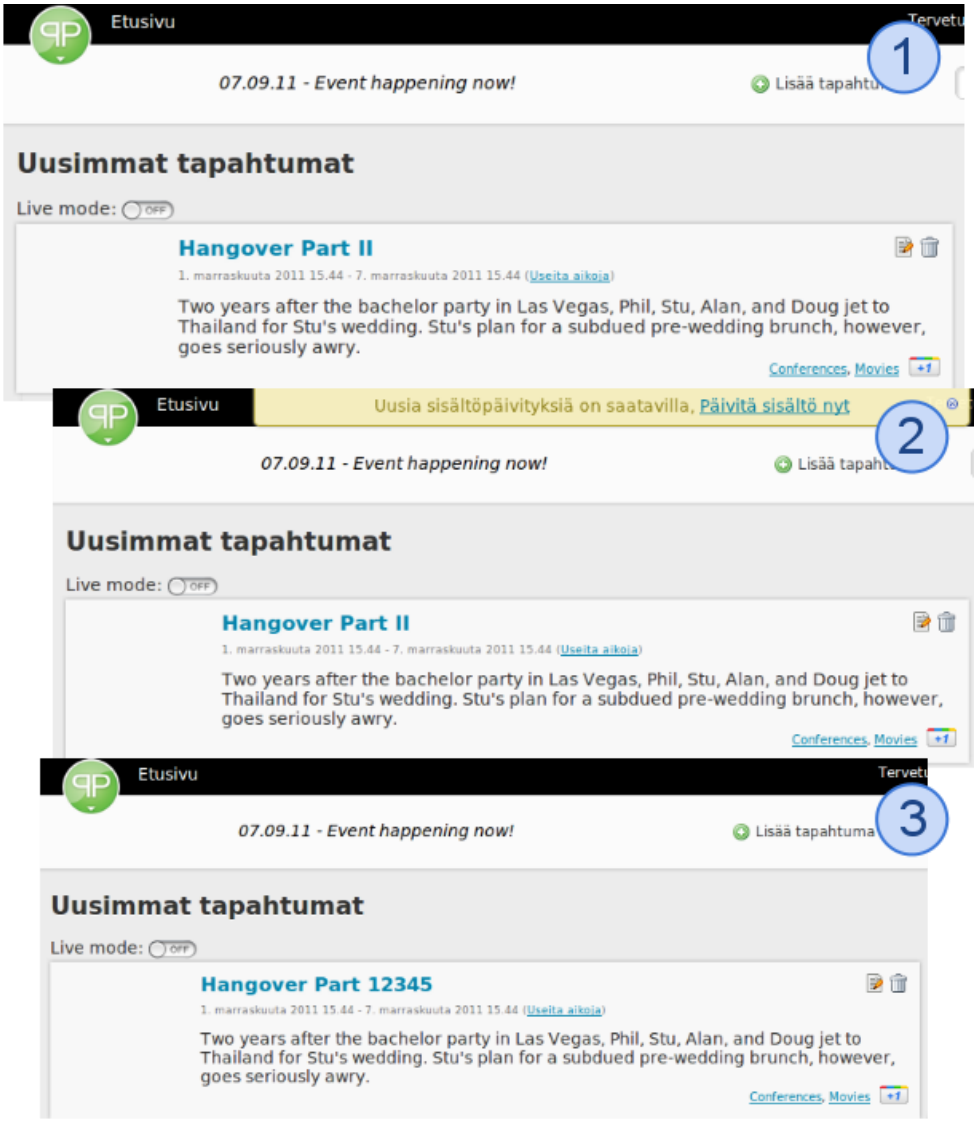
Tulevaisuudessa todennäköisesti @font-face-palvelu kasvaa ja yleistyy. Toiveena on, että omia fonttejaan luovat kirjasintyyppipalvelut ja kirjasintyyppien tekijät, suostuisivat jakamaan fonttinsa julkisesti @font-facen kaltaisten palveluiden käyttöön. Fonttien käyttö voisi näin tulla ilmaiseksi eikä käyttöjärjestelmiin valmiiksi asennettujen fonttien rajallisuus vaikeuttaisi yleispätevän typografiamallin luomista web-sivustoilla.

# 4 REAALIAIKAISUUS KÄYTETTÄVYYDEN TUKENA

Reaaliaikaisuuden käyttäminen sivuston toteutuksessa mahdollistaa uuden sisällön näkymisen sivulla heti kun se on saatavilla, eli käyttäjän ei tarvitse päivittää sivua odottaessaan uuden sisällön saapumista tai halutessaan nähdä esimerkiksi uusimman tapahtuman. Kuviossa 20 kuvataan uuden sisällön näyttäminen käyttäjälle niin, että käyttäjä itse päivittää sivun.

Reaaliaikaisuuden avulla uusin sisältö voidaan näyttää sivulla aina ylimpänä, ensimmäisenä käyttäjän luettavissa. Sivuston sisällön näkyvyyttä suunnitellessa voidaan

siis myös helposti ottaa huomioon se, että sisältöä ei tarvitse tarjota käyttäjälle kuin yhden sivun verran. Käyttäjän ei siis tarvitse vierittää sivua alaspäin, kun uusin sisältö on kuitenkin aina jo heti näkyvillä. Tällaiset sivustot eroavat monesti muista kompaktilla ja tyylikkäällä ulkonäöllään, sillä esimerkiksi taustakuvana voidaan käyttää yhtä kuvaa liukuvärin toistamisen sijasta. Reaaliaikaisuus nähtiin hyvänä ominaisuutena tapahtumakalenterin tapahtumalistauksen toiminnallisuuden vauhdittajana.



**Kuvio 20.** Ilmoituksen näyttäminen käyttäjälle, kun uutta tietoa on saatavilla, ja tietojen näyttäminen.

Toisaalta reaaliaikaisuus voi myös luoda epäselvyyttä käyttäjille, jotka liikkuvat internetissä hitaammin ja luottavat vanhanaikaisiin navigointikäytäntöihin. Kun sisältö vaihtuu koko ajan, voi käyttäjä helposti erehtyä siitä, millä sivulla hän on ja mihin aikaisemmin sivulla näkyneet tapahtumat katosivat.

Reaaliaikaisuus selaimissa on vielä melko uusi asia, mutta tekniikkana se on yleistymässä, sillä sisällön tuoreus ja välitön saatavuus on käyttäjäystävällistä. Esimerkiksi urheilusivuston tulosseurannan pitää toimia sulavasti reaaliajassa ja tehdyt maalit täytyy päivittyä sivulla heti maalin jälkeen ilman, että käyttäjän täytyy päivittää sivua. Myös matkailusivustojen aikatauluosuudet tai lipunmyynti-informaatio on sisältöä, jonka käyttäjä olettaa olevan reaaliaikaista.

Yhtenä mielenkiintoisena tulevaisuuden kehitysvälineenä nähdään Comet. Comet on web-sovellusmalli, joka tarjoaa vaihtoehtoisen ratkaisun perinteiselle reaaliaikaiselle mallille, jossa selain pyytää palvelimelta tietoa. Comet tarjoaa erilaisia tekniikoita, joilla voidaan toteuttaa kommunikointi toiseen suuntaan, eli palvelimelta selaimelle. Comet on käytössä useimmiten nimenomaan tapahtumalähtöisten sovellusten kehityksessä, sillä Cometin tekniikat tarjoavat mahdollisuuden päivittää sivuston osia ja edistää reaaliaikaista asiakas-palvelimen kanssakäymistä. Comet-tekniikat perustuvat niin sanottuun push-tekniikkaan. Se eroaa pull-tekniikasta siten, että palvelin lähettää tiedon selaimelle. Tällä tekniikalla vältetään selainta tekemästä ylimääräisiä kyselyjä palvelimelle, jolloin pahimmassa tapauksessa palvelimen vastauskyky pienenee ja viive kasvaa. (Oxagile 2011)

## 4.1 Node.js

Node.js on palvelimen tapahtumavetoinen JavaScript-ympäristö, joka on rakennettu Googlen v8-JavaScript-moottorin päälle. Node.js:n erikoisuus on tapahtumavetoinen API ja asynkroninen siirräntä. Node.js on tarkoitettu lähinnä skaalautuvien palvelin-

sovellusten ympäristöksi, ja se soveltuukin erinomaisesti pelien ja kollaboratiivisten sovellusten tekemiseen. (Dahl 2011; Finley 2011; O'Dell 2011)

Node.js on suorituskykyinen, sillä se kykenee palvelemaan monia yhteyksiä samanaikaisesti. Muistin kulutus on vähäinen, koska Node.js ei luo säiettä jokaiselle yhteydelle niin kuin perinteisesti on tehty. Node.js-sovelluksien ei tarvitse välittää mahdollisista säikeiden aiheuttamista umpikujista. Node.js:n sisäiset funktiokutsut eivät juuri koskaan suorita siirräntäoperaatiota, joten prosessi ei koskaan ole estetty. Tämän ansiosta suorituskykyisten sovellusten ohjelmointi on suhteellisen vaivatonta. (Dahl 2011)

## 4.2 Socket.IO

Socket.IO tarjoaa rajapinnan, jolla reaaliaikaisten sovellusten toteuttaminen monelle selaimelle olisi mahdollisimman helppoa. Socket.IO implementoi monta eri tapaa, joilla nykypäivän reaaliaikaiset siirtotavat ratkaistaan. Jos selain ei tue HTML5 WebSocket-rajapintaa, niin Socket.IO valitsee kyseiselle selaimelle sopivan ratkaisun. Socket.IO pyrkii käyttämään Flash Socket -rajapintaa toissijaisena vaihtoehtona, koska Flash on yleinen ja käytössä lähes jokaisessa selaimessa. (Adobe 2011) Tarkoituksena olisi tavalla tai toisella simuloida reaaliaikaisuutta, muuttamatta Socket.IO:n asiakasrajapintaa. (Rauch 2011)

HTML5 WebSocket -rajapinta ei kuitenkaan tarjoa kaikkea toiminnallisuutta oletuksena. Socket.IO sisältää aikakatkaisu-, sydämenlyönti- ja yhteyden katkaisu -toiminnallisuudet. Socket.IO pystyy sydämenlyönneillä ("heartbeats") selvittämään, onko yhteyden tila suljettu vai päällä. Palvelimen Socket.IO vaatii, että asiakaspään Socket.IO lähettää sydämenlyönnin palvelimelle joka viidestoista sekunti, ja mikäli tietoa sydämenlyönnistä ei saada 20 sekunnin sisällä, yhteys aikakatkaistaan. (Rauch 2011)

# 5 PROJEKTINHALLINTAMENETELMÄT JA TYÖKALUT

Projektinhallinnan työkaluina käytössä oli muun muassa Git-versionhallinta, Dropbox-tiedostonhallinta, Google Docs -tekstieditori ja tuntiseurantapalvelu Toggl. Skype-pikaviestintä käytettiin etäpalavereiden pitämiseen ja kommunikointiin sekä sovelluksen rakennus- että raportointivaiheessa. Pidimme tärkeänä, että työt saataisiin jaettua tasapuolisesti niin sovelluksen kehitysvaiheessa kuin myös raportointivaiheessa. Hyvillä projektinhallintamenetelmillä ja ennen kaikkea niiden oikealla käytöllä varmistettiin, että projekti saatiin vietyä loppuun asti aikataulun mukaisesti.

