SOVELLUS VARHAISKASVATUKSEN ASIAKASYHTEYDENPITOON

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäki, Tieto- ja viestintätekniikka

Kevät 2018

Jussi Pennanen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Riihimäki

Tekijä Jussi Pennanen **Vuosi** 2018

Työn nimi Sovellus varhaiskasvatuksen asiakasyhteydenpitoon

Työn ohjaaja Jari Mustajärvi

TIIVISTELMÄ

Opinnäytetyön tavoite oli luoda web-sovellus varhaiskasvatusta tarjoavien organisaatioiden asiakasyhteydenpitoa varten. Sovellus luotiin erillisinä palvelin- ja asiakassovelluksina. Palvelinsovellus tarjoaa API:n tietokantaan ja huolehtii käyttäjien todennuksesta ja oikeuksista asiakassovelluksen keskittyessä tarjoamaan selainkäyttöisen käyttöliittymän.

Palvelinsovelluksen toteutuksessa käytettiin Python-kieltä, Flask-sovelluskehystä sekä MongoDB-dokumenttitietokantaa. Asiakassovellus toteutettiin JavaScript-kielellä käyttäen AngularJS-sovelluskehystä.

Kehitysprojekti onnistui aikataulussaan ja kaikki suunnitellut ominaisuudet saatiin toteutettua ilman suuria ongelmia. Ohjelmisto julkaistiin avoimena lähdekoodina MIT-lisenssillä. Ohjelmasta ei tarkoituksellisesti tehty valmista kaupallista käyttöä varten, mutta mahdollisuus omaan kaupalliseen hyödyntämiseen ja jatkokehitykseen jätettiin auki.

Avainsanat olio-ohjelmointi, tietokantaohjelmat, varhaiskasvatus, WWW

Sivut 23 sivua, joista liitteitä 2 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Riihimäki

Author Jussi Pennanen **Year** 2018

Subject Application of customer communication for early childhood education

Supervisor Jari Mustajärvi

ABSTRACT

The purpose of this thesis was to create a web application for customer communication to be used in organizations providing early childhood education. The application was created as separate server and client applications. The server application provides an API for the database and handles user authentication and authorization while the client application focuses on providing a browser-based user interface.

The server application was created using the Python language, the Flask web application framework and the MongoDB document database. The client application was written in JavaScript using the AngularJS framework.

The development project was completed according to schedule and all the planned functionalities were completed without any major issues. The application was released as open source using the MIT license. Some functionality that would be essential for commercial use was intentionally left out. The possibility for utilizing the application commercially and for further development was left open.

Keywords database, early childhood education, object-oriented programming, WWW

Pages 23 pages including appendices 2 pages

SISÄLLYS

Liitteet

1 JOHDANTO

Opinnäytetyön aiheena on kehittää varhaiskasvatusta tarjoavan organisaation käyttöön tarkoitettu sovellus, joka mahdollistaa tehokkaan yhteydenpidon lasten vanhempiin. Sovellus on toteutettu ilman toimeksiantajaa, kuunnellen kuitenkin muutaman varhaiskasvatuksen ammattilaisen toiveita sovelluksen toiminnoista. Koska opinnäytetyöllä ei ole toimeksiantajaa, julkaistaan ohjelmisto avoimena lähdekoodina vapaasti kaikkien saataville ja käytettäväksi.

Toivotut ydintoiminnot sovelluksessa ovat kaikille yhteisiä tiedotteita sisältävä ilmoitustaulu, ryhmä/lapsikohtaiset viestit, ryhmäkohtainen aikataulu sekä tiedostopankki. Käyttäjät pitää pystyä erottelemaa henkilökuntaan ja huoltajiin. Ohjelmiston tulee sisältää ominaisuus sisältöjen näkyvyyden rajaamiseksi käyttäjä- tai ryhmäkohtaisesti.

2 TAUSTAA

2.1 Olemassa olevat tuotteet

Kehitettävää ohjelmistoa vastaavia tuotteita on markkinoilla jo valmiiksi. Kaupallisia tuotteita on ainakin Visma InSchool -kokonaisuuteen kuuluva Wilma ja Adenovan luoma Tenavanetti/Digitaalinen kasvunkansio. Avoimen lähdekoodin projekteista vastaavalla alalla on mainittava alun perin Wilman kilpailijaksi tarkoitettu ja vielä keskeneräinen Qlma.

Wilma on Visma InCommunity Oy:n tuote, joka on osa laajempaa Visma InSchool -ohjelmistokokonaisuutta. Wilma on suunnattu ensisijaisesti koulumaailmaan perusopetuksen oppilaiden huoltajien ja opettajien välisen yhteydenpidon välineeksi, mutta Wilmaa on mahdollista käyttää varhaiskasvatuksessa ja myös perusopetuksen jälkeisessä koulutuksessa. Kyse onkin suuren joukon ominaisuuksia sisältävästä kokonaisuudesta, joka integroituu muihin saman sovelluskokonaisuuden ohjelmiin yhteisen tietokannan kautta. (Visma n.d.)

Adenova on tamperelainen ohjelmistoyritys, jonka yksi referenssituote tunnetaan sekä Tenavanetti- että Digitaalinen kasvunkansio -nimillä. Ohjelmisto sisältää samoja ominaisuuksia kuin tässä työssä kehitetty ratkaisu. Tenavanetti on suunnattu ensisijaisesti päivähoidon käyttöön, ja Adenovan mukaan sen käyttäjinä on sekä kaupunkeja että yksityisiä päiväkoteja. (Adenova 2010.)

Qlma on avoimen lähdekoodin sovellus, jonka lähtökohta oli tarjota Wilmalle vaihtoehto. Projekti alkoi vauhdikkaasti ja tukemaan liittyi yrityksiä ja viranomaistahojakin. (Vänskä 2015.) Alkuinnostuksen jälkeen projekti vaikuttaa kuitenkin jääneen keskeneräiseksi ja kehitys vaikuttaa päättyneen, viimeisiä päivityksiä varsinaiseen ohjelmakoodiin projektin GitHub-repositoryissä on tehty vuonna 2016.

2.2 **Sovelluksen kohderyhmät**

Sovelluksen mahdollinen käyttäjäkunta koostuu kahdesta eri ryhmästä: varhaiskasvatusorganisaatioiden työntekijöistä ja asiakkaista eli lasten vanhemmista. Varhaiskasvatusta tarjoavista organisaatioista kuntien päiväkoti lienee kaikille tuttu, mutta myös seurakunnat ja monet yksityiset toimijat tarjoavat varhaiskasvatusta päiväkerhojen muodossa. Monesti pienemmillä organisaatioilla ei ole rahallisia resursseja ottaa käyttöön vastaavia kaupallisia tuotteita. Pääsääntöisesti sovellus suunnataan toiminnoiltaan kevyttä ja yksinkertaista ratkaisua toivovalle organisaatiolle.

Henkilökunnalle sovellus tarjoaa helpon keinon viestiä samaan aikaan kokonaisen lapsiryhmän kaikkien huoltajien kanssa, yksittäisen lapsen huoltajien kanssa tai jopa kaikkien asiakkaiden kanssa. Vanhemmille sovellus luo yhden paikan, jossa seurata lapsen varhaiskasvatukseen liittyvää tiedotusta, aikatauluja ja viestejä.

2.3 **Ohjelmistokehityksen prosessi**

Sovelluksen varsinainen ohjelmointityö pyritään toteuttamaan viiden viikon ajanjakson aikana. Tätä ennen on jo suunniteltu alustavasti tietorakenteita sekä ohjelmiston yleistä rakennetta. Tässä voidaan nähdä yhtymäkohta perinteiseen vesiputousmalliin, jossa suunnitteluvaihe on selkeästi erillään toteutusvaiheesta. Toteutusvaiheessa edetään enemmänkin Scrum-mallin tyylisesti. Scrumin ydin on pyrähdys, jonka alussa päätetään mitä tehdään, jonka aikana toteutetaan valitut tehtävät ja jälkeen pidetään katselmointikokous (Mäkinen 2017). Poiketen tästä perusmallista kaikkien pyrähdysten sisältö on päätetty jo projektin alussa, eikä pyrähdyksen alussa pidetä kokousta. Yhden henkilön toteuttamassa projektissa tämä olisikin turhaa. Jokaisen pyrähdyksen lopussa sen sijaan on tarkoitus pitää katselmointikokousta vastaava tarkastus opinnäytetyön ohjaajan kanssa.

