Petri Väänänen

MEVN-mobiilisovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

5.3.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Petri Väänänen MEVN-mobiilisovellus 39 sivua 5.3.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Lehtori Peter Hjort

Insinöörityön tavoitteena oli luoda chat-sovelluksesta prototyyppiversio, jolla pystyttäisiin esittämään laitteisto- ja ympäristöriippumattomuus. Chat-sovelluksen avulla oli tarkoitus tutkia ja testata, kuinka ketterää on rakentaa hybridisovellus hyödyntäen verkkosovellustekniikkaa. Hybridisovelluksen etuna on, että sovellusta ei tarvitse luoda erikseen jokaiselle halutulle mobiilialustalle.

Tutkimusprojektissa käytettiin MEVN-sovelluskokoelmaa, joka ei ole käyttöjärjestelmä- tai laitteistoriippuvainen. MEVN-sovelluskokoelma rakentuu MongoDB:stä, Express.js:stä, Vue.js:stä ja Node.js:stä. Valmis sovellus siirrettiin Apache Cordova -kehitystyökalulla mobiilialustalle, ja näin tuotiin myös natiiviominaisuuksia sovellukseen. Cordova loi rajapinnan kehittäjälle mahdollisuuden käyttää Javascript-ohjelmointikieltä laitteistokomponenttien ja alustan natiiviominaisuuksien kutsumiseen.

Sovelluksen prototyypin rakentaminen MEVN-sovelluskokoelmaa käyttäen onnistui hyvin. Sovelluksen käyttöliittymä toimi selaimessa ja Android-mobiililaitteissa hybridisovelluksena. Hybridisovelluksen haastavampia vaiheita oli saada mobiililaitteiden natiivitoiminnot toimintakuntoon, kuten esimerkiksi kameran toiminnot ja tiedostojen lukeminen. Haasteista huolimatta halutut toiminnot saatiin toimimaan.

Alustojen välillä oli kuitenkin huomattavia eroja rajapintatoimintojen määrissä. Esimerkiksi äänenvoimakkuuden painikkeiden tiloja voitiin kuunnella vain Android- ja BlackBerry-alustalla. IOS- ja Windows-alustoilla tämä ei onnistunut. Mikäli kaikkia laiteominaisuuksia ei tarvita sovelluksessa, on hybridisovelluksen rakentaminen oiva tapa rakentaa mobiilisovelluksia usealle mobiililaitteelle samanaikaisesti.

Avainsanat	mevn, mean, mobiilisovellus, hybridisovellus, rest

Author Title Number of Pages Date	Petri Väänänen MEVN mobileapp 39 pages 5 March 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Peter Hjort, Senior Lecturer

The aim of this theses was to create a prototype of the cross-platform chat software and to research and analyze how agile it is to build a hybrid application by using web application development tools. Creating a hybrid application means that there is no need to create the chat software for each desired mobile platform.

The main objectives of the thesis were to study the different components of the MEVN Javascript software stack. MEVN stack stands for collection of Javascript-based technologies: MongoDB, Express JS, Vue JS and Node JS. While the LAMP stack is locked the operating system, the MEVN stack has no such restrictions.

When the application was ready, it was moved to mobile platform by using Apache Cordova tool. The resulting applications are hybrid which means that they are neither truly native mobile application nor purely Web-based.

The thesis concludes that building a prototype application using the MEVN stack was a success. The application's user interface is worked same way in the browser and on the mobile device as a hybrid application. Most challenging part of building a hybrid application was getting the native components to work correctly. Also differences between mobile platforms where notable. An example is that is volume controller which didn't work at all with iOS or Windows Phone platforms but with Android it worked.

Keywords	mevn, mean, mobile app, hybrid app, rest

Sisällys

Lyhenteet

MEVN	Mongo Database, Express JS, Vue JS, Node JS
LAMP	Linux, Apache, MySQL, PHP
DOM	Document Object Model. Dokumentti
MIT	Massachusetts Institute of Technology
API	Application programming interface
RDBMS	Relational database management system. Relaatiotietokanta
SQL	Structured Query Language. Rakenteellinen kyselykieli relaatiotietokannoille.
NoSQL	Not only SQL tai non SQL. Viitataan tietokantoihin, joilla ei ole taulujen välisiä suhteita, joita relaatiotietokannoissa on. Tietokannat saattavat tukea SQL-kyselykieltä.
JSON	JavaScript Object Notation. Tiedonsiirtoformaatti, johon tiedot tallennetaan tekstimuotoisena.