## 5.1 XP

XP eli eXtreme Programmingin tavoitteena on saada aikaan laadukkaampia sovelluksia tuottavammin. XP:ssä hyödynnetään monia käytäntöjä, joista kaikkia tulee käyttää yhdessä, sillä ne tukevat toinen toisiaan. XP sisältää neljä arvoa, joita ovat kommunikaatio, yksinkertaisuus, palaute ja rohkeus. Arvojen tarkoituksena on ohjata käyttäjiä oikeaan suuntaan sovelluksen kehityksessä. Arvoksi voidaan myös ajatella kunnioitusta, joka vallitsee tiimin jäsenten keskuudessa. Keskinäinen kunnioitus on tärkeässä osassa onnistuneeseen ja mielekkääseen XP-työskentelyyn. (Beck 2004, 29–35)

Kommunikoinniksi luokitellaan tiimin ja asiakkaiden välinen tiedonkulku. Kommunikoinnin tarkoitus on kertoa sovelluksen muutoksista ja esittää oikeita kysymyksiä osapuolten välillä, jotta vältytään tulemasta väärinymmärretyksi asioiden suhteen ja ei päädytä sovelluksen toimimattomuuteen. (Beck 2004, 29–30)

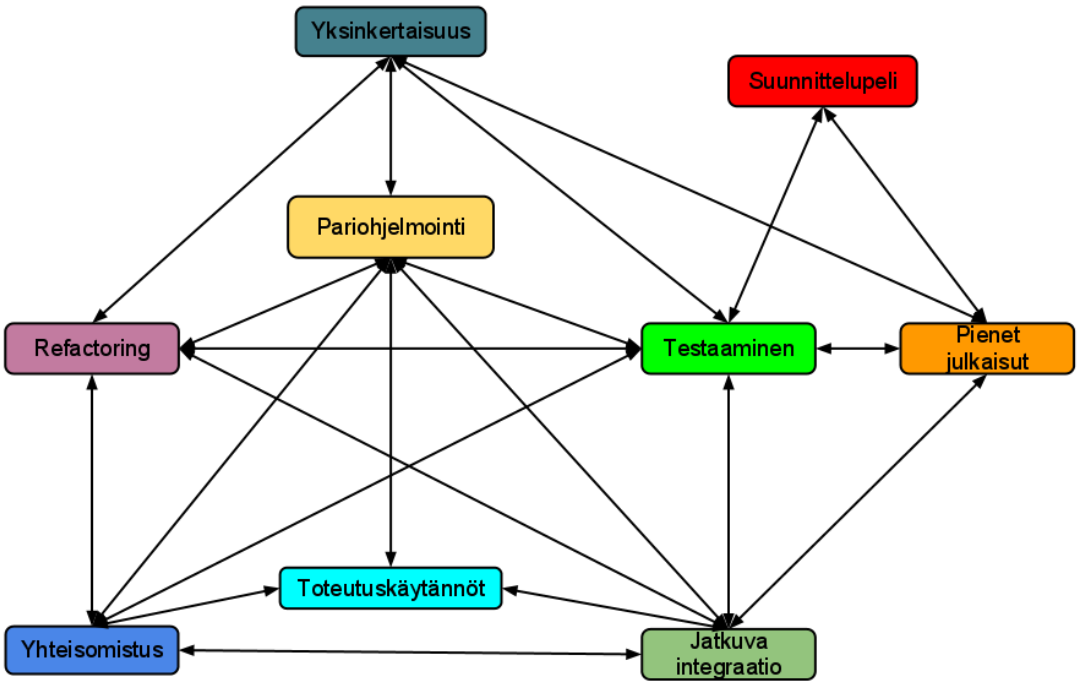
Yksinkertaisuuden merkitys XP:ssä on se, että toiminnallisuudet kehitetään mahdollisimman yksinkertaisesti ilman syvällisempää pohdintaa. Tärkeintä on, että toiminnallisuuden minimivaatimukset saadaan nopeasti toteutettua. Ajatuksena on se, että mikäli toiminnallisuuteen käytetään paljon aikaa, niin voikin olla, että kyseistä toiminnallisuutta ei koskaan oteta mukaan sovellukseen. Mitä enemmän osapuolet kommunikoivat, sitä selvemmin nähdään, mitä tulee tehdä ja ennen kaikkea, mitä ei tulla

tekemään. Mikäli sovellus on yksinkertainen, niin kommunikoinnin tarve on vähäisempää. (Beck 2004, 30–31)

XP:ssä palautetta saadaan tiimiltä, asiakkaalta sekä tekeillä olevalta sovellukselta. Ohjelmoijat tekevät yksikkötestauksia sovelluksen jokaiselle ominaisuuden toiminnolle. Ohjelmoijat saavat testeistä siis konkreettista palautetta sovelluksen tilasta. Konkreettinen palaute toimii yhdessä kommunikaation ja yksinkertaisuuden kanssa. Mitä enemmän palautetta on saatu, sitä helpompi osapuolten on kommunikoida. (Beck 2004, 31–33)

Rohkeutta on poistaa valmiina olevan toiminnallisuuden koodit, johon on käytetty paljon aikaa ja refaktoroida sama toiminnallisuus uudestaan siistimmällä ja vähemmällä koodilla. Konkreettinen palaute tukee rohkeutta uskaltaa kokeilla radikaaleja muutoksia koodiin. (Beck 2004, 33–34)

XP:hen kuuluu useita erilaisia käytäntöjä, jotka ovat ratkaisuiltaan yksinkertaisia käyttää, mutta oleellisia XP:n kannalta. Kuviossa 21 olevalla kaaviolla havainnollistetaan, kuinka XP:n käytännöt ovat toistensa tukena. (Beck 2004, 53–61)




**Kuvio 21.** Kaavio kuvaa, kuinka XP:n käytännöt tukevat toinen toistaan.

(Beck 2004, 70)

Suunnittelupelissä asiakkaalta saadaan sovellukseen tulevat toiminnallisuudet tärkeysjärjestyksessä. Sovelluksen toimittaja antaa asiakkaan kuvaileman sovelluksen toiminnallisuuksista kustannus-, aikataulu- ja työaika-arviot. Asiakas on lopulta se, joka päättää, mitä asioita seuraavaan julkaisuun otetaan mukaan. (Beck 2004, 55–56)

Testaamisen tarkoituksena on, että sovellukselle kirjoitetaan testitapauksia. Kaikkien testien on suoriuduttava onnistuneesti, jotta sovelluksen kehittäminen voisi jatkua. Asiakkaat vastaavasti tekevät käyttötapauksille toiminnallisuustestejä, jotta huomataan käyttötapauksissa ilmeneviä puutteita. (Beck 2004, 57–58)

XP:ssä on tärkeää, että sovelluksen ominaisuuksia julkaistaan pieninä julkaisuina. Pieni julkaisu sisältää ominaisuuden tärkeimmät toiminnallisuusvaatimukset. Tarkoituksena on julkaista toimivia ominaisuuksia ja julkaista niihin myöhemmin päivityksiä lyhyissä jaksoissa. Pienien julkaisujen ideana on saada ominaisuudet nopeasti

käyttäjien käytettäväksi, jotta se saadaan esimerkiksi tuottamaan pääomaa tuotteen omistajalle. (Beck 2004, 56)

Jatkuvassa integraatiossa liitetään testatut toiminnallisuudet toimivaan sovellukseen. Mikäli integraatiossa ilmenee virheitä, niin ne korjataan saman tien, sillä sovelluksen tulee suorittaa kaikki testit onnistuneesti. (Beck 2004, 59–60)

Yksinkertainen rakenne pitää sisällään ensimmäisen version vaadittavat toiminnallisuudet ja ominaisuudet. Yksinkertaisesta rakenteesta huolimatta sovelluksen tulee suoriutua kaikista testeistä. Sovelluksen tulisi sisältää mahdollisimman vähän luokkia ja metodeja. (Beck 2004, 57)

Kaikki sovelluskoodit kirjoitetaan pareittain, josta käytetään nimitystä pariohjelmointi. Pariohjelmoinnissa kaksi ihmistä istuu saman tietokoneen ääressä, ja toinen katsoo, kun toinen ohjelmoi. Ohjelmoija toteuttaa jotakin sovelluksen osaa parhaiten katsomallaan tavalla, kun toinen ohjelmoijista miettii asiaa syvällisemmin: tuleeko tämä lähestymistapa toimimaan. (Beck 2004, 58–59) Pariohjelmoinnin väitetään lyhentävän aikataulua, sillä pareittain ohjelmoiminen on nopeampaa ja koodi sisältää vähemmän virheitä kuin yksin ohjelmoitaessa. Pareja tulisi vaihdella säännöllisesti, jotta ohjelmoijat oppisivat tuntemaan sovelluksen muitakin osia. (McConnell 2004, 483–484)

Refaktorointi on käytäntö, jossa mietitään koodin uudelleen kirjoittamista. Ominaisuuden toteuttamisen jälkeen mietitään, voisiko ominaisuuden toteuttaa vähemmällä ja siistimmällä koodilla, kuitenkaan muuttamatta ominaisuuden toiminnallisuutta. (Beck 2004, 58)

Toteutuskäytännöt ovat XP-tiimin yleisten pelisääntöjen määrittelemistä. Jokaisella tiimin jäsenellä tulisi olla lähes samanlaiset työskentelytavat, jotta työ sujuisi yhtenäisesti. Koodin kirjoittamisenkin tulisi jokaisella olla samannäköistä, jotta toisen tekemää koodia olisi helppo lukea ja ylläpitää. (Beck 2004, 61)

Kaikilla XP-tiimin jäsenillä on yhteisomistus sovelluksen koodeihin ja he ovat yhdessä vastuussa niiden käytöstä. Yhteisomistus vaatii sovelluskoodin integrointia lyhyessä ajassa, jotta ohjelmakoodissa ei synny konflikteja. Yhteisomistuksessa korostetaan toteutuskäytännöissä määriteltyjen pelisääntöjen käyttämistä, jotta koodi on ymmärrettävää. Mikäli yhteisomistusta ei oteta huomioon sovelluksen kehityksessä, niin sovelluksen kehitys luultavasti hidastuu huomattavasti. (Beck 2004, 59)

Tapahtumakalenterin toteutuksessa lähinnä harjoiteltiin XP:n käytäntöjä. XP:n yksi kehittävimmistä asioista oli se, miten jokainen on vastuussa joka osa-alueesta ja jokaisella on tarvittava osaaminen projektin suorittamiseen. Tapahtumakalenterin koodista oli yhteisomistus, joka mahdollisti toisen henkilön tekemän koodin parantamisen. Yhteisomistus poistaa mahdollisuuden rajata henkilöitä vain tiettyjen ominaisuuksien kehittämiseen. Tapahtumakalenterin integraatio tapahtui lataamalla uusimmat koodit versionhallinnasta. Erityisen tärkeää oli testata sovellus ennen uusien muutosten viemistä versionhallintaan, jotta tapahtumakalenterin koodi pysyi ehjänä.

## 5.2 Scrum

Scrum on ketterässä ohjelmistokehityksessä yleisesti käytetty projektinhallintamenetelmä, mutta sitä voidaan myös käyttää muissakin projekteissa. Scrumin avulla pystytään keskittymään aina projektin tärkeimpiin asioihin. Scrum-projektissa on kyse yhteistyöstä ja kommunikaatiosta tiimin sisällä ja tiimin ja tuotteen omistajan välillä. Tuotteen omistajan palaute on tärkeä osa Scrumia, jotta toteutettaisiin juuri niitä asioita, joita tuotteen omistaja haluaa sovellukseen. (Scrum Alliance 2011) Scrumin rooleihin kuuluu tuotteen omistaja, tiimi ja Scrum Master. (Schwaber 2004, 28)

Tuotteen omistaja vastaa tuotteen vaatimusmäärittelystä, projektin kannattavuudesta, julkaisujen sisällön määrittelemisestä ja aikataulutuksesta. Tuotteen omistajan tehtävänä on varmistaa, että sovelluksen kaikkein tärkein toiminnallisuus kehitetään ensin.