Yhtymäkohta prosessissa on helppo nähdä myös DevOps-malliin, jossa ohjelmistokehittäjien sekä tuotantohenkilöiden välinen kommunikaatio sekä ohjelmistokehityksessä käytettävä automaatio ovat tärkeässä roolissa (Mäkinen 2017). Yhden henkilön kokonaisuudessaan toteuttamassa projektissa kaikki tehtävät keskittyvät samalle henkilölle. Tarkoitus ei kuitenkaan ole pystyttää varsinaista tuotantoon soveltuvaa palvelua, mutta testipalvelin kylläkin. Myöskään ei ole tarkoitus ottaa käyttöön automatiikkaa ohjelmiston asennukseen tai testaukseen.

3 TEKNOLOGIAT

3.1 Python

Python on tulkattu korkean tason ohjelmointikieli, joka soveltuu olio-ohjelmointiin. Kielen kehitys on aloitettu 1980-luvun lopulla ja ensimmäinen versio julkaistiin vuonna 1991. Pythonia voi käyttää useilla eri alustoilla (yleisimmät UNIX-variantit, Mac sekä Windows) yleiskäyttöisenä ohjelmointikielenä. (Python Software Foundation 2018.)

Python pyrkii olemaan helposti luettava kieli. Syntaksissa käytetään monissa muissa kielissä käytettyjen kaarisulkeiden tai muiden välimerkkien sijasta tyhjää tilaa (whitespace). Python on hyvä ohjelmointikieli aloittelevalle ohjelmoijalle, mutta sisältää kuitenkin suuren määrän kirjastoja, jotta sitä voi käyttää monien todellisten ongelmien ratkaisuun. (Wikipedia 2018 a.)

Tämän opinnäytetyön kannalta olennaisimpia kirjastoja ovat web-ohjelmistokehykset sekä liitynnät tietokantoihin.

3.2 Flask

Flask on Python-kielellä kehitetty ohjelmistokehys web-sovellusten toteutusta varten. Flaskin kehitys on aloitettu vuonna 2010 pääkehittäjä Armin Ronacherin toimesta. Kehys sisältää muun muassa Jinja2-templatemoottorin sekä kehitystyötä helpottavan kehityspalvelimen. Flask on lisensoitu BSD-lisenssillä. (Ronacher 2018.)

Toiseen yleiseen web-kehityksessä käytettävään ohjelmistokehykseen Djangoon verrattuna Flask on huomattavasti kevyempi, eikä ota niin paljon kantaa ohjelmiston rakenteeseen ja toteutukseen (Dwyer 2017). Flask on myös hyvin yksinkertainen eikä vaadi suurta määrää boilerplate-koodia. Flaskia käyttäen esimerkiksi hello world -ohjelma vaatii vain 5 riviä koodia, kuten kuvasta 1 nähdään. Sama kuva esittää myös Flaskin näkymien perusrakenteen. Rivillä 3 määritellään Python-kielen decorator-toimintoa hyväksikäyttäen reitti (”/”), johon navigoitaessa kutsutaan riviltä 6 alkaen määriteltyä funktiota hello(). Näkymäfunktio palauttaa halutun datan, joka tässä tapauksessa on merkkijono, mutta voisi olla myös vaikkapa HTML- tai JSON-muotoista dataa.

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Kuva 1. Hello world -ohjelma Flaskilla.

3.3 JSON

JSON eli JavaScript Object Notation on avoin standardi tieto-olioiden välittämiseen ihmisten luettavassa muodossa. Vaikka JSON on lähtöisin JavaScript-kielestä, se on kuitenkin riippumaton käytettävästä ohjelmointikielestä. JSON-muotoinen tieto koostuu avain-arvopareista. JSONin perusdatatyypit ovat numero (number), merkkijono (string), boolean (tosi tai epätosi), lista (array), olio (object) sekä tyhjä arvo ”null”. (Wikipedia 2018 b.)

Kuvassa 2 on yksinkertainen esimerkki JSON-muotoisesta tiedosta. Kentät ”name” ja ”email” edustavat merkkijono-muitoista tietoa, ”yearOfBirth” on numero, ”comments” on lista, joka sisältää olioita.

```json
{
    "name": "Adam Ant",
    "email": "adam@ant.com",
    "yearOfBirth": 1981,
    "comments": [{
        "title": "Disappointing",
        "content": "I expected Adam to be an actual ant."
    },
    {
        "title": "Small",
        "content": "He is quite small."
    }]
}
```

Kuva 2. JSON-esimerkki.

3.4 NoSQL

NoSQL-tietokannat (sanoista ”non SQL”, joskus myös ”not only SQL”) ovat tietokantoja, joiden tietomalli on järjestetty jollain muulla kuin relaatiotietokannoista tutulla tavalla. NoSQL-tietokannat eivät kaikki tallenna tietoa samassa muodossa, vaan termi käsittää monia erilaisia tiedontallennusmalleja. (MongoDB 2018 a.)

NoSQL-tietokantoja on kehitetty vastaamaan relaatiotietokantoja paremmin nykyaikaisen sovelluskehityksen ja sovellusten sekä niiden tuottaman suuren tietomäärän vaatimuksiin. NoSQL-tietokantoja pidetään yleisesti joustavampina ja paremmin skaalautuvina.

3.4.1 Avain-arvotietokannat

Avain-arvotietokannat (key-value store) sisältävät vain nimettyjä avaimia ja avaimeen liitettyjä arvoja. Joissakin toteutuksissa avaimeen voidaan liittää myös arvon tyyppi. Avain-arvotietokanta on yksinkertaisin NoSQL-tietokantojen tyyppi. (MongoDB 2018 a.)

3.4.2 Dokumenttitietokannat

Dokumenttitietokannoissa jokaista avainta kohden on tietorakenne, jota kutsutaan dokumentiksi. Dokumentti sisältää yhden tai useampia avaimia sekä jokaiseen avaimeen liitetyn arvon. Arvo voi olla myös useampia arvoja sisältävä taulukko, binääridataa tai alidokumentti. Samankaltaiset dokumentit tyypillisesti liitetään toisiinsa tunnisteilla (tag) tai kokoelmilla. (MongoDB 2018 a.)

Dokumenttien rakennetta ei tarvitse määritellä etukäteen, eikä olemassa olevien dokumenttien rakenne määrittele tulevien dokumenttien rakennetta tiukasti edes samassa kokoelmassa. Tämä mahdollistaa tallennettavan tiedon rakenteen muuttumisen sovelluksen vaatimusten muuttuessa ilman, että tietokantaa tarvitsee muutostöiden ajaksi poistaa käytöstä. Merkittävää haittaa ei ole myöskään siitä, jos tallennettavan tiedon rakennetta ei etukäteen tiedetä kokonaan tai lainkaan. Joissakin dokumenttitietokantatoteutuksissa voidaan tallennettavalle tiedolle kuitenkin asettaa kelpoisuussääntöjä.

Dokumenttitietokannat soveltuvat hyvin olio-ohjelmoinnin yhteydessä käytettäväksi, mikäli tallennettava dokumentti on JSON-tyyppinen oliokuvaus. Oliot voidaan tällöin tallentaa tietokantaan suoraan dokumentteina ilman, että tietorakennetta tarvitsee merkittävästi muuttaa sovellusohjelman ja tietokannan välillä.

3.4.3 Muut tyypit

Muita NoSQL-tietokantojen tyyppejä ovat tietojen välisten yhteyksien tallennukseen keskittyvä graafitietokanta (graph store) sekä wide-column store, jossa käytetään tauluja, sarakkeita ja rivejä, mutta sarakkeen tietotyyppi ja nimi voivat muuttua riviltä toiselle (Wikipedia 2018 c).