1 Johdanto

Mobiilisovelluksien ja verkkosivustojen tarkoituksena on luoda ja tarjota tietoa tehokkaasti ja reaaliaikaisesti. Yhä useammin verkkosivustot rakennetaan mobiili ensin -ajatuksella (mobile first), sillä mobiilikäyttäjien määrä kasvaa verkkosivustoilla. [1.] Koska mobiilikäyttäjien määrät ovat kasvaneet, myös hakukoneindeksointimenetelmiä kehitetään jatkuvasti mobiililaiteystävällisimmiksi. [2.]

Mobiili ensin -ajatusmallin tarkoituksena on suunnitella verkkosovellukset niin, että sisältö on helposti saatavissa ensisijaisesti mobiililaitteissa. Se ei kuitenkaan tarkoita samaa kuin responsiivinen suunnittelu. Responsiivisessa suunnittelussa lähestymistapana on suunnitella ulkoasu ensin isoille näytöille, josta ulkoasua skaalataan tarpeen mukaan pienemmille laitteille. Mobiili ensin- ja responsiivisessa suunnittelussa molemmissa on kuitenkin tavoitteena saada ulkoasusta ja tekniikasta mahdollisimman joustavaa ja sisällön tuotosta vaivatonta.

Sovelluskehittäjän näkökulmasta asiat eivät ole näin suoraviivaisia. Edellä mainitut seikat ovat käyttöliittymäpuolen tehtäviä, joissa saadaan ulkoasu toimimaan kaikille laitteille. Harvemmin on puhuttu palvelinsovelluksen skaalautuvuudesta eri laitteille samalla tavalla siinä määrin, kuin käyttöliittymän suunnittelua voidaan tehdä. Nykyään hyvin monet työpöytäsovellukset ovat siirtyneet verkkopalveluiksi, kuten Microsoftin Office-palvelu. Ennen se asennettiin erikseen Windows-tietokoneelle, ja jos Office-työtä haluttiin jatkaa toisella tietokoneella, esimerkiksi Applen tuotteilla, oli Office myös asennettava koneelle erikseen. Microsoft on joutunut kehittämään molemmille käyttöjärjestelmille oman sovelluspakettinsa. Nykyään voidaan käyttää samaa Office-pakettia kaikilla alustoilla, eli sovellus on alustariippumaton.

Insinöörityön lähtökohtana oli luoda prototyyppiversio chat-sovelluksesta, jolla pystyttäisiin toteuttamaan laitteisto- ja ympäristöriippumattomuus. Insinöörityön tavoitteena oli chat-sovelluksen avulla tutkia, kuinka ketterää on rakentaa sovellus alustariippumattomaksi sekä perinteisille tietokoneille että mobiililaitteille.

Insinööriraportti on rakennettu osiin, joista ensimmäisessä perehdytään moderneihin verkkosovelluksien kehitystyökaluihin, kuten MEVN stack -työkalupakettiin. Seuraavaksi

käydään läpi mobiilisovelluksen kehitystyökaluja. Lopuksi esitellään tehtyä sovelluskehitystä ja sen vaiheita ja tehdään loppuyhteenveto tutkimuksesta.

2 MEVN-sovellus

Perinteinen web-sovellus on kehitetty LAMP-sovelluskokoelmalla. LAMP-sovelluskokoelmassa käyttöjärjestelmänä toimii Linux, web-palvelimena Apache, relaatiotietokantajärjestelmänä MySQL tai MariaDB ja ohjelmointikielenä PHP, Perl tai Python. Insinöörityöprojektissa käytettiin MEVN-sovelluskokoelmaa, joka ei ole käyttöjärjestelmästä tai laitteistosta riippuvainen. MEVN-sovelluskokoelma rakentuu MongoDB:stä, Express.js:stä, Vue.js:stä ja Node.js:stä.

MEVN-sovelluskokoelma:

- MongoDB toimii sovelluskokoelman tietokantana.
- Express.js on minimalistinen ja joustava sovelluskehys (engl. framework) Node.js:lle.
- Vue.js on sovelluskehys käyttöliittymän kehittämiseen.
- Node.js on palvelinympäristössä suoritettava verkkosovellus.