Tuotteen omistaja tekee tuotteen työlistan, josta selviää tuotteelle asetetut vaatimukset ja ominaisuudet. (Poimala, Heikniemi & Blåfield 2011)

Scrum Master on tiimin ohjaaja, jonka tehtäviin kuuluu opettaa Scrumin toimintaperiaatteet kaikille projektiin osallistujille. Scrum Master tarkkailee myös, että sprintin tavoitteet tulee täytettyä ja että kaikki noudattavat Scrumia sen vaatimalla tavalla. (Schwaber 2004, 19) Tiimin jäsenet raportoivat päivittäin töiden hidastumiseen vaikuttavista tekijöistä hänelle. Scrum Masterin tehtävänä on varmistaa ja pitää huoli siitä, että tiimityöskentely on tuottavaa jokaisena päivänä, selvittää esiin tulleita ongelmia ja pitää päivittäiset seurantapalaverit. (Poimala ym. 2011)

Tiimi koostuu henkilöistä, jotka ovat mukana tekemässä työtä. Tiimin henkilöillä on tarvittava osaaminen suorittaa työ alusta loppuun. Sillä tarkoitetaan sitä, että jokainen henkilö osaa toimia jokaisessa sovelluksen kehitysvaiheessa. Jokainen jäsen on tärkeä projektin onnistumisen kannalta. Kaikki yhdessä vastaavat tuotteen kehityksestä tasapuolisesti, eli yksittäinen henkilö ei voi vastata jostakin tietystä osa-alueesta. Tiimissä ihmiset tekevät niitä asioita, joita osaavat parhaiten, mutta tiimi kuitenkin yhdessä vastaa tehtävien jaosta, jotta vältytään työtehtävien siirtämisestä muille. (Schwaber 2004, 19)

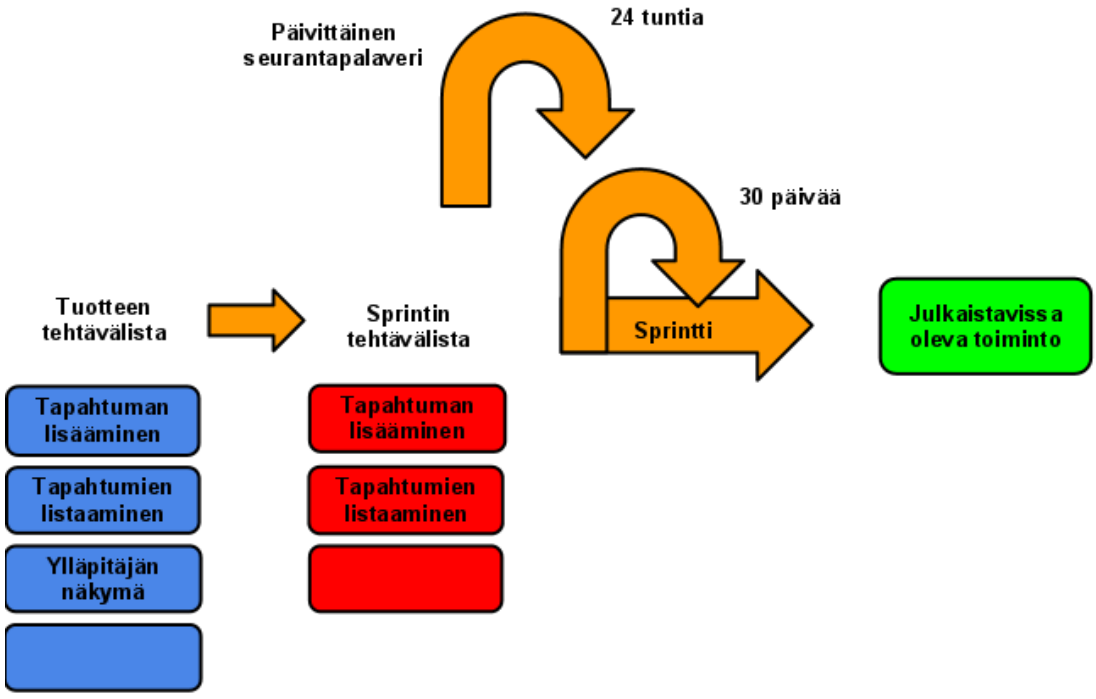
Scrumissa toiminta on syklimäistä. Syklien tärkeimpiä vaiheita ovat sprintti ja päivittäinen seurantapalaveri. Sprinttiä voidaan kutsua jaksoksi, jonka aikana mukaan otetut työtehtävät saadaan valmiiksi. Sprintin kesto on yleisesti yksi kuukausi eli 30 päivää. Sprintin sisältö suunnitellaan etukäteen ennen sprintin alkua. Sprintin alussa pidettävää suunnittelupalaveria on ohjaamassa tuotteen omistaja. Tuotteen omistaja kertoo työlistan tärkeimmästä toiminnallisuudesta, jonka jälkeen tiimiläiset voivat esittää kysymyksiä, jotta he sitten osaavat toteuttaa toiminnallisuudet itsenäisesti ryhmässä. Sprintin tehtäviksi valitaan sellaisia asioita, joita pidetään sillä hetkellä tärkeimpinä projektin kehitystä ajatellen ja jotka ajatellaan saatavan valmiiksi sprintin aikana. Kehitystiimi esittelee sprintin lopussa, mitä sprintin aikana on saavutettu. (Schwaber 2004, 17)

Kehitystiimi ja Scrum Master kokoontuvat päivittäiseen seurantapalaveriin, jonka kesto on noin 15 minuuttia. Tiimin jokaiselta jäseneltä kysytään samat kysymykset jokaisessa päivittäisessä seurantapalaverissa:

- Mitä olet tehnyt edellisen päivittäisen seurantapalaverin jälkeen?
- Mitä aiot tehdä ennen seuraavaa päivittäistä seurantapalaveria?
- Onkin jokin hankaloittanut työtäsi?

(Schwaber 2004, 20)

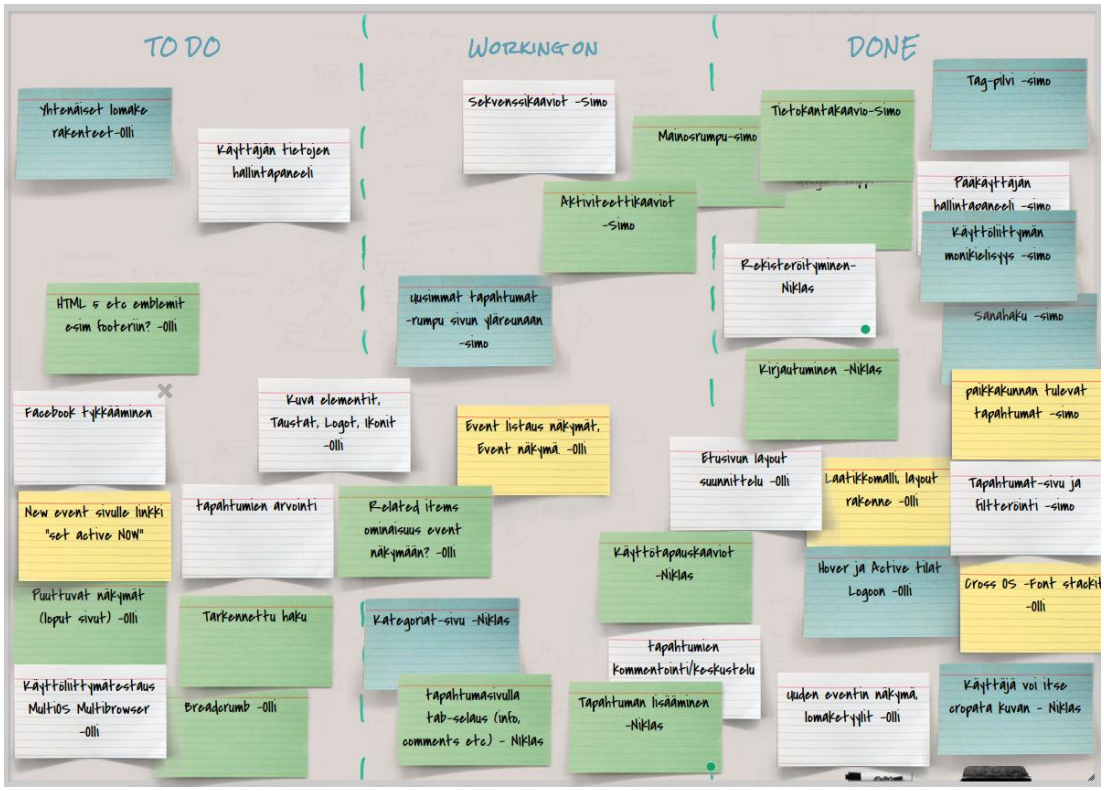
Päivittäisessä seurantapalaverissa ei keskitytä muihin asioihin. Tarkoituksena on päästä ajan tasalle työn etenemisessä. Päivittäisen seurantapalaverin jälkeen voidaan yhdessä käydä läpi mahdollisia työn hidastajia. Kuviossa 22 on kuvattu sprintin kiertokulku kaavion avulla.



**Kuvio 22.** Sprintin kiertokulku.

(Schwaber 2004, 17–27)

Scrumin käyttö projektin hallintamenetelmänä ohjasi tapahtumakalenterin kehitystä tehokkaasti eteenpäin ja koko ajan oli selvää, mitä seuraavaksi tehdään. Tapahtumakalenterin kehityksessä käytimme Scrumia omiin tarpeisiimme mukautettuna. Meillä oli käytössä Scrum-taulu, josta kävi ilmi, mitä ominaisuuksia tapahtumakalenteriin tullaan toteuttamaan, mitkä ominaisuudet ovat työn alla ja mitkä ominaisuudet on jo valmiita. Scrum-taulun käyttö helpotti tapahtumakalenterin kokonaisuuden hahmottamista sisällöllisesti ja aikataulullisesti. Kuviossa 23 on käyttämämme Scrum-taulu noin puolessa välissä sovelluksemme kehitystä.




**Kuvio 23.** Tapahtumakalenterin Scrum-taulu.

(Scrumblr 2011)

### 5.3 Git-versionhallinta

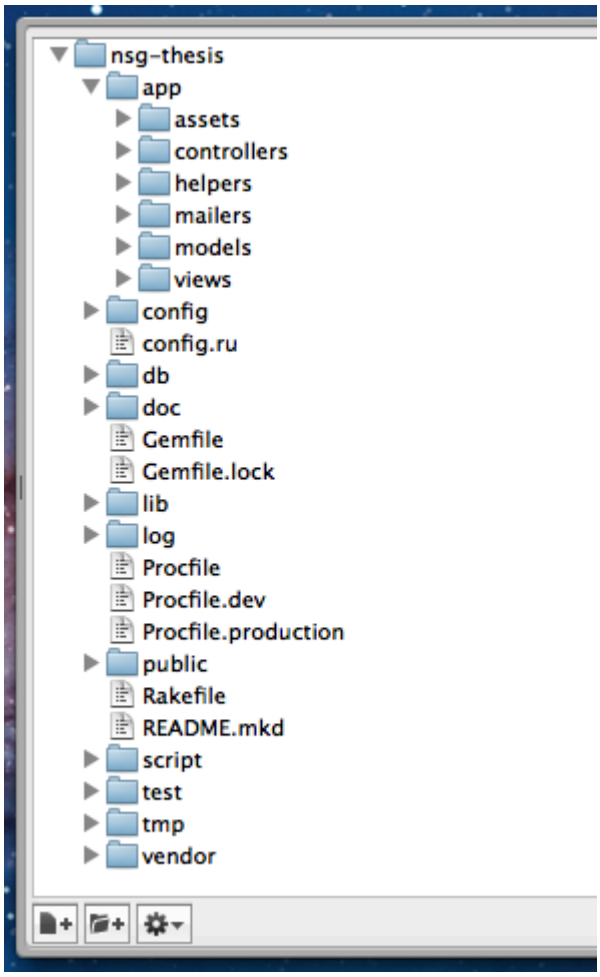
Git-versionhallinta on apuväline projektin lähdekoodiin tehtävien muutosten seuraamiseen. Linux-käyttöjärjestelmien isänä tunnettu Linus Torvalds suunnitteli ja pani Git-versionhallinnan kehityksen alulle pääasiassa Linux-ytimen kehitystyötä varten. (Swicegood 2008, 10) Tapahtumakalenterin ohjelmointivaiheessa päädyttiin käyttämään Git-versionhallintaa, koska sitä on käytetty muissakin yhteisissä projekteissa ja sen peruskäyttö oli jokaiselle tuttua.