3.5 **MongoDB**

MongoDB on dokumenttitietokanta, joka relaatiotietokannasta poiketen säilöö tietoa dokumentteina (document). Dokumentit kuuluvat kokoelmaan (collection) ja kokoelmat puolestaan tietokantaan (database). Dokumentit tallennetaan JSON-mallia laajentavana BSON-mallisena (Binary JSON) tietona. Dokumentti sisältää yhden tai useampia nimettyjä kenttiä, ja jokainen kenttä sisältää arvon. Arvona voi olla myös arvoja sisältävä lista, binäärimuotoista dataa tai alidokumentti. (MongoDB 2018 b.)

MongoDB:tä kehittää MongoDB Inc. ja se on julkaistu ensimmäisen kerran vuonna 2009. MongoDB on avoimen lähdekoodin sovellus, jonka lähdekoodi on nähtävillä ainakin GitHub-palvelussa. (Wikipedia 2018 d.)

3.6 **MongoEngine**

MongoEngine on Python-kielelle luotu Document-Object mapper (MongoEngine 2018). MongoEngineä käyttämällä Python-kieliset oliot voidaan tallentaa MongoDB-tietokantaan MongoEnginen huolehtiessa tietokantayhteyksistä.

3.7 **AngularJS**

AngularJS on Googlen kehittämä ja pääasiallisesti ylläpitämä JavaScript-ohjelmistokehys, jota voidaan käyttää asiakaspuolen ohjelmointiin web-sovelluksissa. AngularJS käyttää omia elementtejään HTML-koodin seassa, sitoen elementit malliin, joka on esitetty JavaScript-muuttujien muodossa. Muuttujien arvot voidaan määritellä ohjelmakoodissa tai ne voidaan hakea ulkopuolisesta lähteestä esimerkiksi JSON-muodossa. (Wikipedia 2018 e.)

3.8 **Avoin lähdekoodi**

Avoimella lähdekoodilla tarkoitetaan tapaa tarjota käyttäjille mahdollisuus tutustua ohjelmiston lähdekoodiin, muokata sitä, välittää sitä edelleen ja luoda uusia siihen perustuvia ohjelmia. Ohjelmakoodin mukana toimitetaan lisenssi, joka määrittelee lähdekoodiin liittyvät oikeudet ja velvollisuudet.

3.8.1 BSD-lisenssi

BSD-lisenssi on alun perin Kalifornian yliopistossa Berkeleyssä kehitetty lisenssi, joka on ehdoiltaan erittäin salliva (Wikipedia 2015). Kolme kohtaa sisältävässä lisenssissä ainoat vaatimukset ovat, että lisenssiteksti sisällytetään joko binäärimuodossa tai lähdekoodina levitettävään ohjelmistoon, eikä ohjelmiston kehittäjien nimiä käytetä ohjelmiston pohjalta kehitettyjen uusien ohjelmien markkinoinnissa ilman etukäteen saatua kirjallista

lupaa. Lisenssi sallii kaupallisten ohjelmistojen tekemisen avoimeen ohjelmaan pohjautuen siten, että uuden ohjelman lähdekoodia ei julkaista.

BSD-lisenssillä on julkaistu esimerkiksi Flask, jonka lisenssi on liitteessä 1.

3.8.2 MIT-lisenssi

MIT-lisenssi on BSD-lisenssiin verrattuna vielä hieman sallivampi, koska lisenssi ei sisällä kieltoa kehittäjien nimien käyttämisestä markkinoinnissa. Käyttäjälle annetaan vapaat oikeudet mm. käyttää, kopioida, muokata, julkaista ja myydä ohjelmistoa. Ohjelman pohjalta luotujen uusien ohjelmien lähdekoodin julkaisua ei vaadita, mutta lisenssiteksti pitää aina liittää ohjelmiston tai sen merkittävän osan mukaan.

MIT-lisenssillä on julkaistu esimerkiksi AngularJS, jonka lisenssi on nähtävillä liitteessä 2.

3.8.3 GNU GPL-lisenssit

GPL-lisenssi on kehitetty alun perin GNU-projektin tarpeisiin yhdenmukaistamaan eri ohjelmien erilaiset lisenssit. GPL-lisenssi vaatii, että kaikki ohjelmaan pohjautuvat uudet ohjelmat on myös julkaistava samalla lisenssillä, eli vaatii uusiltakin ohjelmilta avoimuutta lähdekoodin suhteen. GPL-lisenssin uusin versio on versio 3, joka on julkaistu vuonna 2007. GPL-lisenssi eri versioineen on käytetyin avoimen lähdekoodin lisenssi. (Wikipedia 2017)

GNU GPL-lisenssin teksti on MIT- tai BSD-lisensseihin verrattuna erittäin pitkä. Lisäksi koska käytössä on eri versioita, tekstin liittäminen tähän opinnäytetyöhön ei ole mielekästä. Lisenssin eri versiot ovat luettavissa GNU-projektin sivuilta. (Free Software Foundation 2016.)

MongoDB on julkaistu GPLv3-lisenssistä hieman muokatulla AGPL-lisenssillä, joka vaatii, että myös verkon kautta käytettäväksi tarjottavan ohjelman lähdekoodi on oltava saatavilla, vaikka ohjelmaa ei sinänsä levitetäkään.

3.8.4 Muut lisenssit

Monet avoimen lähdekoodin projektit käyttävät jotain muuta lisenssiä, joka on joko muokattu yleisistä lisensseistä tai luotu itse. Esimerkiksi Python on julkaistu omalla lisenssillään, ja sen eri osille on eri lisenssejä (Python Software Foundation 2018 b).

4 OHJELMISTON KEHITYS

Ohjelmiston kehitystyö tehtiin neljässä vaiheessa. Ensimmäisessä vaiheessa suunniteltiin ohjelmiston pääasiallinen rakenne sekä tietomallien sisältö, ja valittiin käytettävät teknologiat. Toisessa vaiheessa aloitettiin varsinainen ohjelmointityö palvelinsovelluksesta. Kolmannessa ja neljännessä vaiheessa kehitettiin asiakasohjelmisto.

Ohjelmiston koodi pidetään GitHub-palvelussa projektia varten perustetussa repositoryssä nimeltä ”jussipennanen/vaka-astia”. Samassa paikassa on myös tarvittava dokumentaatio wiki-muodossa.

4.1 Ohjelmiston rakenne

Ohjelmisto päätettiin tehdä erillisinä palvelin- ja asiakassovelluksina, jotta kumpikin osa voitaisiin kehittää toisistaan erillisinä. Rajapintaa tarjoavaa erillistä palvelinsovellusta voidaan myös hyödyntää, vaikka asiakasohjelmisto haluttaisiin tulevaisuudessa kirjoittaa kokonaan uudestaan tai tarjota mobiililaitteisiin erillistä applikaatiota.

Palvelinsovelluksen toteutuskieleksi valittiin Python ja web-sovelluskehykseksi Flask. Tietokantana käytetään MongoDB:tä, johon yhteys Python-ohjelmasta hoidetaan MongoEngine-kirjastoa hyödyntäen. MongoEnginestä on olemassa myös Flaskia varten muokattu versio, mutta tässä projektissa sitä ei päätetty käyttää.

4.2 Tietomallit

Suunnitteluvaiheessa luotiin tietomallit käyttäjälle, ilmoitustauluviestille, yksityisviestille sekä kommentille, tapahtumalle sekä tiedostolle. Koska tietokantayhteydet hoidetaan MongoEnginellä, myös tietomallit luotiin sitä käyttäen. MongoEnginessä luokat – jotka kääntyvät MongoDB-tietokannassa dokumenteiksi – ovat Document-luokan alaluokkia. Kuvassa 3 on nähtävillä yksityisviestin tietomalli.

```
# Private mesasge class. Can be private to single user or group of users
class PrivateMessage(Document):
    title = StringField()
    content = StringField()
    date = DateTimeField()
    author = StringField()
    tags = ListField(StringField()) # for limiting visibility to users
    comments = ListField(EmbeddedDocumentField(PrivateComment))
```

Kuva 3. Yksityisviestin tietomalli

MongoEngine:ssä luokan ominaisuudet määritellään sen omilla luokilla, kuten tekstimuotoista tietoa sisältävällä StringField-luokalla. MongoEngine kääntää nämä luokat MongoDB:n dokumentin kentiksi tallentaessaan objektin tietokantaan dokumentiksi ja takaisin kun dokumentti haetaan tietokannasta objektiksi. Yksityisviestissä oleva ominaisuus ”comments” on määritelty listakenttänä, joka sisältää EmbeddedDocument-tyyppisiä olioita. EmbeddedDocument-luokan alaluokat ovat tietokannassa dokumentteja, jotka eivät esiinny yksinään, vaan toisen dokumentin osana.