2.1 MongoDB-tietokanta

MongoDB on avoimen lähdekoodin NoSQL-tietokanta, joka tallentaa joustavasti dokumentteja JSON-tapaiseen tiedostomalliin. Dokumenttien rakennetta voidaan muokata nopeasti tarpeen mukaan siten, ettei se vaikuta muiden dokumenttien rakenteisiin. Dokumentteja voidaan tallentaa yhdestä moneen kenttään. Kenttä voi sisältää binäärejä, taulukoita tai lapsidokumentteja. [3.]

MongoDB tarjoaa myös erittäin kattavan ajurituen monille suosituille ohjelmointikielille sekä sovelluskehyksille. Tuettuja kieliä ovat muun muassa Java, Javascript, .NET, Python, Perl ja PHP. Tuki löytyy reilulle 30 muulle kielelle yhteisön tekemänä. [3.]

Vaihtoehtoisia NoSQL-tietokantoja ovat esimerkiksi Apache Software Foundationin luomat Cassandra- ja HBase-tietokannat. Cassandran ideaalinen käyttötilanne on, kun tietomassat ovat suuria ja tiedot rakenteettomia. Tällaisten tietomassojen hallitseminen luotettavasti Cassandralla onnistuu kätevästi lähestyttävällä SQL-tapaisella kyselykielellä.

HBase-tietokanta on suunniteltu myös isoille big data -tietomassoille. Pääasiallinen tapa käyttää big dataa on tallentaa valtavia määriä tietomassoja poistamatta juuri lainkaan tietoa tietomassasta. Tietomassaa tarkastellaan ja analysoidaan myöhemmin. Mongodb on näihin nähden helppo ja heti valmis käytettäväksi tietokannan pystytyksen jälkeen. Myös Mongodb:n tapaisia JSON-pohjaisia tiedostomalleja käyttäviä tietokantoja ovat Apachen CouchDB sekä RethinkDB.

NoSQL- ja SQL-tietokantojen suurimmat erot ovat niiden käyttötavassa, ja sen kautta tulevat nopeudet. Molemmat tietokannat ovat todella nopeita, jos niiden käyttö on oikeanlaista. SQL-tietokanta on hyvä vaihtoehto, jos tietomassa on hyvin kompleksista, kuten varausjärjestelmät. NoSQL sopii hyvin mobiilisovelluksiin tai esineiden internetiin. Mobiilisovellukselle riittää pieni tietokanta, jota voidaan helposti laajentaa, kun käyttäjäkunta kasvaa ja sovellus monipuolistuu. Esineiden internet tuottaa paljon puolistrukturoituja telemetriatietoja, jotka ovat XML- tai JSON-muodossa, joista saattaa puuttua osa arvoista tai arvoja on enemmän kuin tarvitaan.