Enimmäkseen käyttämämme Gitin perusominaisuudet olivat tiedoston lisääminen (git add), oman säilytyspaikan muutosten tarkastelu (git status), muutoksen hyväksyminen (git commit), viimeisimpien muutosten hakeminen säilytyspaikasta (git pull) ja muutosten vieminen säilytyspaikkaan (git push).

Tapahtumakalenterin lähdekoodia säilytettiin ja jaettiin GitHubissa. Englanniksi puhutaan sanasta “Repository”, jolla tarkoitetaan projektin tiedostojen säilytyspaikkaa. Säilytyspaikassa versionhallinta seuraa kaikkia tapahtumakalenteriin tehtyjä muutoksia, eli mitä tiedostoa tekijä on muuttanut, mitä muutoksia siihen on tehty, viesti miksi muutos on tehty ja kuka muutoksen on tehnyt. Säilytyspaikka pitää sisällään täydellisen historian ja tarkistuksen seurantaominaisuudet. (Swicegood 2008, 16–17)

Kaikilla sovelluksen kehittäjillä oli kopiot säilytyspaikan tiedostoista työasemien paikallisessa säilytyspaikassa. Sitä mukaa, kun uusia ominaisuuksia tai muutoksia olemassa olevaan koodiin tehtiin, niin muutokset hyväksyttiin paikalliseen säilytyspaikkaan. Paikallisesta säilytyspaikasta muutokset lähetettiin GitHubissa olevaan säilytyspaikkaan, jotta jokaisella olisi aina viimeisin versio toimivasta sovelluksesta. Mikäli muutoksia ei lähetetä ja työnjaossa on epäselvyyksiä, niin joku muu saattaa tehdä saman muutoksen.

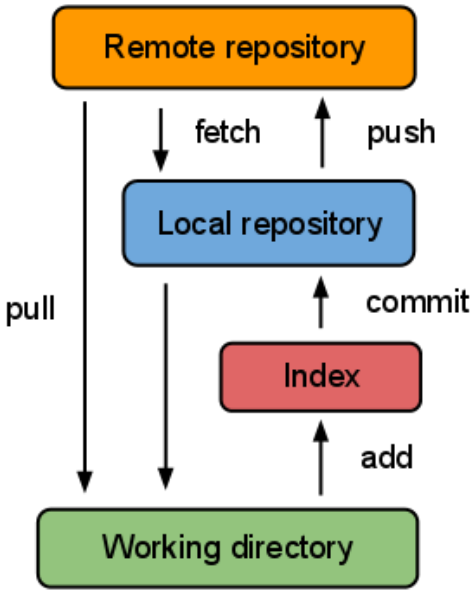
Working Tree, eli sovelluksen puurakenne on käyttäjän näkymä säilytyspaikan tiedostoista. Puurakenne sisältää kaikki sovelluksen tiedostot, muun muassa lähdekoodin, dokumentit, kuvat ja testit. (Swicegood 2008, 18) Kuviossa 24 on esitetty tapahtumakalenterin puurakenne.




**Kuvio 24.** Tapahtumakalenterin säilytyspaikan puurakenne.

Branch eli kehityshaara on tapa ylläpitää vaihtoehtoista historiaa sovelluksessa. Gitin päähaara on nimeltään Master. Gitiä voidaan käyttää tehokkaasti myös sen päähaaralla, mutta vielä tehokkaammin sivuhaarojen avulla. Sovelluksen ohjelmoinnissa uusia sivuhaaroja saatetaan luoda, kun kehitetään uusia ja laajempia ominaisuuksia. Hyvä puoli ominaisuuden kehittämisessä erillisessä sivuhaarassa on, että voidaan tehdä

erilaisia kokeiluja ja muutoksia ilman, että se muuttaa toimivaa sovellusta. Kun ominaisuus on saatu valmiiksi ja se on testattu, niin se liitetään päähaaraan. (Swicegood 2008, 65–67)




**Kuvio 25.** Datan liikkuminen sovelluksen säilytyspaikassa.

(Git Software 2011)

Kuviosta 25 saa selkeän kuvan, kuinka data liikkuu sovelluksen säilytyspaikassa. ”Working directory” on käyttäjän työasemalla oleva työkansio, jossa muutokset sovelluksen tiedostoille tehdään. ”Remote repository” kuvaa tapahtumakalenterin säilytyspaikkaa GitHubissa. Käyttäjä lataa omaan työkansioonsa tapahtumakalenterin ajan tasalla olevat tiedostot. ”Local repository” kuvaa käyttäjän paikallista säilytyspaikkaa tapahtumakalenterin tiedostoille. Myös paikalliseen säilytyspaikkaan haetaan ajan tasalla olevat tiedostot sellaisenaan, eli Git ei yhdistä tiedostoja käyttäjän muutosten kanssa. Kun tehdään muutoksia tapahtumakalenterin tiedostoille, niin muokatut tiedostot näkyvät työkansiossa ja odottavat käyttäjän toimenpiteitä. Kun käyttäjä on

tehnyt tarvittavat muutokset, niin sen hetkisen tilan tiedostot lisätään väliaikaiseen muistiin (Index) odottamaan tulevaa muutosten hyväksymistä paikalliseen säilytyspaikkaan. Käyttäjä hyväksyy muutokset paikalliseen säilytyspaikkaan, josta muutokset lähetetään GitHubissa sijaitsevaan säilytyspaikkaan.

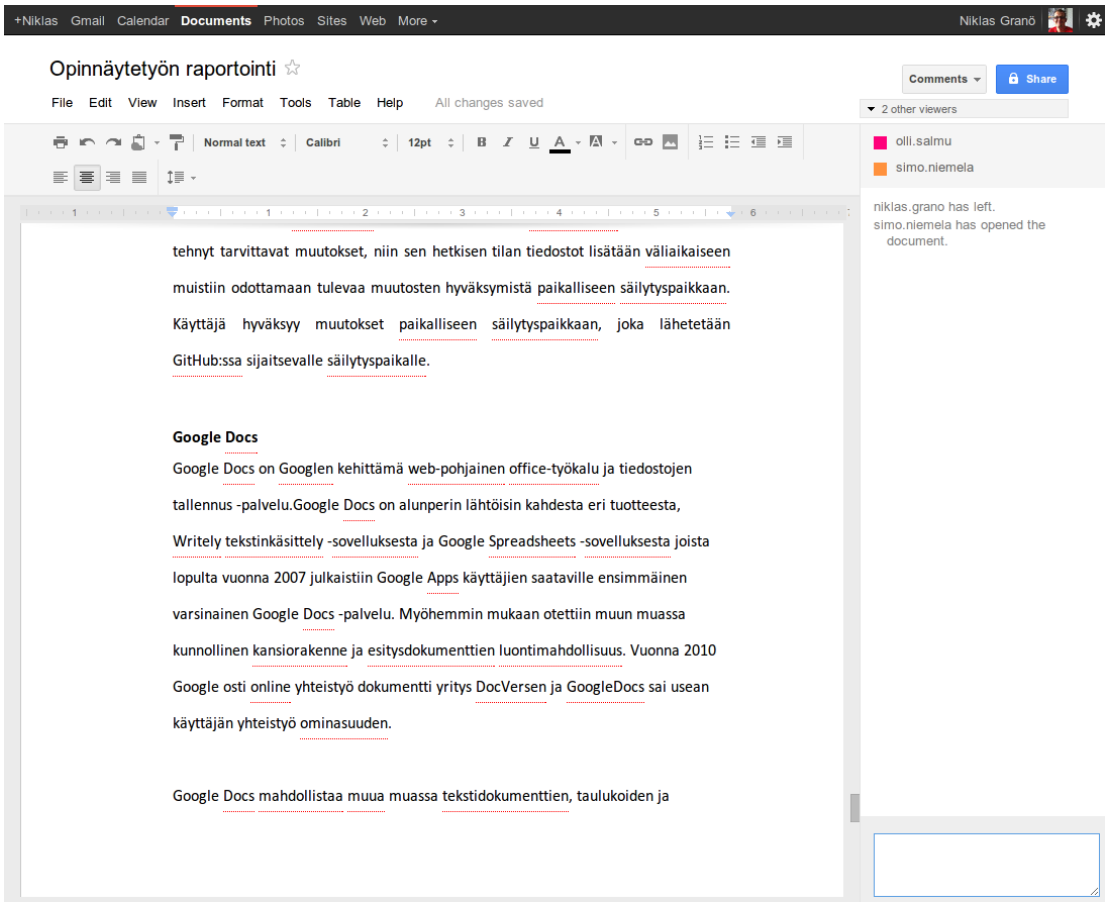
## 5.4 Google Docs

Google Docs on Googlen kehittämä web-pohjainen tekstin-, taulukoiden ja esitysten hallintatyökalu sekä tiedostojen tallennus-palvelu. Google Docs on lähtöisin kahdesta eri tuotteesta, Writely tekstinkäsittely-sovelluksesta ja Google Spreadsheets-sovelluksesta, joista vuonna 2007 julkaistiin Google Apps -käyttäjien saataville ensimmäinen varsinainen Google Docs -palvelu. Myöhemmin mukaan otettiin muun muassa kunnollinen kansiorakenne ja esitysdokumenttien luontimahdollisuus. Vuonna 2010 Google osti DocVersen ja Google Docs sai usean käyttäjän yhteistyöominaisuuden. (Google Docs 2011)

Google Docs mahdollistaa muun muassa tekstidokumenttien, taulukoiden ja PowerPoint-tyylisten esitysten luomisen. Luodut dokumentit voidaan helposti ladata sovelluksesta käyttäjän koneelle useissa eri tiedostomuodoissa. Kaikki dokumentit tallentuvat automaattisesti Googlen palvelimille, jossa käyttäjälle tarjotaan yksi gigatavu tallennustilaa ilmaiseksi. Maksullisena palveluna Google mahdollistaa jopa 16 teratavun tallennuskapasiteetin. (Google Docs 2011)

Googlen kehittämää Google Docs palvelua käytettiin pidempien dokumenttien kirjoittamiseen, koska se mahdollisti dokumenttien vaivattoman kollaboratiivisen kirjoittamisen. Google Docsin web-käyttöliittymä sopi projektityöskentelyyn hyvin, sillä töitä tehtiin yleisesti kukin omalta koneeltamme yhtäaikaisesti. Palvelu hoitaa sekä dokumentin tallennuksen että oikeaan version päivittämisen automaattisesti, joten usean käyttäjän yhtäaikainen työskentely sujuu saumattomasti. Käytettävissä on normaalien formatointityökalujen lisäksi muun muassa tekstin kommentointi, linkkien

lisäämismahdollisuus ja käyttäjien välinen keskustelutoiminto. Google Docsia käytettiin pääasiassa raportin kirjoittamiseen, mutta sillä piirrettiin myös projektin toimintaan liittyviä kaavioita. Kuviossa 26 on Google Docsin dokumentin kirjoitusnäkymä, vasemmassa laidassa reaaliaikainen keskustelualue. (Google Docs 2011)