Kaikki tietomallit pyrittiin luomaan siten, että niissä ei ole suoria viittauksia toisiinsa. Viittauksien käyttö olisi hidastanut tietokantahakuja kaiken tarvittavan tiedon hakemisen vaatiessa useamman haun. Viittauksia olisi voinut käyttää viittaamalla käyttäjään tallennettaessa dokumentin tekijä tai tagit olisi voinut tallentaa omaan tietorakenteeseensa. Mikäli olisi ollut tarve tietää vaikkapa dokumentin tekijästä muutakin tietoa kuin nimi, olisi viittaus dokumentista käyttäjään ollut aiheellinen, ja lisätiedon hakumahdollisuus olisi ollut tärkeämpää kuin tietokantahakujen optimointi. Tässä tapauksessa kuitenkin katsottiin, että dokumentin tekijän tiedoista nimen liittäminen suoraan dokumenttiin riittää, eikä viittausta tarvita. Samoin tagit tallennetaan suoraan dokumenttiin.

4.3 **Palvelinsovellus**

Palvelinsovellus kehitettiin Python-kielellä käyttäen Flask-sovelluskehystä. Sovelluksen tarjoamat reitit määriteltiin Flaskin tarjoamalla decoratorilla, jolla asiakkaan tekemät pyynnöt ohjataan oikealle funktiolle käsiteltäväksi. Monessa tapauksessa käsittelyfunktiolle ei tarvinnut välittää pyyntöä kokonaisuudessaan. Kuvassa 4 näkyy yhden sovelluksen tarjoaman reitin määrittely.

Kuvassa 4 näkyvä @-merkillä alkava rivi on edellä mainittu decorator, joka sisältää määrittelyn sekä reitistä (’/bulletin’) että http-metodista joita kutsuttaessa käytetään alla määriteltyä funktiota pyynnön käsittelyyn. Käyttäjän todennus hoidetaan sovelluksessa ennen kuin pyyntö välitetään käsittelyfunktiolle. Funktio enforce_login palauttaa toden, mikäli pyynnössä on mukana voimassa oleva token. Esimerkkitapauksessa vaaditaan lisäksi, että käyttäjä kuuluu henkilökuntaan määrittelemällä funktiokutsussa avainsana-argumentti faculty todeksi.

Käsittelyfunktiot määriteltiin palauttamaan Pythonin dictionary-muotoinen tieto. Kuvassa 4 näkyvässä määritelmässä funktio post_bulletin_message -funktio käsittelee pyynnön (request) ja palauttaa tuloksen, joka muutetaan JSON-muotoon ja palautetaan asiakkaalle.

```python
# Route for posting a new bulletin board message
@server.route('/bulletin', methods=["POST"])
def bulletin_post():
    if enforce_login(request, faculty=True):
        return jsonify(post_bulletin_message(request))
```

Kuva 4. Ilmoitustauluviestin tallennusreitti

4.3.1 Käyttäjän todennus palvelinsovelluksessa

Käyttäjänimenä jokaisella käyttäjällä toimii sähköpostiosoite. Käyttäjän tietoihin tietokannassa tallennetaan myös satunnaisesti luotu salt-arvo sekä salasanasta ja salt-arvosta muodostettu MD5-tiiviste. Salasanaa ei sellaisenaan siis tallenneta tietokantaan. Käyttäjistä tallennetaan myös tieto siitä, onko kyseessä henkilökuntaan kuuluva käyttäjä vai muu käyttäjä.

Sisään- ja uloskirjautumista varten palvelinsovelluksessa on omat reittinsä. Sisäänkirjautumisen onnistuessa sovellus palauttaa käyttäjälle tokenin, joka tallennetaan myös tietokantaan käyttäjän tietoihin. Samalle käyttäjälle voidaan myöntää useampia tokeneja. Tokenin perusteella käyttäjä tunnistetaan kaikissa muissa pyynnöissä.

Uloskirjautumisen yhteydessä voidaan poistaa käytöstä vain se token, jota käyttäen uloskirjautumispyyntö tehtiin tai kaikki saman käyttäjän tokenit.

4.3.2 Ilmoitustauluviestit

Ilmoitustauluviestien käsittely on yksinkertaisinta, koska niiden näkyvyyttä ei rajoiteta käyttäjäkohtaisesti mitenkään. Palvelinsovellus sisältää reitit uuden viestin lisäämiselle, viestin poistamiselle, viestilistan hakemiselle sekä yksittäisen viestin hakemiselle.

Viestin lisääminen ja poistaminen on mahdollista vain henkilökuntakäyttäjille, joten käyttäjän tyyppi varmistetaan ennen näiden pyyntöjen käsittelyä. Muut käyttäjät voivat hakea suppeat tiedot sisältävän listan viesteistä ja yksittäisen viestin kaikki tiedot.

Ilmoitustauluviestin lisäämiseksi tehtävän POST-pyynnön pitää sisältää JSON-muotoisena ilmoitustauluviestin tiedot. Samaa tyyliä käytetään myös muissa tiedon lisäämispyynnöissä.

4.3.3 Yksityisviestit

Yksityisviestien käsittely eroaa ilmoitustauluviesteistä kolmella tavalla: yksityisviestien näkyvyyttä on mahdollista rajata tageilla, yksityisviesteihin on mahdollista lisätä kommentteja, ja myös muut kuin henkilökuntakäyttäjät voivat lisätä yksityisviestejä.

Jokaiseen käyttäjään on tietokannassa mahdollista liittää lista tageja, ja muille kuin henkilökuntakäyttäjille tämä lista on käytännössä pakollinen. Yksityisviesti annetaan käyttäjälle vain, jos käyttäjän tiedoissa ja viestin tiedoissa on vähintään yksi yhteneväinen tagi. Viestilistaa tai yksittäistä viestiä haettaessa varmistetaan, että käyttäjä on henkilökuntakäyttäjä, tai tagivaatimus täyttyy. Jos muu kuin henkilökuntakäyttäjä lisää yksityisviestin, varmistetaan lisäksi, ettei viestissä ole sellaisia tageja joita käyttäjällä itsellään ei ole.

Kommenttien lisäämisessä tagivaatimus otetaan myös huomioon, jotta käyttäjä ei voi kommentoida viestiä, jota hän ei näe. Kommenttien näkyvyyttä ei ole rajattu erikseen, vaan käyttäjä voi nähdä kaikki kommentit, jotka liittyvät hänen näkemäänsä viestiin. Sovellus tarjoaa reitit kommentin lisäämiselle ja poistamiselle.

4.3.4 Tapahtumat

Tapahtumia käsitellään näkyvyyden rajoituksen osalta samalla tavalla kuin yksityisviestejä. Tämän lisäksi tapahtumilla olevan alkamisajankohdan perusteella on mahdollista hakea luettelo vain tietyllä aikavälillä alkavista tapahtumista. Sovelluksessa on reitit tapahtumalistan hakemiselle, yksittäisen tapahtuman tietojen hakemiselle, tapahtuman lisäämiselle ja tapahtuman poistamiselle. Kuten ilmoitustauluviesteissäkin, tapahtuman lisääminen ja poistaminen on sallittua vain henkilökuntakäyttäjille.