2.1.1 Käsitteitä

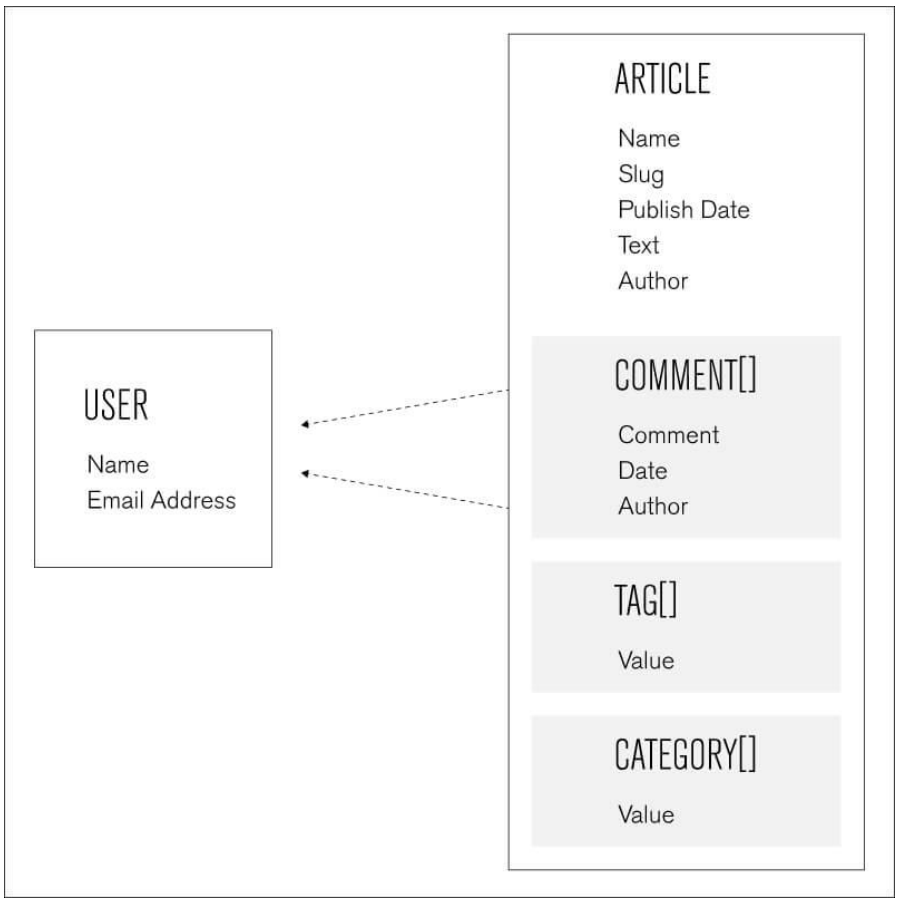
MongonDB-tietokanta rakentuu yksinkertaisuudessaan kokoelmista, jotka vastaavat relaatiotietokannassa tauluja. Dokumentit ovat rivejä, ja kentät ovat sarakkeita. Kentät voivat sisältää BSON-serialisoituja binääreitä, taulukoita tai lapsidokumentteja. Dokumenttien ei eivät tarvitse olla rakenteeltaan samanlaisia keskenään. Dokumentteja voidaan rakentaa ja laajentaa hyvin vapaasti ilman, että tietokannan alasajoja joudutaan tekemään. [Taulukko 1.]

Taulukko 1. Relaatiotietokannan ja Mongodb:n termit.

SQL	**MongoDB**
tietokanta (database)	tietokanta (database)
taulu (table)	kokoelmat (collections)
rivi (row)	dokumentti tai BSON (document)
sarake (column)	kenttä (field)
indeksi (index)	indeksi (index)
taulukoiden liitäntä (joins)	sulautetut kokoelmat (embedded documents)
pääavain (primary key)	pääavain (primary key)
pääavaimen valinta vapaa	_id-kenttä on automaattisesti pääavain
aggregaatit	aggregaatti pipeline

Tietokanta tukee indeksointia pääavaimella ja toissijaisella avaimella, jolloin päästään tehokkaasti käsiksi haluttuun dokumenttiin ja sen kenttiin. Jokainen dokumentti voidaan indeksoida, samoin sen sisältämät kentät, mukaan lukien taulukot ja lapsidokumenttikentät.

Relaatiotietokannan ja MongoDB:n tietokannan mallien eroja voidaan kuvata helposti artikkelin avulla [kuva 1]. Relaatiotietokannassa tietokannan taulut rakentuisivat viidestä taulusta: kategoriat, artikkeli, kirjoittaja, avainsanat ja kommentit. MongoDB:n mallissa tässä tapauksessa olisi vain kaksi dokumenttia. Ensimmäinen olisi artikkeli-dokumentti, joka sisältää kategoriat, avainsanat ja kommentit. Toisessa dokumentissa olisi kirjoittaja.



Kuva 1. Esimerkki artikkeli-dokumentista [3].

2.1.2 Replikointi

Tietokannan replikoinnilla saadaan luotettavuutta tietomassojen säilyttämiseen. Replikoitaessa tietokantaa on tietokanta jaettava vähintään kahteen toissijaiseen tietokantaan. Suositeltava määrä on kolme replikaattia tietokannasta. Replikoidut tietokannat voivat olla samalla palvelimella tai hajautettuna muille palvelimille. Hajautettaessa eri palvelimille luotettavuus kasvaa tietojen saatavuuden ja nopeuden osalta, kun puhutaan maantieteellisistä etäisyyksistä, jolla voidaan saada latenssia minimoitua tiedonsiirrossa. Hajautettuilla tietokannoilla voidaan myös minimoida verkkoliikenteen ruuhkautuminen palvelimissa.

Replikoitaessa voidaan MongoDB:ssä kopioida alustavasti koko tietokanta, minkä jälkeen kopioidaan pelkästään muutokset [4]. Relaatiotietokannoissa voidaan yleisesti ottaen replikoida koko tietokanta tai yksittäisiä tauluja [5]. Replikointi on yksisuuntainen, mikä estää konfliktien muodostumista ensisijaisen ja toissijaisen tietokannan välillä.