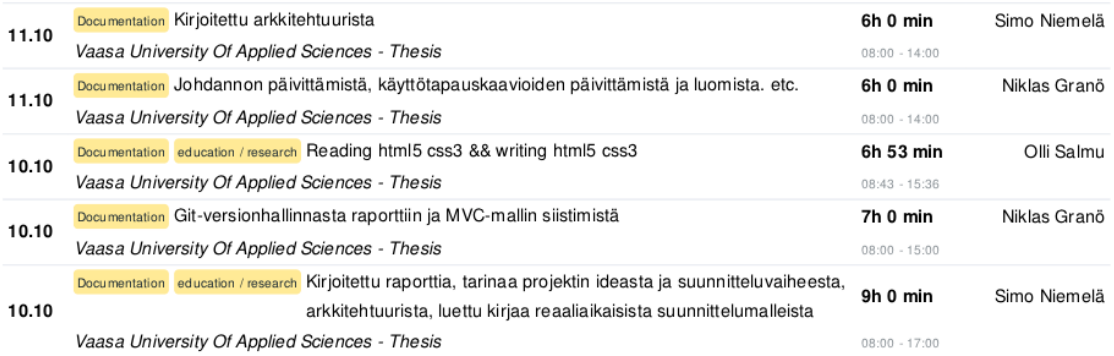



**Kuvio 26.** Google Docs -näkymä.

## 5.5 Tuntiseuranta

Tuntiseurannan työkaluna käytettiin ennestään tuttua tuntiseurantajärjestelmää Togglia. Togglia on käytetty useissa aikaisemmissa projekteissa ja todettu sen olevan käytännöllinen muutaman henkilön projekteissa. Toggl tarjosi tuntiseurannan perusominaisuudet ilmaiseksi, jotka riittivät tähän projektiin hyvin. Toggl valittiin tuntiseurantajärjestelmäksi, koska sillä saatiin luotua projekteja, joihin voitiin liittää useita

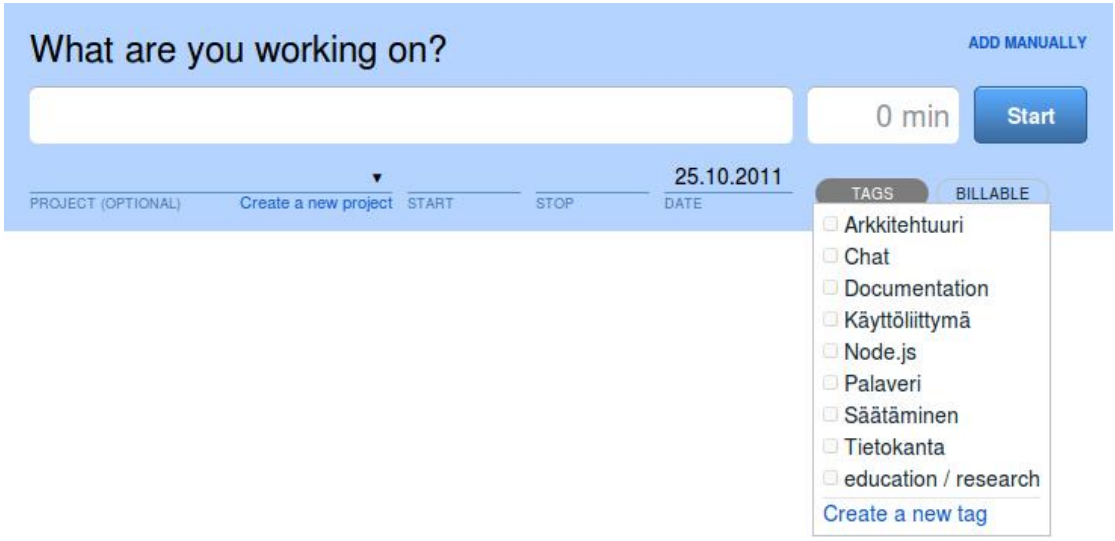
osallistujia (työntekijöitä) ja projektin tuntimääriä ja jakautumisia voitiin seurata tarkoilla kuvion 27 mukaisilla raporteilla. (Toggl 2011)



| | | | |
|---|---|---|---|
| 11.10 | Documentation Kirjoitettu arkkitehtuurista<br>*Vaasa University Of Applied Sciences - Thesis* | **6h 0 min**<br>08:00 - 14:00 | Simo Niemelä |
| 11.10 | Documentation Johdannon päivittämistä, käyttötapauskaavioiden päivittämistä ja luomista. etc.<br>*Vaasa University Of Applied Sciences - Thesis* | **6h 0 min**<br>08:00 - 14:00 | Niklas Granö |
| 10.10 | Documentation education / research Reading html5 css3 && writing html5 css3<br>*Vaasa University Of Applied Sciences - Thesis* | **6h 53 min**<br>08:43 - 15:36 | Olli Salmu |
| 10.10 | Documentation Git-versionhallinnasta raporttiin ja MVC-mallin siistimistä<br>*Vaasa University Of Applied Sciences - Thesis* | **7h 0 min**<br>08:00 - 15:00 | Niklas Granö |
| 10.10 | Documentation education / research Kirjoitettu raporttia, tarinaa projektin ideasta ja suunnitteluvaiheesta, arkkitehtuurista, luettu kirjaa reaaliaikaisista suunnittelumalleista<br>*Vaasa University Of Applied Sciences - Thesis* | **9h 0 min**<br>08:00 - 17:00 | Simo Niemelä |

**Kuvio 27.** Osa tuntiseurantajärjestelmän luomasta pdf-raportista.

Kuviossa 28 näkyvä Togglin käytännöllinen ja helposti omaksuttava ulkoasu oli myös tärkeä seikka valintaperusteena. Sovelluksesta on saatavilla web-sovelluksen lisäksi sekä työpöytä- että mobiiliversio, joten sen käyttö on mahdollista missä ja milloin tahansa. Projektin etenemistä voitiin seurata muun muassa henkilöittäin, josta kävi ilmi tuntityömäärät, suoritetut työtehtävät ja henkilön tehokkuusprosentti projektissa. Raportoituihin työtehtäviin saatiin myös lisättyä tageja, jotka helpottivat projektin tietyn osa-alueen kehityksen seuraamista. Voitiin tutkia esimerkiksi viime viikon työtuntimääriä tagilla “käyttöliittymät”. Raportit saatiin myös tarvittaessa tuotua järjestelmästä ulos Excel- tai pdf-muodossa. (Toggl 2011)

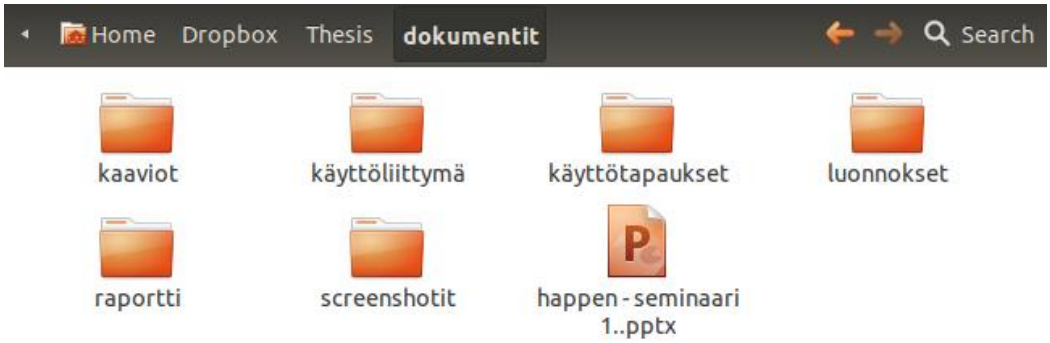



**Kuvio 28.** Työtehtävän ja ajankäytön merkitseminen tuntiseurantajärjestelmän web-käyttöliittymässä.

## 5.6 Dropbox

Dropbox on pilvessä toimiva tiedostonhallintasovellus, jonka ansiosta tiedostot kulkevat aina kätevästi mukana. Dropboxissa voidaan jakaa tiedostoja ja kansioita muiden käyttäjien kanssa, joka mahdollistaa yhteistyön samojen tiedostojen ja projektien parissa. Dropboxin sisällön voi myös ladata omalle työasemalle, jolloin tiedostot ovat käytössä myös offline-tilassa. Silloin käyttöön tulee Dropbox-kansio, joka toimii samalla tavalla kuin käyttöjärjestelmän kansiot. Ero on siinä, että Dropbox-kansiossa olevat tiedostot synkronoituvat palvelimen kanssa, jos Dropbox-sovellus on päällä, eli sekä koneella oleva Dropbox-kansio sekä palvelimella oleva kansio sisältävät samat tiedot. Dropbox on viisas synkronoinnin suhteen, sillä se päivittää vain muuttuneet tiedostot palvelimelle, eikä koko Dropboxin sisältöä. Dropboxista on saatavilla myös mobiiliversio useimmille älypuhelimille. Puhelimella otetut valokuvat ja kuvatut videot voi helposti tallentaa Dropboxiin ja jakaa niitä haluttujen henkilöiden kanssa. (Dropbox 2011)

Dropboxissa säilytettiin tapahtumakalenterin käyttöliittymäsuunnitelmia, käyttötapauskaavioita ja monia muita tärkeitä tiedostoja, joista kävi ilmi sovelluksen rakenne ja toiminnallisuudet. Dropbox ilmoitti aina projektikansioon tehdyistä uusista muutoksista, joka helpotti pysymään ajan tasalla muiden tekemissä muutoksissa. Kuviossa 29 näkymä Dropbox–kansiosta.




**Kuvio 29.** Ubuntu-käyttöjärjestelmän Dropbox-näkymä yhteiskäytössä olevista tiedostoista.

# 6 KRIITTINEN TARKASTELU

**Samanaikaisten tietokantaoperaatioiden ja huomautusviestien lähetysten määrä.**

Rails- ja Node.js-sovellusten välinen kommunikointi tapahtuu sillan kautta, ja viestit lähetetään yhdestä säikeestä Rails-sovelluksen sillasta. Tämä tarkoittaa sitä, että jos monta samanaikaista tietokantaoperaatiota suoritetaan, voi Rails-sovellus lopettaa toimintansa ja jotkut huomausviestit jäädä lähettämättä. Ongelman välttämiseksi tulisi käyttää jono-tyyppistä toteutusta sillassa siten, että jokainen lähetettävä viesti lisätään jonoon ja lähetyssäie poimii jonosta uuden viestin, kun edellinen on lähetetty. Hyvä toteutusvaihtoehto viestien lähetykseen voisi olla Message Queuing -suunnittelumalli, jota käytetään useasti juuri tämän tyyppisiin ongelmiin. Mallissa toimii kaksi säiettä, yksi lisää viestejä pinoon ja toinen säie poimii viestin pinosta suoritettavaksi (Douglas 2003, 212–213).

**Navigoinnin ripeys tapahtuma sivun suodatus-toiminnossa.**

Tapahtuma-sivulla tiedot haetaan tietokannasta aina, kun suodatuslinkkiä on napsautettu. Tämä tuntuu hitaalta, kun linkkejä napsautetaan nopeammin toistuvasti. Kun samaa linkkiä napsautetaan, niin data haetaan heti uudelleen. Käyttäjänä huomaa selvästi viiveen, kun sisältö piilotetaan ja näytetään. Ratkaisuna tähän ongelmaan voitaisiin käyttää hypoteettista JavaScript-koodissa olevaa muuttujaa, moniulotteista taulukkoa, johon tallennetaan Ajax-pyynnön jälkeen saatu data. Tämä data tallennetaan taulukkoon avaimella, josta ilmenee, mille suodatuslinkille data kuuluu. Kun suodatuslinkkiä napsautetaan, niin ensimmäiseksi katsotaan, onko dataa olemassa taulukkomuuttujassa suodatuslinkin nimellä. Jos dataa esiintyi muuttujassa, niin se esitetään tästä tietolähteestä. Mikäli taulukkomuuttujassa ei ollut dataa, suoritetaan Ajax-pyyntö normaalisti, jonka jälkeen data tallennetaan taulukkomuuttujaan.