4.3.5 Tiedostot

Tiedostojen lisäämistä, poistamista, listausta ja noutamista varten on omat reittinsä. Lisääminen ja poistaminen on rajoitettu vain henkilökuntakäyttäjille. Tiedostot sisältävät myös tag-ominaisuuden näkyvyyden rajaamiseksi. Tiedostojen lisääminen eroaa muun tiedon lisäämisestä siten, että POST-pyyntö tehdään multipart/form-data -tyyppisenä, ja tiedoston metatiedot annetaan URL-parametreina. Yksittäisen tiedoston nouto palauttaa tiedoston datan, eikä JSON-muotoista vastausta kuten muissa hakupyynnöissä.

4.4 **Palvelinsovelluksen testiympäristö**

Palvelinsovellusta varten pystytettiin Ubuntu Linux-pohjainen palvelin DigitalOcean-palveluun. Sovellus vaatii toimiakseen Python-lisäosat

mongoengine, flask, flask-cors ja dnspython, jotka asennettiin pip-työkalua käyttäen. Lisäosilla oli omia vaatimuksiaan, jotka pip osaa kuitenkin asentaa automaattisesti. Web-palvelimeksi asennettiin Apache2, joka wsgi-moduulia hyväksikäyttäen osaa toimia Python-kielisten ohjelmien palvelimena. Wsgi-moduulista asennettiin Pythonin 3-versiota tukeva versio. Palvelimelle haettiin myös ilmainen ssl-sertifikaatti Linux Foundationin tarjoamasta Let's Encrypt-palvelusta. Sertifikaatin hakemiseen ja asentamiseen on tarjolla CertBot-niminen työkalu, jota käytettiin. Kaikki tarvittavat ohjelmistot löytyivät suoraan Ubuntun pakettikirjastosta.

Tietokannan osalta käytettiin MongoDB Atlas -palvelua, joten tietokantaa ei tarvinnut asentaa paikallisesti. Palvelussa on ilmainen taso, joka rajoituksistaan huolimatta soveltui testikäytön tarpeisiin hyvin.

4.5 **Asiakassovellus**

Palvelinsovelluksen tarjoamia rajapintoja ei ole tarkoitettu käytettäväksi suoraan, vaan käyttöliittymäksi tarvitaan asiakassovellus. Opinnäytetyöprojektin aikana kehitettiin selainkäyttöinen asiakassovellus. Sovellus on toteutettu yhden sivun applikaationa (single-page application) käyttäen AngularJS-ohjelmistokehystä. Yhden sivun applikaation ero perinteisiin erillisiin sivuihin verrattuna on, että sivustolla oleva ohjelmakoodi muuttaa olemassa olevaa sivua eri näkymien toteuttamiseksi, eikä lataa uutta sivua näkymän vaihtuessa. Tästä seuraa, että sovellus toimii enemmän työpöytä- tai mobiilisovelluksen tapaisesti. (Wikipedia 2018 e.)

Sovellus käyttää ngRoute-moduulia sivuston eri osien väliseen reititykseen. Moduulia käyttämällä voidaan määritellä osoitteen polku, johon navigoitaessa käytetään tiettyä AngularJS-sovelluksen controlleria. Jokaiselle sivulle on luotu oma controller, joka sisältää pääosan kyseisen sivun tarvitsemasta ohjelmakoodista. Polkuja määriteltäessä asetetaan käytettävän controllerin ohella myös template-tiedosto, jonka sisältämä HTML-koodi sisällytetään pääsivuun sille varattuun kohtaan. Myös template-tiedostot sisältävät jonkin verran ohjelmalogiikkaa, koska samaa tiedostoa käytetään usean eri näkymän toteuttamiseen. Kuvassa 5 näkyy viestisivun määrittely. Kutsuttaessa polkua ”/messages” käytetään template-tiedostoa ”contentpages/messages.html” ja controlleria nimeltä ”messagesController”.

```
// messages page
.when('/messages', {
    templateUrl : 'contentpages/messages.html',
    controller : 'messagesController'
})
```

Kuva 5. ngRoute-reitin määrittely.

Sovelluksen käynnistyessä pääsivu sekä sovelluksen sisältämä JavaScript-koodi ja AngularJS:n koodi ladataan heti palvelimelta selaimeen. Template-tiedostot sen sijaan ladataan vasta käyttäjän siirtyessä sitä vaativalle sivulle. Tämän sovelluksen yhteydessä template-tiedostojen dynaamisesta latauksesta ei ole suurta tietoliikennekapasiteettia säästävää hyötyä, koska template-tiedostot ovat melko pieniä.

Tietopyynnöt palvelimelle tapahtuvat asynkronisesti, eli ohjelman suoritus ei jää odottamaan palvelimen vastausta. Ohjelmakoodissa tämä tarkoittaa esimerkiksi viestilistauksen hakevan ja käsittelevän ohjelmakoodin jakamista kahteen osaan: ensimmäinen osa lähettää http-pyynnön palvelimelle ja rekisteröi pyyntöön funktion, jota kutsutaan vastauksen saapuessa. Vastausta ei jäädään odottamaan, ja muu ohjelmakoodi voi jatkaa suoritustaan. Kun vastaus saapuu, rekisteröityä funktiota kutsutaan, se käsittelee vastauksen ja suorittaa vaadittavat toimenpiteet.

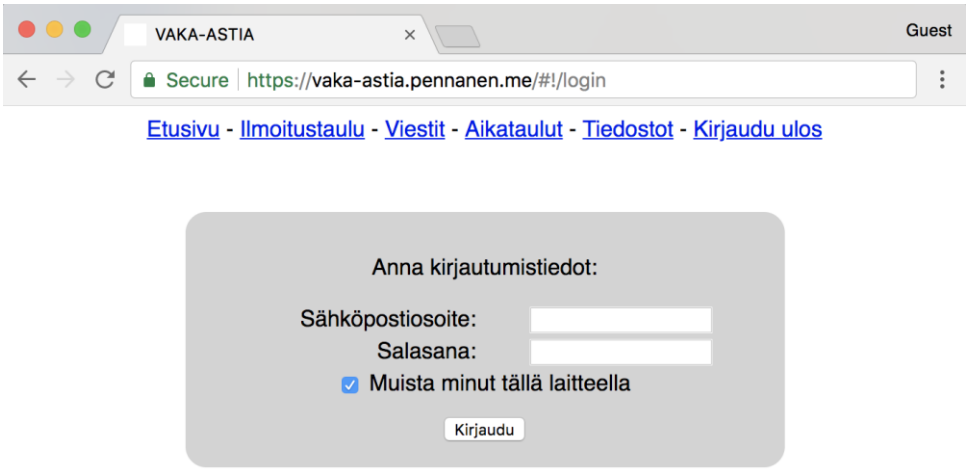
4.5.1 Sisään- ja uloskirjautuminen

Asiakassovellukseen kirjauduttaessa kirjaudutaan itse asiassa sen välityksellä palvelinsovellukseen. Kaikki käyttäjän tunnistamiseen liittyvät toimet hoidetaan palvelimella, ja asiakassovellukseen tallennetaan token, jota käyttäen se voi tehdä tietopyyntöjä palvelimelle. Tokenin ja muun käyttäjäinformaation tallentamiseen käytetään AngularJS:n rootScope-ominaisuutta, jotta tiedot ovat kaikkien eri sivuista vastaavien osien käytössä. Käyttäjä voi myös halutessaan tallentaa tokenin evästeeseen, jotta käyttäjätunnusta ja salasanaa ei tarvitse syöttää seuraavalla kerralla, kun sovellusta käytetään.

Sisäänkirjautumista lukuun ottamatta kaikki muut sivut sisältävät tarkistuksen siitä, onko käyttäjä jo kirjautunut sisään ja onko sovelluksella tiedossaan käyttäjästä kaikki tarvittava tieto. Jos ainoastaan token on tiedossa, sisäänkirjautumiskoodi hakee sitä käyttäen loput käyttäjän tiedot palvelimelta. Jos tokenia ei ole, käyttäjää pyydetään syöttämään käyttäjätunnus ja salasana.