MongoDB:ssä ensisijaista tietokantaa käytetään oletuksena tietojen kirjoittamiseen ja lukemiseen, myös tietokannan replikaattia voidaan käyttää lukemiseen [6]. MongoDB-replikointi on sisäänrakennettu ominaisuus ja täysin automatisoitu, joten tietokannan alhaallaoloajat ja virhetilanteet eivät vaikuta kriittisesti tietokannan toimintaan [7]. Mikäli ensisijainen tietokanta vioittuu tai se ei vastaa, toissijaisesta replikaatista tehdään ensisijainen tietokanta automaattisesti.

2.1.3 Tietokantakyselyt

Yksi tärkeimmistä avaintekijöistä MongoDB:n käytölle on sen joustavuus kyselyissä, sillä kyselyistä voidaan rakentaa monimutkaisia käyttäen toissijaisia indeksointeja, aggregaatteja ja dokumenttien yhdistämistä [3].

- Avain-arvokyselyllä voidaan palauttaa mikä tahansa kenttä, joka dokumentista löytyy.

- Vertailuoperaattorilla voidaan palauttaa seuraavilla tavoilla: suurempi kuin, pienempi kuin tai yhtä suuri kuin.

- Valintaehtokyselyillä palautetaan yleensä ryhmä tuloksia totuusarvoilla (ja, tai, ei).

- Paikkatietokysely palauttaa läheisyyden perusteella kriteereihin, risteyksiin, pisteisiin, muotoihin. Esimerkiksi kun lennetään reittilento Helsinki – Ranska, lentääkö lentokone Saksan yli.

- Aggregaattikyselyt prosessoivat dataa ja palauttavat niiden yhteenlasketun tuloksen. Esimerkiksi yhdistetään monta dokumenttia ja palautetaan vain niiden yhdistetty tulos.

- Liitokset ja kaavioiden läpikäynnit. MongoDB ei tue liitoksia, kuten SQL, mutta $lookup-vaiheella voidaan ”liittää” kokoelmia yhteen.

2.1.4 Tietokannan mallinnus

MongoDB-tietokannan joustavuus tulee siitä, ettei rakennetta tarvitse määritellä ennen sen käyttöönottoa, kuten SQL-tietokannassa. Tämä antaa vapaat kädet tietokannan suunnittelulle ja sen optimoimiselle. Tietokannan rakenteen suunnittelussa suositellaan mietittäväksi, miten ohjelmisto tulee käyttämään tietoa ja minkälaisia yhteyksiä eri tiedoilla on toisiinsa nähden. Dokumentit voidaan suunnitella viittaamaan keskenään toisiinsa, tai tieto voidaan niin sanotusti sulauttaa dokumenttiin. [21.]

Sulautettu dokumenttirakenne tuo tietokannalle keskimäärin huomattavaa tehokkuutta lukuoperaatioihin. Sulautetun rakenteen ansiosta dokumentin päivittämiseen tarvitaan vain yksi kirjoitus kerta, kun viittaavien dokumenttien päivittämiseen tarvitaan viitattavien dokumenttien määrä. Sulautetulla rakenteella on kuitenkin huonot puolensa, sillä jos dokumentti kasvaa varatun alueen ylitse, muistissa tapahtuu luonnollisesti tietojen pirstoutumista [8]. MongoDB:n uusimissa versioissa tällainen pirstoutuminen on saatu kuriin käyttämällä Power of two allocationia eli 2. potenssin allokointia muistiin, ennen kuin dokumentti tallennetaan tietokantaan. [9.]