**Tietokantakyselyjen määrästä johtuva sivuston hidastuminen.**

Nykyisiä tietokantakyselyitä ei ole optimoitu riittävästi. Tapahtumalistat aiheuttavat suuren määrän kyselyitä. Jokaista tapahtumaa kohden haetaan sen tiedot, tapahtuman kategoriat, tapahtuman kuva ja tuleva näytösaika. Tauluista ”kategoriat” ja ”kuvat” haetaan tietoa välitaulun kautta. Kehityspalvelimella neljän tapahtuman tietokantahaut kestävät keskimäärin yhteensä 50 millisekuntia, kesto saisi olla ehkä noin 10 ms. Ratkaisu voisi olla HTTP:n Expire- tai ETag-otsikot, jolloin tietokantaoperaatioita ei suoritettaisi, vaan näkymä palautettaisiin välimuistista, jos se ei ole muuttunut.

**Mainosrummun naiivius.**

Sivun oikeassa laidassa oleva mainosrumpu alkaa pyöriä alusta aina, kun sivu päivitetään, mikä ei ole kovin hyvä asia mainostajien kannalta. Mainosrummun tila tulisi tallentaa tietokantaan tai muistiin, jotta tiedettäisiin, mistä mainoksesta aloitetaan, kun sivu on päivitetty. Tietokannan täytyy olla suorituskykyinen, koska mainoksen tila päivitetään ja luetaan joka kerta kun se vaihtuu, eli muutaman sekunnin välein.

**Tapahtumien tarkistaminen ja siitä aiheutuva viive käyttäjän kannalta, ja minkä takia kommentteja ei tarkisteta.**

Käyttäjä saattaa kokea tapahtumien hyväksyttämisen ylläpitäjällä tarpeettomana ja aikaa vievänä prosessina. Käyttäjän voisi olettaa hyväksyvän tapahtumien tarkistamisen ylläpidolla, mikäli ylläpito onnistuu pitämään viiveen lyhyenä. Käyttäjän voi kuvitella ajattelevan, eikö tapahtumaa voitaisi tarkistaa vasta sen jälkeen, kun se on julkaistu. Mikäli käyttäjien lisäämät tapahtumat julkaistaisiin heti tapahtuman tallennuksen jälkeen, niin myös aiheettomia ilmoituksia pääsee sovellukseen.

Suurin osa käyttäjien lisäämistä tapahtumista on kuitenkin aiheellisia. Aiheettomien ja loukkaavien viestien julkaisemiselta halutaan kuitenkin välttyä. Tapahtumakalenteri halutaan pitää siistinä ja kielellisesti korrektina sivustona. Tapahtumakalenterin suunnitteluvaiheessa mietittiin, miten tapahtumien tarkistaminen voitaisiin toteuttaa mahdollisimman käyttäjäystävällisesti. Vaihtoehtona olisi ollut toteuttaa niin sanottu kirosanasuodatin, joka pitäisi sisällään listan kielletyistä sanoista. Suodattimen ideana olisi ollut tarkistaa tapahtuma kiellettyjen sanojen osalta tapahtuman tallennuksen yhteydessä. Kirosanasuodatin olisi ollut hyvä ratkaisu, mutta monikielisyys vaikeuttaisi kirosanasuodattimen ylläpitoa.

Kirjautuneet käyttäjät voivat kommentoida tapahtumia vapaasti. Kommentteihin ei mietitty eikä edes haluttu minkäänlaisia tarkistuksia, vaan tapahtumien kommentointi haluttiin pitää vapaana keskusteluna. Ylläpitäjällä on kuitenkin mahdollisuus muokata ja poistaa kommentteja, mikäli näkee sen tarpeelliseksi.

**Rekisteröitymismahdollisuuden tarjoaminen tapahtumakalenteriin.**

Tällä hetkellä sovelluksessa tarjotaan kirjautumismahdollisuus käyttäen muiden palveluiden jo olemassa olevia tunnuksia, yksi jatkokehitysmahdollisuus olisi antaa käyttäjän rekisteröityä tapahtumakalenteriin. On mahdollista, että käyttäjällä ei ole tunnuksia mihinkään tarjotuista palveluista tai että käyttäjä ei halua käyttää tai yhdistää tietojaan toisesta tilistään.

**Käyttäjien saaminen sivustolle.**

Tapahtumakalenteri voisi tehdä yhteistyötä erilaisten tapahtumajärjestäjien kanssa. Heille voitaisiin antaa edullisesti mainostilaa ja enemmän näkyvyyttä heidän tapahtumillensa. Sovelluksessa pitäisi olla jokin syy, mikä saisi ihmiset tulemaan sivustolle uudestaan ja uudestaan. Sivustolla voitaisiin esimerkiksi järjestää erilaisia kilpailuja, joissa käyttäjät voisivat voittaa ilmaislippuja tapahtumiin itsellensä ja kaverillensa. Palkintoja voisi myös ajatella antavansa, kun yksi käyttäjä on lisännyt tietyn määrän tapahtumia tai esimerkiksi, kun joku käyttäjistä lisää sovellukseen tuhannennen tapahtuman. Tapahtumakalenterin ei tarvitse vain sisältää tapahtumia, vaan siellä voisi olla myös muutakin ajanvietettä, kuitenkin niin että tapahtumakalenteri on kantava idea. Sovelluksen runko on suunniteltu niin, että sitä on helppo myöhemmin laajentaa esimerkiksi edellä mainituilla ominaisuuksilla. Tapahtumakalenteriin voitaisiin myöhemmin toteuttaa asiakkuuksille mahdollisuus luoda profiilisivu.

**Facebook yhteistyön kattavuus ja Facebookista saatavan tiedon käyttö.**

Tapahtumakalenteria suunniteltaessa ajateltiin hyödyntää Facebookin tapahtuma-ominaisuutta. Ideana olisi ollut, että tapahtumakalenterin tapahtumasta olisi voitu luoda myös Facebook-tapahtuma. Ongelmaksi olisi kuitenkin tullut tapahtuman ylläpitäminen, sillä tapahtuman muutokset olisi pitänyt tehdä sekä Facebookiin että tapahtumakalenteriin. Tapahtumista voidaan kuitenkin “tykätä”, jolloin käyttäjän Facebook-seinälle tulee linkki tapahtumaan. Sen ajateltiin olevan riittävän hyvä ominaisuus tiedon jakamiseksi tapahtumista.

**Käyttöliittymän muokkaaminen oman näköiseksi.**

Sovelluksen käyttäjillä on erilaisia tarpeita. Jotkut käyttäjät haluavat nähdä ja seurata tapahtumia tietyillä kriteereillä. Käyttäjillä tulisi siis olla mahdollisuus muokata käyttöliittymää oman näköiseksi siten, että käyttäjä voi valita itseään kiinnostavia tapahtumia ja kategorioita. Käyttäjän täytyisi olla kirjautuneena tapahtumakalenteriin, jotta kriteerien asettaminen olisi mahdollista. Kriteerit tallennettaisiin käyttäjän profiiliin, jotta ne olisivat käytettävissä myös seuraavilla kirjautumiskerroilla. Käyttäjä voi mää-

ritellä käyttöliittymässä näkyvät tapahtumat esimerkiksi oman kaupungin tapahtumien mukaan.

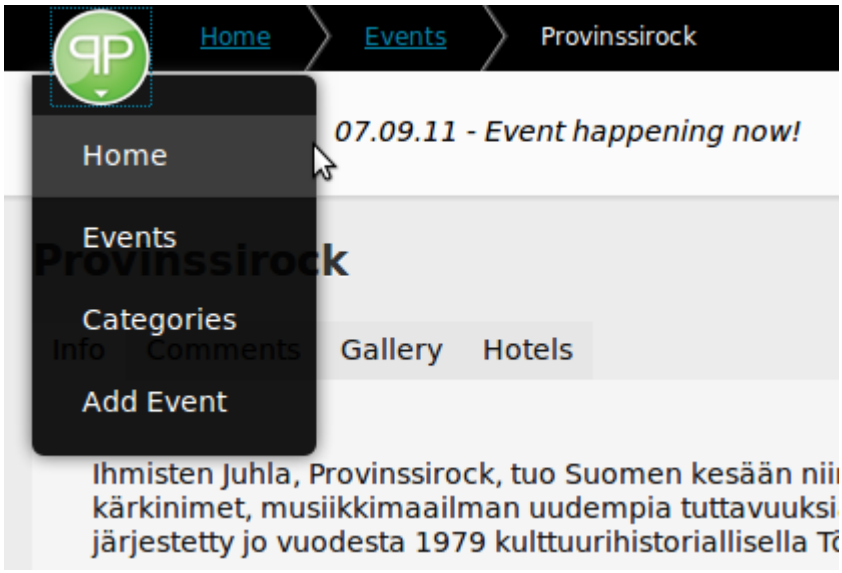
**Käyttöliittymän opittavuus.**

Yhden napin käyttöliittymä voi yksinkertaisuudessaan olla käyttäjälle aluksi vaikea oppia, sillä monet käyttäjät ovat tottuneet näkemään sivuston ylälaidassa vaakasuoraan asetellun valikon eli listan linkkejä joiden kautta sivustolla navigoidaan. Yhden napin kautta suoritettavassa navigoinnissa on sekä hyvät että huonot puolensa. Hyvänä puolena se, että käyttäjä tekee lähes kaiken navigointinsa saman napin kautta eli palaa toistuvasti samaan nappiin. Käyttöliittymä on siis opittavuudeltaan varsin helppo. Mutta taas toisaalta kun valikossa olevat sivun nimet ovat piilossa napin takana avautuvassa valikossa, ei sivuston nimiä näytetä sivuston ylälaidassa. Eikä niitä siis voida myöskään käyttää selventämään käyttäjälle, millä sivustolla milloinkin ollaan ja mitä muita sivuja sivustolla on tarjolla.

**Napsautusten määrä sovelluksen jatkuvassa käytössä.**

Tapahtumakalenterin valikkorakenne on piilotettu logoon, jotta käyttöliittymässä ei olisi mitään ylimääräistä näkyvissä. Logon ulkoasusta on yritetty tehdä huomiota herättävä, jotta käyttäjä ymmärtäisi painaa sitä. Yhden napin käyttöliittymä pakottaa käyttäjän siis napsauttamaan logoa aina halutessaan navigoida sivulta toiselle, eli jokaiseen sivun vaihtoon menee vähintään kaksi napsautusta. Pidempiaikaisessa käytössä tämä voi tuntua rasittavalta ja tarpeettomalta.

Valikkorakenteesta ei kuitenkaan haluttu luopua, koska se oli tärkeä osa käyttöliittymäsuunnittelua. Päätimme ratkaista ongelman toteuttamalla sovellukseen murupolun päävalikon rinnalle, joka näyttää polun, miten käyttäjä on päätynyt sillä hetkellä olevalle sivulle. Kuviossa 30 on esitetty sovellukseen toteutettu murupolku.




**Kuvio 30.** Sovelluksen navigointia vahvistavan murupolun toteutus.

# 7 PÄÄTELMÄT

Opinnäytetyön aihe oli laaja, joten siitä löytyi paljon aiheita, joihin perehtyä. Toisaalta projekti oli enemmän eksperimentaalinen toteutus kuin jonkin olemassa olevan pohjalta toteutettu. Projektissa käytettiin koko ohjelmistokehityksen eri osa-alueiden skaalaa, joihin kuului muun muassa vaatimusten keräämistä, käyttötapausten määrittelyä, käyttöliittymän- ja tietokannan suunnittelua ja toteuttamista.