Uloskirjautumisen yhteydessä pyyntö lähetetään tiedossa olevaa tokenia käyttäen palvelimelle, jonka jälkeen asiakassovelluksen tiedoista poistetaan käyttäjän tiedot sekä poistetaan mahdolliset evästeet. Tämän jälkeen käyttäjä palautetaan sisäänkirjautumissivulle.

Kuvassa 6 näkyy kirjautumissivun ulkoasu. Asiakassovelluksen kehityksessä panostettiin ensisijaisesti toiminnallisuuksiin, joten ulkoasu jäi melko karuksi.



Kuva 6. Asiakassovelluksen kirjautumissivu

4.5.2 Ilmoitustaulu

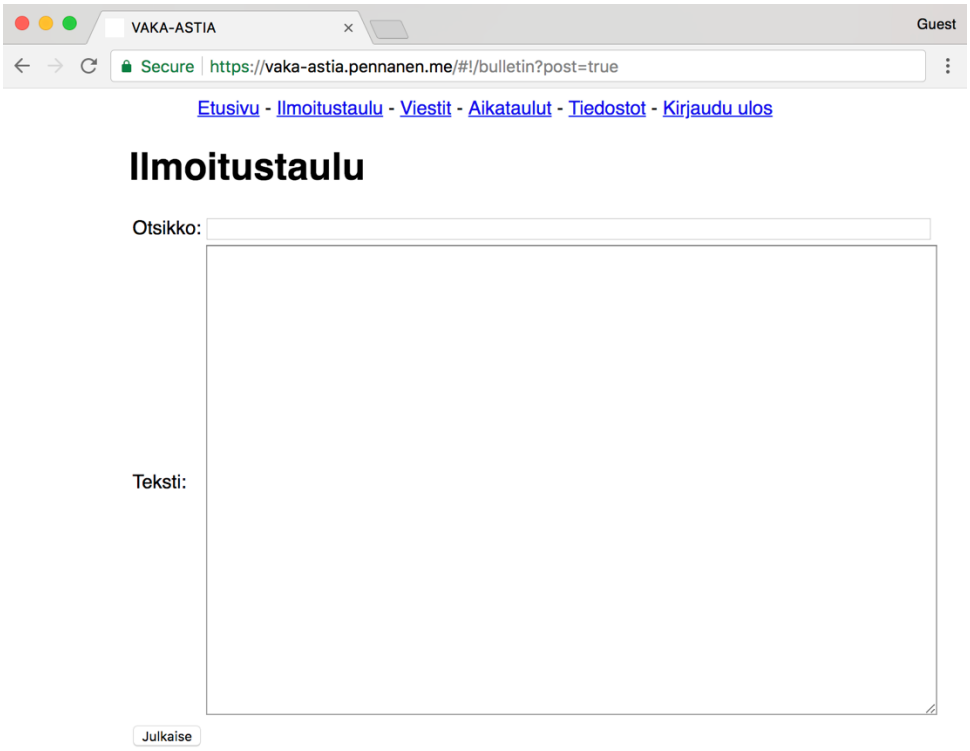
Ohjelman ydintoiminnoista yksinkertaisin on ilmoitustaulu, joten se päätettiin jo suunnitelmavaiheessa toteuttaa ensimmäisenä. Ilmoitustaulua varten toteutettiin tiedotteiden listanäkymä, yksittäisen tiedotteen kokonaisuudessaan näyttävä näkymä sekä uuden tiedotteen kirjoitusnäkymä. Kaikki kolme hyödyntävät samaa controlleria sekä samaa template-tiedostoa. Template-tiedostossa käytetään div-elementtejä ja niissä ng-if -määritystä, jota käyttäen valitaan, näytetäänkö kyseinen elementti sivulla vai ei. Kaikkia kolmea näkymää varten on oma elementtinsä, joista vain yksi näytetään kerrallaan. Listanäkymässä käytetään viestielementillä ng-repeat-määritystä, joka piirtää jokaisesta viestistä oman samanlaisen elementtinsä riippumatta viestien määrästä.

Kuvassa 6 näkyy sekä ng-if- että ng-repeat-määritysten käyttötapa ilmoitustauluviestien listauksessa. Ensimmäisellä rivillä oleva ng-if-määritys tarkastaa, että mode-muuttujan arvi vastaa merkkijonoa ”list”, eli halutaan näyttää listanäkymä. Yksittäinen viesti näytetään div-elementtinä joka alkaa toiselta riviltä. Mallista haetaan taulukko ”messages”, josta jokaista yksittäistä alkiota ”msg” kohden luodaan yksi div-elementti. Kuvassa näkyy myös AngularJS:n tapa hakea mallista näytettävää tietoa näkymään, kuten rivillä 3 haetaan msg-alkion ominaisuus ”title” määrityksellä ”{{msg.title}}”.

```
<div class="messagelist" ng-if="mode == 'list'">
    <div ng-repeat="msg in messages">
        <a class="listmessage" href="#!bulletin?id={{msg.id}}">{{msg.title}}<br>
        <span class="listauthor">Julkaissut: {{msg.author}}</span></a>
    </div>
</div>
```

Kuva 7. ng-if ja ng-repeat

Kuvassa 8 on ilmoitustauluviestin lisäämiseksi käytettävä lomake. Julkaise-painike on kytketty funktioon, joka lukee kenttien tiedot, koostaa niistä JSON-muotoisen pyynnön ja lähettää sen palvelinsovellukselle. Pyynnön onnistuessa käyttäjä palautetaan ilmoitustauluviestien listanäkymään.



Kuva 8. Lomake Ilmoitustauluviestin lisäämiseksi.

4.5.3 Yksityisviestit

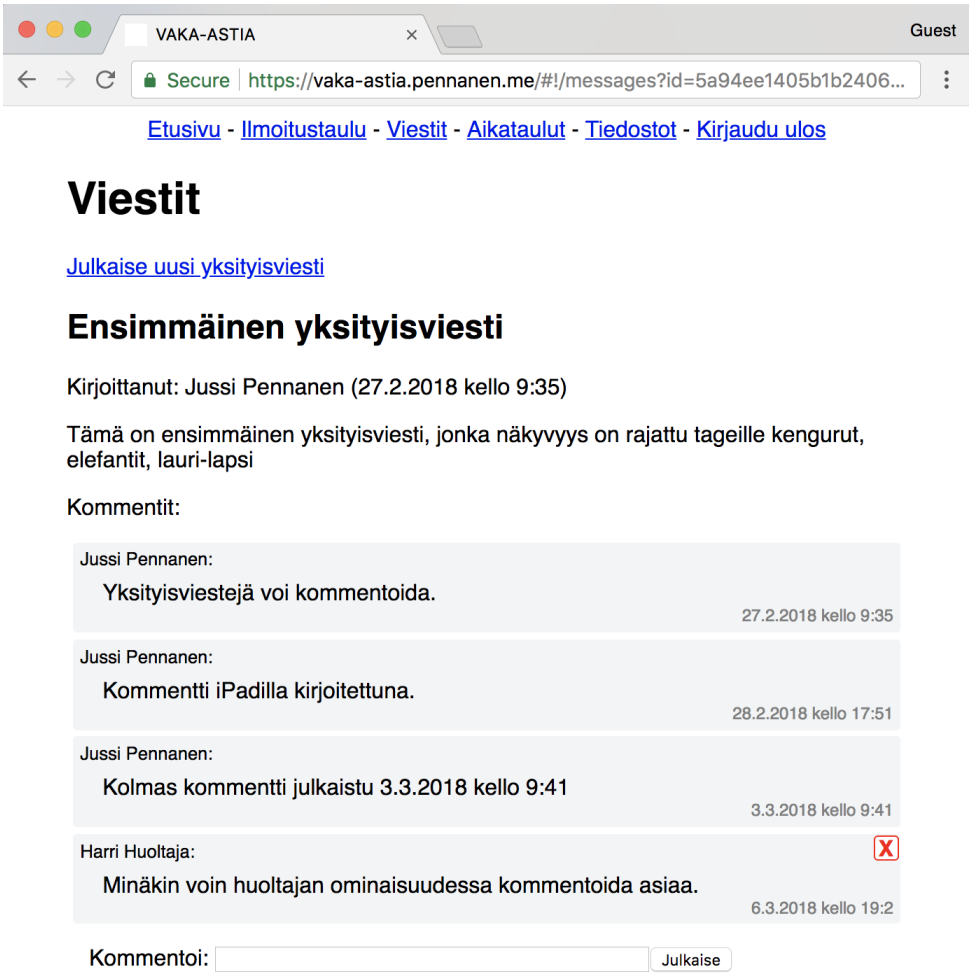
Yksityisviestiominaisuus sisältää samat kolme näkymää kuin ilmoitustaulu. Erona ilmoitustauluun yksityisviestit sisältävät kommentointimahdollisuuden, joka piti ottaa huomioon yksittäisen viestin näyttävässä näkymässä. Kommentin lisääminen viestiin lähettää palvelimelle pyynnön, ja mikäli siihen saadaan hyväksyvä vastaus, lisätään vastauksen sisältämät tiedot dynaamisesti suoraan kommentit sisältävään tietomalliin. Tällä tavalla sivua tai viestin sisältöä ei tarvitse ladata uudelleen kommenttia lisättäessä. AngularJS huolehtii mallia päivitettäessä siitä, että näkymä päivittyy automaattisesti.

Yksityisviestien tallentamisessa on otsikon ja sisällön lisäksi syötettävä myös viestin tagit. Vaikka tagit voivat periaatteessa olla mitä tahansa merkkijonoja, päädyttiin kuitenkin siihen, että tageissa muutetaan syöttämisen jälkeen mahdolliset välilyönnit väliviivaksi. Tagit syötetään pilkulla erotettuina.

Kuvassa 9 on nähtävillä yksityisviesti, jota on kommentoitu. Järjestelmään on kirjauduttu huoltajakäyttäjänä, jolla on oikeus poistaa yksityisviesteistä

ainoastaan itse kirjoittamansa kommentit. Henkilökuntakäyttäjänä kirjauduttaessa punainen rasti, josta kommentti poistetaan, näkyisi jokaisen kommentin kohdalla.

Yksityisviestien kommenttikentän vieressä oleva Julkaise-painike on kytketty funktioon, joka lähettää palvelimelle kommentointipyynnön. Kommentoinnin onnistumisen vahvistavan vastauksen saapuessa kommentti lisätään suoraan näkymään ilman koko näkymän uudelleenlatausta.



Kuva 9. Yksityisviestit.

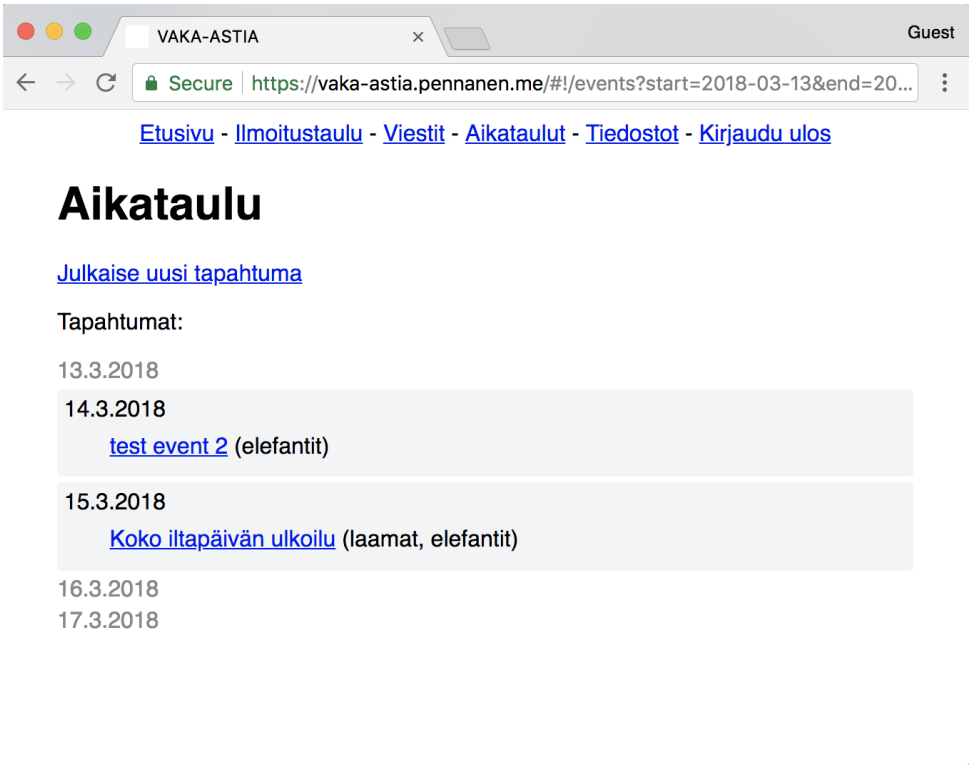
4.5.4 Aikataulut

Aikataulusivulla näytetään kuluvasta päivästä alkaen seuraavan 4 viikon tapahtumat. Ohjelma tunnistaa URL-parametrit start ja end, joilla voi rajata aikataulua asettamalla niille aloitus- ja lopetuspäivät muodossa vuosi-kuukausi-päivä (YYYY-MM-DD). Aikarajaukselle ei kuitenkaan käyttöliittymään luotu valintamahdollisuutta.

Kuvassa 10 näkyy aikataulunäkymä viiden päivän ajalta. Palvelimelta pyydetään halutun aikavälin tapahtumat. Controllerissa luodaan taulukko, jossa on halutun aikavälin päivät Date-muotoisina olioina. Tämän taulukon

yli iteroidaan templatessa, ja jokaisen päivän kohdalla tarkastetaan, onko palvelimelta tulleessa vastauksessa tapahtumia kyseiselle päivälle. Päivät, joissa on tapahtumia, näkyvät korostettuna. Korostus tapahtuu asettamalla ng-class-määrityksellä elementille tietty luokka, jos annettu lauseke tuottaa tosi-arvon, ja toinen luokka muutoin.

Järjestelmään on kuvan 10 tapauksessa kirjauduttu henkilökuntakäyttäjänä, joille näytetään myös linkki uuden tapahtuman luomiseen. Huoltajakäyttäjät eivät tätä linkkiä näe.



Kuva 10. Aikataulunäkymä.