Opinnäytetyöraportti on selvitys toteuttamastamme reaaliaikaisesta tapahtumakalenterisovelluksesta. Sovelluksen toteuttamiseen käytettiin kaksi kolmasosaa koko opinnäytetyön toteutukseen käytetystä ajasta. Raportti sisältää enemmän käytännön asiaa kuin teoriaa, mutta teoriaa käytettiin runsaasti raportin tukena. Käytännön työn teko oli paljon raportin tekoa mielekkäämpää. Toisaalta raportin kirjoittaminen oli helppoa ja vaivatonta, koska sovellus oli toteutettu ennen raporttia. Raportissa käydään läpi sovelluksen toteutus ja siihen sovellettava teoria.

Opinnäytetyön laajuuden ansiosta jokaiselle riitti tasapuolisesti tekemistä. Perusteellinen suunnittelu ja työnjako olivat tekijöitä, joilla työ saatiin käyntiin ja ketterän kehityksen menetelmien avulla projektin kokonaisuutta pystyttiin hallitsemaan. Scrumin avulla keskityttiin toteuttamaan sillä hetkellä sovellukselle kriittisimpiä tehtäviä. XP:n käytännöt osoittautuivat hyödyllisiksi nopeuttamalla ongelman ratkaisua ja helpottamaan ohjelmakoodin kirjoittamista. Lyhyillä julkaisuilla vältettiin julkaisujen välisiä ristiriitoja. Työtuntiseurantajärjestelmällä pystyttiin seuraamaan työtunteja niin, että kaikilla osa-alueilla pysyttiin aikataulussa.

Tapahtumakalenteriin toteutettiin kaikki keskeiset vaatimusmäärittelyyn kerätyt toiminnallisuudet. Testaus jäi hieman puutteelliseksi käytettävyyden osalta, joten esimerkiksi yhden napin käyttöliittymä ei välttämättä ole paras mahdollinen ratkaisu. Tiedonsiirto eri sosiaalisten palveluiden välillä jäi suunniteltua vähäisemmäksi, koska tapahtumien synkronointi molemmin puolin koitui haasteeksi.

Suunnitteluvaiheessa olisi voitu valita jokin muu sovelluskehys Ruby on Railsin sijasta. Jos sovelluskehys olisi ollut jokin muu, olisi se vaatinut enemmän opettelua, eikä sillä kehittäminen välttämättä olisi ollut sen nopeampaa kuin Railsilla. Perinteisemmän ohjelmointikielen käyttö olisi tehnyt syntaksista luettavampaa ja toiminnallisuuden rakentamisesta helpommin lähestyttävää.

Ulkoasulla poikettiin perinteisestä otsakerakenteesta, niin että poistettiin tyypillinen vaakanavigaatio. Tiedostimme tämän olevan riski, sillä toteuttamamme yhden napin navigaatio poikkesi selkeästi standardin mukaisesta käyttöliittymärakenteesta. Toisaalta sovellusmaisen ulkoasun suosio voisi tulevaisuudessa kasvaa mobiililaitteiden yleistyessä.

Testivetoista kehitystä ei käytetty tarpeeksi, järjestelmätestaus aloitettiin kunnolla vasta sovelluksen valmistuttua. Selkeää testaussuunnitelmaa ei ollut, vaan testasimme toiminnallisuuden aina heti sen valmistuttua käyttöliittymän kautta manuaalisesti. Sovelluksessa ei ollut regressiotestejä tai muuta automatisointia, joka olisi varmistanut järjestelmän toimivuuden, niin ettei uusi ominaisuus riko jo testattua toiminnallisuutta.

Projektinhallintamenetelmiä käyttämällä sovellus etenee systemaattisesti kun eri vaiheita suunnitellaan ja toteutetaan oikeassa järjestyksessä. Nykyaikaiset työkalut helpottavat projektinhallintaa visualisoimalla esimerkiksi käynnissä olevat, tehdyt ja tehtävät työt Scrum-taulun muodossa.

Google Docsin mahdollistama kollaboratiivinen työskentely katsottiin eduksi raportin kirjoittamisessa ja kaavioiden muodostamisessa, sillä kirjoittaminen samanaikaisesti muiden kanssa lisäsi motivaatiota ja helpotti tekstin muodostamista. Omia ja muiden tekemiä virheitä oli helppo korjata ja teksti eli koko ajan muutosten mukaan. Kollaboratiivinen työkalu toimi pilvessä ja näin vältyttiin massamuistilaitteiden aiheuttamilta synkronoinnin monimutkaisuuksilta. Raportista oli käytössä aina ajantasainen

versio “pilvessä” ja pystyimme kaikki muokkaamaan tekstiä yhtäaikaisesti reaaliajassa.

Käyttämämme kollaboratiivinen työkalu ei tukenut viitehallintaa, joten käytimme työkalun kommentointitoimintoa lähdeviittausten merkitsemiseen. Kommentit ja lähdeviittaukset siirtyivät myös vaivattomasti muihin tekstinhallintatyökaluihin. Käytimme myös palvelussa olevaa keskustelutoimintoa raportin kirjoittamisen tukena. Keskustelussa kävimme muun muassa läpi tekstissä olevia puutteita ja siihen vielä lisättäviä osia tai tehtäviä muutoksia.

Skaalautuvien tekniikoiden käyttö mahdollistaa sovelluksen jatkuvuuden teknisessä mielessä eli uusien ominaisuuksien lisääminen on vaivatonta ja optimointi on mahdollista. Uudet tekniikat, esimerkiksi reaaliaikaisuus, mahdollistavat uusien ideoiden kehittelyä ja vanhojen ideoiden ratkaisemisen uudelleen modernilla tavalla. Sovelluksen suunnittelussa ei kuitenkaan voida tukeutua pelkästään uusiin menetelmiin, vaan olemassa olevat standardit ja klassiset käytännöt on otettava huomioon.

Menetelmiäkin tärkeämpänä sovelluksen menestyksen kannalta näkisimme sisällönmäärän ja käyttäjäkunnan laajuuden, sillä loppukäyttäjää ei välttämättä edes kiinnosta millä menetelmillä sivusto on toteutettu. Vaikka käyttöliittymä olisi graafisesti tyylikäs ja käytetyt menetelmät olisivat kuinka ketteriä tahansa, sisällön laatu ja määrä saa käyttäjän palaamaan sivustolle uudestaan ja uudestaan.

# 8 LÄHTEET

A Guide to Active Record Associations (19.10.2011) [online]. [Viitattu 19.10.2011]. <URL:http://guides.rubyonrails.org/association_basics.html#polymorphic-associations>

Adobe (14.7.2011) Flash Platform run times statistics [online]. [Viitattu 25.10.2011]. <URL:http://www.adobe.com/products/flashplatformruntimes/statistics.html>

Adobe Kuler (2011) [online]. [Viitattu 9.11.2011]. <URL:http://kuler.adobe.com/#themeID/1573432>

Ambler Scott W. (2011) [online]. [Viitattu 5.12.2011]. <URL:http://www.agiledata.org/essays/tdd.html>

Asset Pipelines (2011) [online]. [Viitattu 10.10.2011]. <URL: http://guides.rubyonrails.org/asset_pipeline.html>

Asynchronous applications (2003) [online]. [Viitattu 9.11.2011]. <URL:http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.sdk_12.5.1.ctref/html/ctref/ctref21.htm>

Beck Kent (2004). Extreme Programming Explained: Embrace Change.

Burnham Trevor (2011). CoffeeScript.

Chapman Cameron (22.9.2009). Guide to CSS Font Stacks: Techniques and Resources [online]. [Viitattu 3.11.2011]. <URL:http://coding.smashingmagazine.com/2009/09/22/complete-guide-to-css-font-stacks/>

Cockburn Alistair (2001). Writing Effective Use Cases.

CSS3 Tutorial (2011). W3Schools [online]. [Viitattu 3.11.2011]. <URL: http://www.w3schools.com/css3/default.asp>

Dahl Ryan (10.10.2011). node.js [online].[Viitattu 20.10.2011]. <URL: http://nodejs.org/#about>

Don’t Repeat Yourself (11.10.2011) [online]. [Viitattu 10.11 2011]. <URL:http://c2.com/cgi/wiki?DontRepeatYourself>

Douglas Burce Powel (2003). Real-Time Design Patterns.

Dropbox Features (2011) [online]. [Viitattu 24.10.2011]. <URL:http://www.dropbox.com/features>

Facebook Statistics (2011) [online]. [Viitattu 20.10.2011]. <URL:https://www.facebook.com/press/info.php?statistics>

Finley Klint (25.1.2011). Wait, What's Node.js Good for Again? [online]. [Viitattu 20.10.2011]. <URL: http://www.readwriteweb.com/hack/2011/01/wait-whats-nodejs-good-for-aga.php>

Freeman Eric, Freeman Elisabeth, Sierra Kathy & Bates Bert (2004). Head First Design Patterns.

Frontend (2011) The Developer Community [online]. [Viitattu 9.11.2011]. <URL: http://frontend.fi/>

Geolocation API Specification (10.2.2010) [online]. [Viitattu 4.11.2011]. <URL:http://dev.w3.org/geo/api/spec-source.html>

Getting Brian (22.10.2007). What Are “Tags” And What Is “Tagging?” [online]. [Viitattu 10.11.2011]. <URL:http://www.practicalecommerce.com/articles/589-What-Are-Tags-And-What-Is-Tagging->

Getting Started with Rails (10.11.2011) [online]. [Viitattu 10.11.2011]. <URL: http://guides.rubyonrails.org/getting_started.html>

Git Software (2011) [online]. [Viitattu 22.11.2011]. <URL: http://en.wikipedia.org/wiki/Git_%28software%29>
GitHub (2011) [online]. [Viitattu 3.11.2011]. <URL: http://www.github.com>

Google Docs (2011) [online]. [Viitattu 24.10.2011]. <URL: https://docs.google.com>

Gravelle Rob (2011). Comet Programming: Using Ajax to Simulate Server Push [online]. [Viitattu 3.11.2011]. <URL:http://www.webreference.com/programming/javascript/rg28/>

Guillermo Rauch (23.6.2011). LearnBoost Socket.IO [online]. [Viitattu 25.10.2011]. <URL: https://github.com/LearnBoost/socket.io-spec>

Hansson David Heinemeier (22.5.2011). Riding Rails: Rails 3.1: Release candidate [online]. [Viitattu18.10.2011]. <URL:http://weblog.rubyonrails.org/2011/5/22/rails-3-1-release-candidate>

Hogan Brian P. (2010). HTML5 and CSS3: Develop with Tomorrow's Standards Today (Pragmatic Programmers).

HTML <!DOCTYPE> Declaration (2011) W3Schools [online]. [Viitattu 3.11.2011]. <URL: http://www.w3schools.com/tags/tag_doctype.asp>

Hunt Andy, Thomas Dave (1999). Pragmatic Programmers [e-kirja].

IBM (2011). Cloud Computing [online]. [Viitattu 7.11 2011]. <URL:http://www-05.ibm.com/fi/solutions/cloud/>

IP2Location (4.11.2011) Wireless Geolocation (HTML5 Geolocation API) vs. IP Geolocation [online]. [Viitattu 4.11.2011]. <URL:http://www.ip2location.com/html5geolocationapi.aspx>

JSON (2011). Introducing JSON [online]. [Viitattu 9.11.2011]. <URL:http://www.json.org/>

Juselius Ulrika (2004). Typografia [online]. [Viitattu 10.11.2011]. <URL:http://www.phpoint.fi/ulrikaj/www/typo.htm>

Kalid Azad (14.8.2007). Understanding Models, Views and Controllers [online]. [Viitattu 14.11.2011]. <URL:http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/>

Käyttötuotteen heuristinen arviointi [online]. [Viitattu 3.11.2011]. <URL:http://mlab.uiah.fi/polut/Design/tyokalu_heuristinen_arvio.html>

KISS Principle (Keep It Simple, Stupid) (1.2.2001) [online]. [Viitattu 9. 11 2011]. <URL:http://searchcio-midmarket.techtarget.com/definition/KISS-Principle>

Koskimies Kai, Mikkonen Tommi (2005). Ohjelmistoarkkitehtuurit.