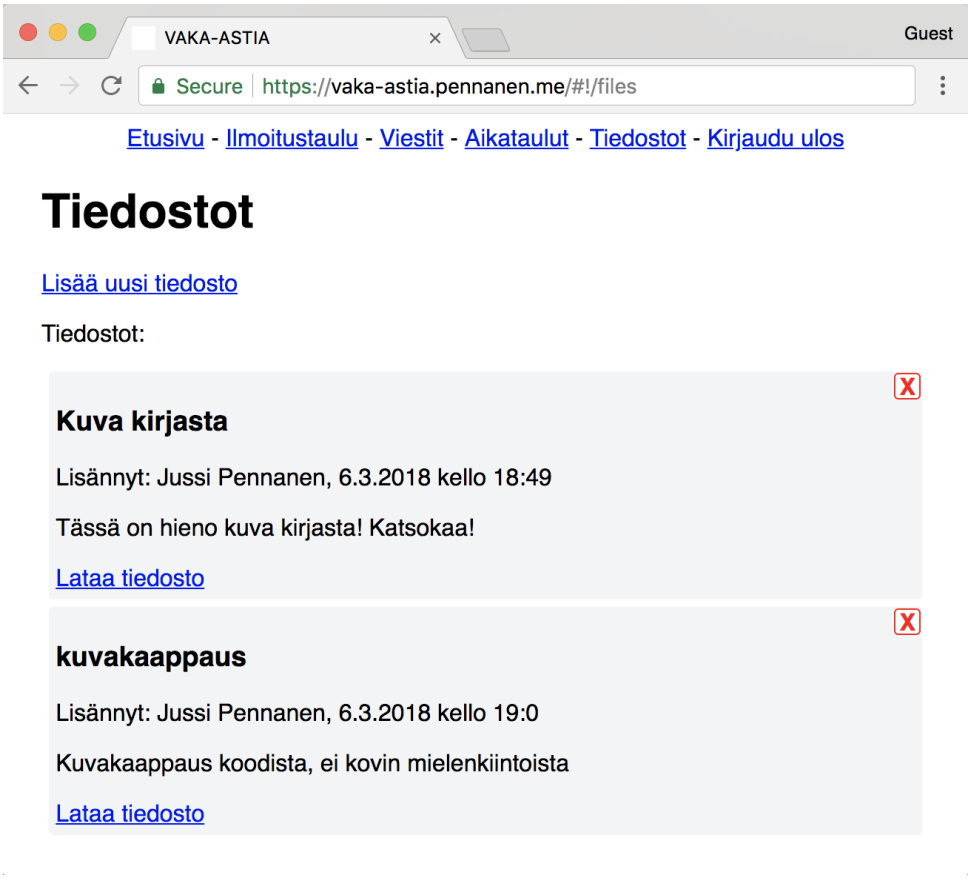
4.5.5 Tiedostot

Tiedostonäkymässä näytetään luettelo käyttäjän näkyvillä olevista tiedostoista tiedoston kuvauksen ja tiedoston latauslinkin kanssa. Yksittäisen tiedoston tarkemmat tiedot näyttävää näkymää ei ole. Henkilökuntakäyttäjille näytetään jokaisen tiedoston poistamiseksi punainen rastipainike. Kaksi tiedostoa näyttävä tiedostonäkymä on nähtävillä kuvassa 11.

Tiedoston latauslinkki osoittaa suoraan palvelinsovellukseen, josta tiettyä reittiä kutsumalla palvelinsovellus palauttaa tietokannasta luetun tiedoston sisällön alkuperäisellä tiedostonimellä.

Tiedostojen tallennus tapahtuu lähettämällä tiedoston sisältävät lomaketiedot palvelinsovellukselle. Palvelimelle ilmoitetaan tässä tapauksessa

myös paluuosoite, johon käyttäjän selain tuodaan pyynnön onnistuessa. Tämä on ainoa kohta, kun asiakassovelluksesta poistutaan käytön aikana.



Kuva 11. Tiedostonäkymä.

5 YHTEENVETO

Ohjelmiston kehitys sujui pääosin suunnitelman mukaisesti. Aikataulussa pysyminen ei tuottanut ongelmia, eikä odottamattomia haasteita kohdattu. Asiakassovelluksen kehityksen aikana jouduttiin palvelinsovellukseen tekemään korjauksia ja muutoksia, jotka olisi ehkä voitu välttää palvelinsovelluksen tarkemmalla suunnittelulla ja testaamisella.

Ohjelmiston kehitystä aloitettaessa oli ajatuksena tehdä Python-projekti, mutta lopulta JavaScript-kielistä ohjelmakoodia syntyi hieman enemmän. JavaScript-koodin määrää olisi varmasti mahdollista vähentää ainakin kirjoittamalla jotkin eri controllereiden sisällä lähes samanlaisena esiintyvät funktiot uudelleen yleiskäyttöisemmiksi, mutta tämä saattaisi vähentää koodin luettavuutta. Asiakassovelluksen osalta koodissa olisi kuitenkin varmasti kehittämisen varaa. JavaScript-koodin kirjoittamisessa oli myös eniten opeteltavaa projektin aikana.

Opinnäytetyöprojektin puitteissa toteutettiin alun perin suunnitellut toiminnallisuudet. Sovellukseen ei tarkoituksellisesti toteutettu muutamia toimintoja, jotka olisivat kaupallista käyttöä varten olennaisia. Ohjelmassa ei esimerkiksi ole käyttäjien hallintaa lainkaan, eikä lisättyjä sisältöjä voi muokata. Ulkopuolisen toimijan tarjoama ohjelmaan pohjautuva palvelu vaatisi joka tapauksessa jatkokehitystä. Ohjelman lisenssi ei tätä estä, mutta toisaalta se ei estä myöskään ohjelman tekijää itseään jatkokehittämistä ohjelmaa ja jättämättä jatkokehityksen tuloksia julkaisematta avoimena lähdekoodina.

Valmista sovellusta testasi henkilö, joka kuuluisi ohjelman suunniteltuun käyttäjäkuntaan. Käyttäjä huomasi sovelluksessa puutteita, joita kehittäjä ei ollut ajatellut eikä alkuperäisessä tarpeiden kartoituksessa tullut ilmi. Jälkikäteen ajateltuna kuvitteellisen asiakkaan pitäminen tiiviimmin kehitysprosessissa mukana olisi saattanut tuottaa paremman lopputuloksen myös tämän projektin osalta.

LÄHTEET

Adenova (2010). Adenova kotisivut. Haettu 8.2.2018 osoitteesta http://www.adenova.fi/homepage/

Dwyer, G. (2017). Flask vs. Django: Why Flask Might Be Better. Blogipostaus 13.2.2017. Haettu 23.1.2018 osoitteesta https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v

Free Software Foundation (2016). Licenses. Haettu 24.1.2018 osoitteesta http://www.gnu.org/licenses/licenses.html

MongoDB Inc. (2018 a.). NoSQL Databases Explained. Haettu 23.1.2018 osoitteesta https://www.mongodb.com/nosql-explained

MongoDB Inc. (2018 b.). MongoDB architecture. Haettu 23.1.2018 osoitteesta https://www.mongodb.com/mongodb-architecture

MongoEngine (2018). MongoEngine. Haettu 23.1.2018 osoitteesta http://mongoengine.org

Mäkinen, M. (2017). *Ketterästi ohjelmistokehitykseen.* Opinnäytetyö. Tieto- ja viestintätekniikka. Hämeen ammattikorkeakoulu. Haettu päivämäärä 12.2.2018 osoitteesta http://urn.fi/URN:NBN:fi:amk-2017100915866

Python Software Foundation (2018 a.). Python General FAQ. Haettu 23.1.2018 osoitteesta https://docs.python.org/3/faq/general.html

Python Software Foundation (2018 b.). History and License. Haettu 24.1.2018 osoitteesta https://docs.python.org/3/license.html

Ronacher, A. (2018). Flask overview. Haettu 23.1.2018 osoitteesta http://flask.pocoo.org/

Visma (n.d.). Wilma – Visma InSchool – Visma. Haettu 8.2.2018 osoitteesta https://www.visma.fi/inschool/wilma/

Vänskä, O. (2015). Haukuttu Wilma saa kilpailijan - tiedot WhatsAppiin jo ensi vuonna? *Tivi.* Haettu 8.2.2018 osoitteesta https://www.tivi.fi/Kaikki_uutiset/haukuttu-wilma-saa-kilpailijan-tiedot-whatsappiin-jo-ensi-vuonna-3329181

Wikipedia (2018 a.). Python (programming language). Haettu 23.1.2018 osoitteesta https://en.wikipedia.org/wiki/Python_(programming_language)

Wikipedia (2018 b.). JSON. Haettu 23.1.2018 osoitteesta https://en.wikipedia.org/wiki/JSON

Wikipedia (2018 c.). Wide column store. Haettu 23.1.2018 osoitteesta https://en.wikipedia.org/wiki/Wide_column_store

Wikipedia (2018 d.). MongoDB. Haettu 23.1.2018 osoitteesta https://en.wikipedia.org/wiki/MongoDB

Wikipedia (2018 e.). Single-page application. Haettu 28.2.2018 osoitteesta https://en.wikipedia.org/wiki/Single-page_application

Wikipedia (2015). BSD-lisenssi. Haettu 24.1.2018 osoitteesta https://fi.wikipedia.org/wiki/BSD-lisenssi

Wikipedia (2017). GNU General Public License. Haettu 24.1.2018 osoitteesta https://fi.wikipedia.org/wiki/GNU_General_Public_License

Wikipedia (2018 e.). AngularJS. Haettu 24.1.2018 osoitteesta https://en.wikipedia.org/wiki/AngularJS

Liite 1

BSD-lisenssi Flask-projektista

Liite 2

MIT-lisenssi AngularJS-projektista