Kosonen (2011). Testauksen suunnittelu ja valmistelu [online]. [Viitattu 3.11.2011]. <URL:http://cna.mikkeliamk.fi/Public/KosonenH/Projektinhallinta/IT_projektinhallinta/it-projektinhallinta_luku09.pdf>

Krug Steve (2006). Don’t Make Me Think 2nd edition.

McConnell Steve (2004). Code Complete 2.

McGowan Aaron (1.3.2010). Object Serialization in PHP [online]. [Viitattu 9.11.2011]. <URL:http://www.amcgowan.ca/blog/general/object-serialization-in-php/>

Microsoft (9.5.2011). Message Queuing (MSMQ) [online]. [Viitattu 10.11.2011]. <URL: http://msdn.microsoft.com/en-us/library/ms711472.aspx>

MongoDB (2011) [online]. [Viitattu 9.11.2011]. <URL: http://www.mongodb.org/>

MVC Architecture (28.3.2008) [online]. [Viitattu 6.10.2011]. <URL:http://www.roseindia.net/struts/mvc-architecture.shtml>

MySQL (2011). Why MySQL? [online]. [Viitattu 31.10.2011]. <URL: http://www.mysql.com/why-mysql>

Nielsen Jakob (17.4.2006). F-Shaped Pattern for Reading Web Content [online]. [Viitattu 20.10.2011]. <URL:http://www.useit.com/alertbox/reading_pattern.html>

Nielsen Jakob (17.4.2006). F-Shaped Pattern for Reading Web Content [online]. [Viitattu 20.10.2011]. <URL:http://www.useit.com/alertbox/reading_pattern.html>

Nielsen Jakob (1999). Designing Web Usability.

Nielsen Jakob (2005). Ten Usability Heuristics [online]. [Viitattu 20.10.2011]. <URL: http://www.useit.com/papers/heuristic/heuristic_list.html>

North Dan (2011). Introducing BDD [online]. [Viitattu 14.11.2011]. <URL:http://dannorth.net/introducing-bdd/>

O'Dell Jolie (10.3.2011) Why Everyone Is Talking About Node [online]. [Viitattu 20.10.2011]. <URL:http://mashable.com/2011/03/10/node-js/>

Obie Fernandez (19.3.2008). Big Name Companies Using Ruby on Rails [online]. [Viitattu 10.11.2011]. <URL:http://blog.obiefernandez.com/content/2008/03/big-name-compan.html>

Oxagile (2011). Comet Programming in Web Development [online]. [Viitattu 14.11.2011]. <URL:http://www.oxagile.com/article/253-comet-programming-web-development>

Performance Tuning SQL Server Joins (1.11.2005) [online]. [Viitattu 19.11.2011]. <URL:http://www.sql-server-performance.com/2006/tuning-joins/>

PHP (2011) PHP: Hypertext Preprocessor [online]. [Viitattu 31.10.2011]. <URL:http://www.php.net/>

Poimala Sami, Heikniemi Jouni & Blåfield Henrik (2011). Ketterät käytännöt - Scrum [online]. [Viitattu 17.10.2011]. <URL:http://www.ketteratkaytannot.fi/fi-FI/Menetelmat/Scrum/>

Puhakka Aapo (13.5.2004). Väliohjelmistojen (middleware) kehitysnäkymiä [online]. [Viitattu 9.11.2011]. <URL:http://aapo.iki.fi/valiohjelmistot.html>

Rails on Rack. (18.10.2011) [online]. [Viitattu 18.10.2011]. <URL:Ruby on Rails Guides: http://guides.rubyonrails.org/rails_on_rack.html#internal-middleware-stack>

Ruby on Rails (2011) [online]. [Viitattu 5.12.2011]. <URL: http://rubyonrails.org/>

Ruby on Rails Documentation (7.10.2011) [online]. [Viitattu 10.11.2011]. <URL: http://api.rubyonrails.org/>

Ruby Sam, Thomas Dave & Hansson David Heinemeier (2011). Agile Web Development with Rails (4th edition).

Savolainen Erkki (2001) Lauseoppi eli Syntaksi [online]. [Viitattu 10.11.2011]. <URL:http://www.finnlectura.fi/verkko/esittely/sivu3.htm>

Schwaber Ken (2004). Agile Project Management with Scrum [e-kirja].

Scrum Alliance (2011) [online]. [Viitattu 24.10.2011]. <URL: http://www.scrumalliance.org/>

Scrumblr (2011). [online]. [Viitattu 24.10.2011]. <URL: http://scrumblr.ca/nsgthesishappen>

Skype (2011) About [online]. [Viitattu 10.11.2011]. <URL:http://about.skype.com/>

Smashing Magazine - Breadcrumbs In Web Design: Examples And Best Practices (17.3.2009) [online]. [Viitattu 3.11.2011]. <URL:http://www.smashingmagazine.com/2009/03/17/breadcrumbs-in-web-design-examples-and-best-practices-2/>

Social Statistics (2011) [online]. [Viitattu 20.10.2011]. <URL: http://socialstatistics.com/>

Socket.IO Faq (2011) [online]. [Viitattu 25.10.2011]. <URL: http://socket.io/#faq>

Sriram Srinivasan (1.8.1997). Advanced Perl Programming [online]. [Viitattu 3.10.2011]. <URL:http://docstore.mik.ua/orelly/perl/advprog/ch04_02.htm>

Style with attitude (2011) [online]. [Viitattu 10.10.2011]. <URL:http://sass-lang.com/>

Swicegood Travis (2008). Pragmatic Version Control Using Git [e-kirja].

Symfony (2011) What is Symfony [online]. [Viitattu 10.11.2011]. <URL: http://symfony.com/what-is-symfony>

Thomas Dave, Fowler Chad & Hunt Andy (2009). Programming Ruby 1.9 (3rd edition).

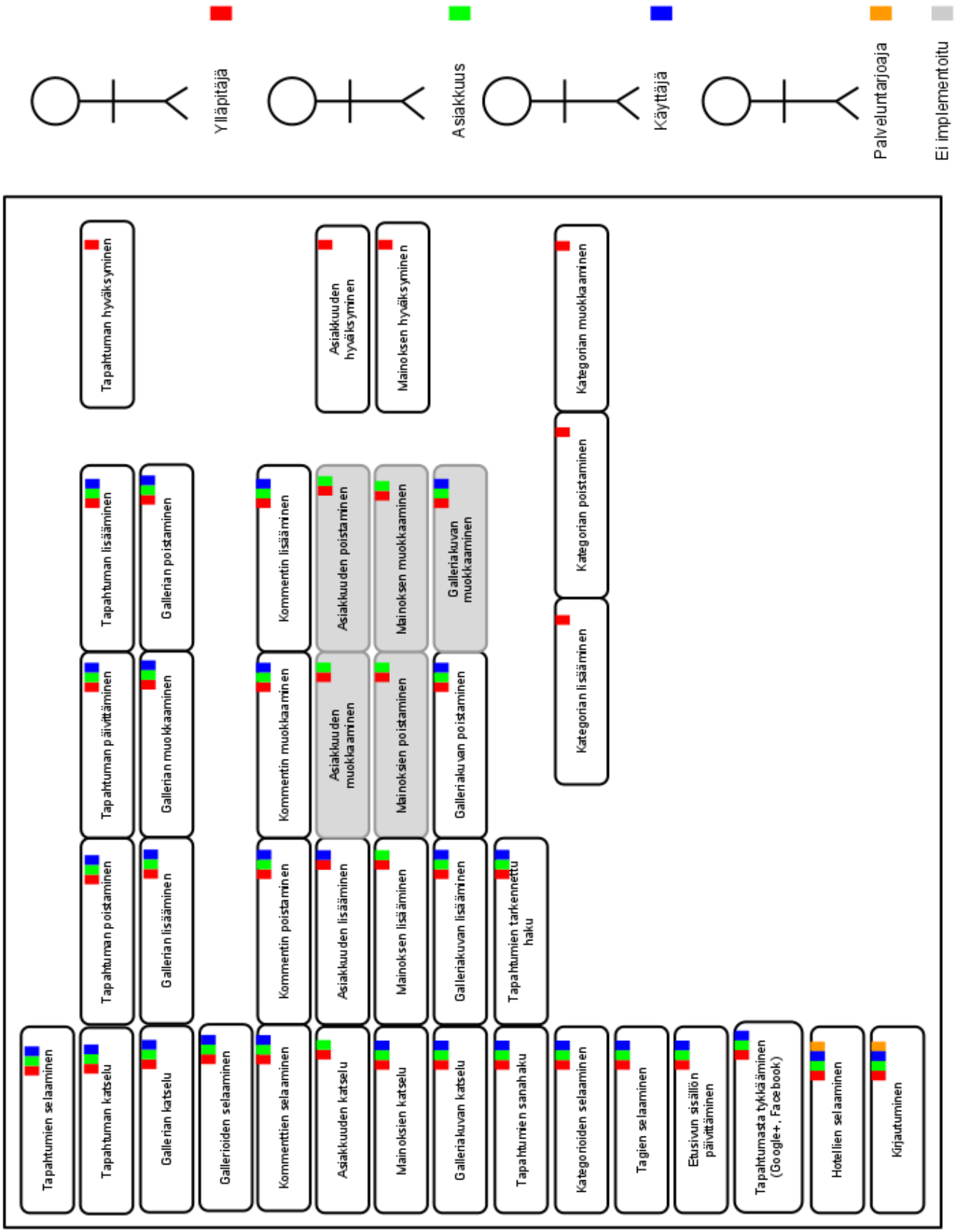
to_json (2008) [online]. [Viitattu 12.10.2011]. <URL:http://apidock.com/rails/ActiveRecord/Serialization/to_json>

Toggl (2011) Time Tracking That Works [online]. [Viitattu 24.10.2011]. <URL: https://www.toggl.com/>

Typeface Foundries (2011) [online]. [Viitattu 10.11.2011]. <URL: http://www.designingwithtype.com/typefaces_foundries.html>

W3C (25.5.2011) Working Draft [online]. [Viitattu 10.11.2011]. <URL: http://www.w3.org/TR/html5/>

Ward Matt, Charchar Alexander, Inschauste Fransisco, Rundle Mike, Jovanovic Janko, Heilmann Christian, Anaylan Vivien, Kolb Christoph, Weinschenk Susan, Bradley Steven (2011). The Smashing Book #2.


Tapahtumakalenteri
Tapahtumien selaaminen
Tapahtuman katselu
Gallerian katselu
Gallerioiden selaaminen
Kommenttien selaaminen
Asiakkuuden katselu
Mainoksien katselu
Galleriakuvan katselu
Tapahtumien sanahaku
Kategorioiden selaaminen
Tagien selaaminen
Etusivun sisällön päivittäminen
Tapahtumasta tykkääminen (Google+, Facebook)
Hotellien selaaminen
Kirjautuminen
Tapahtuman poistaminen
Gallerian lisääminen
Kommentin poistaminen
Asiakkuuden lisääminen
Mainoksen lisääminen
Galleriakuvan lisääminen
Tapahtumien tarkennettu haku
Tapahtuman päivittäminen
Gallerian muokkaaminen
Kommentin muokkaaminen
Asiakkuuden muokkaaminen
Mainoksien poistaminen
Galleriakuvan poistaminen
Tapahtuman lisääminen
Gallerian poistaminen
Kommentin lisääminen
Asiakkuuden poistaminen
Mainoksen muokkaaminen
Galleriakuvan muokkaaminen
Kategorian lisääminen
Kategorian poistaminen
Kategorian muokkaaminen
Tapahtuman hyväksyminen
Asiakkuuden hyväksyminen
Mainoksen hyväksyminen
Ylläpitäjä
Asiakkuus
Käyttäjä
Palveluntarjoaja
Ei implementoitu