MongoDB ei tue viittauksia samalla tavalla kuin SQL-taulujen välillä. Tiedot tallennetaan denormalisoivalla tavalla tai erillisiin dokumentteihin eri kokoelmaan tai tietokantaan. Denormalisoinnissa tietokannan eheyden ylläpitäminen on hankalampaa, koska samaa tietoa voi esiintyä monessa eri paikassa. Tämän avulla kuitenkin saadaan huomattavaa suorituskykyä, sillä tauluja ei tarvitse yhdistellä, vaan tarvittava tieto on heti saatavilla yhdellä kyselyllä. Dokumenttien yhdistämiseen eri dokumenteista tai kokoelmista käytetään dokumentin uniikkia avainta _id kenttää. Tämä samainen avain on tallennettu myös toiseen haluttuun dokumenttiin. Ensin kysely rakennetaan vain hakemaan pääasiallinen dokumentti, minkä jälkeen haetaan uudelleen seuraava dokumentti edellisen dokumentin _id-kentän arvon avulla. Toisin sanoen jokainen operaattori tehdään atomisella tavalla, eli vain yhteen dokumenttiin voidaan kirjoittaa kerrallaan. Siksi kirjoitus- ja lukusuorituskyky on denormalisointitavalla nopeampaa, kun kaikki on yhdessä dokumentissa. Pitää myös huomata, että lapsidokumentit sulautetulla tavalla liitettynä kuuluvat myös samaan atomioperaattoriin.

2.2 ExpressJS-sovelluskehys

Express on Node.js:lle rakennettu sovelluskehys, joka on tällä hetkellä yksi nopeimmin kasvavista ja suosituimmista Nodelle luoduista sovelluskehyksistä. Kilpaileviakin ohjelmistokomponenttikirjastoja Expressille on, kuten Hapi.js ja Socket.io. Expressin etuihin kuuluu minimalistisuus, jolla voidaan helposti rakentaa verkkosovelluksia, mobiilisovelluksia ja ohjelmointirajapintoja.

Node.js ja Express.js tuovat uuden lähestymistavan rakentaa verkkosivuston palvelinpuoli. Expressin ideana on tuoda Nodelle kevyt ohjelmointirajapinta tai sovelluskehys, jolla voidaan rakentaa ja ylläpitää palvelinsovellusta helpolla tavalla. Express 4.0 -versiossa tuotiin paljon muutoksia, ja yksi niistä on uudistettu Routing- eli reititysominaisuus. Se toi mukanaan API-toiminnallisuudet, kuten use, get, param ja route.

2.3 Vue JS -sovelluskehys

Vue JS on kasvava Javascript-sovelluskehys käyttöliittymän kehittämiseen. Se on vapaa avoimen lähdekoodin ohjelmisto, jolla on MIT-lisenssi. Avoin lähdekoodi tarjoaa käyttäjälle mahdollisuuden muokata, kopioida ja käyttää teosta omien tarpeiden mukaan. Lähdekoodit, joiden käyttöehdot ovat MIT-lisenssin mukaiset, voidaan käyttää ja muokata myös kaupallisissa suljetun koodin ohjelmistoissa. Ainoana ehtona on säilyttää lisenssin teksti lähdekoodissa. Vue JS -sovelluskehys on vahva haastaja käyttöliittymien rakentamiseen Reactin ja Angluarin rinnalle. Vue JS:n ideana on pitää arkkitehtuuri mukautuvana ja kevyenä. Vue JS:n luoja Evan You työskenteli aiemmin Googlen kehittämässä Angular JS -projektissa. Siksi Vue JS on saanut vaikutteita Angular JS:sta, kuten DOM:iin asetettavista Javascript-direktiiveistä (directive), jotka muokkaavat DOM:a direktiivin tilan mukaisesti. [16.]

Vuea on kuitenkin yksinkertaistettu, ja siinä on kasvatettu direktiivien ja komponenttien välistä eroa, kun verrataan Angular JS:ään. Direktiivit on suunniteltu vain DOM:n muovaamiseen, kun komponenttien tarkoituksena on jakaa DOM-kokonaisuus pieniin näkymäosiin, joilla on omat logiikkansa. [17.]

Suorituskyvyssä Vue on hieman ripeämpi Reactiin nähden tavanomaisimmissa tapauksissa, sillä Vuessa virtuaalisen DOM:n toteutus on kevyempi kuin Reactin kehittämä toteutus. DOM ei ole suunniteltu tai edes optimoitu dynaamiseen käyttöön, ja siksi on kehitelty DOM:sta abstrakti versio, jota kutsutaan virtuaaliseksi DOM:ksi. Virtuaalinen DOM tuo suorituskykyä DOM:n muokkaamiseen, mutta sen liittäminen on teknisesti hyvin haastavaa. Vain murto-osa Javascript-sovelluskehyksistä on lyönyt läpi suorituskyvyssä. Haastavuus on siinä, kun kysytään, miten ja milloin virtuaalinen muutos tuodaan oikeaan DOMiin. [10.]

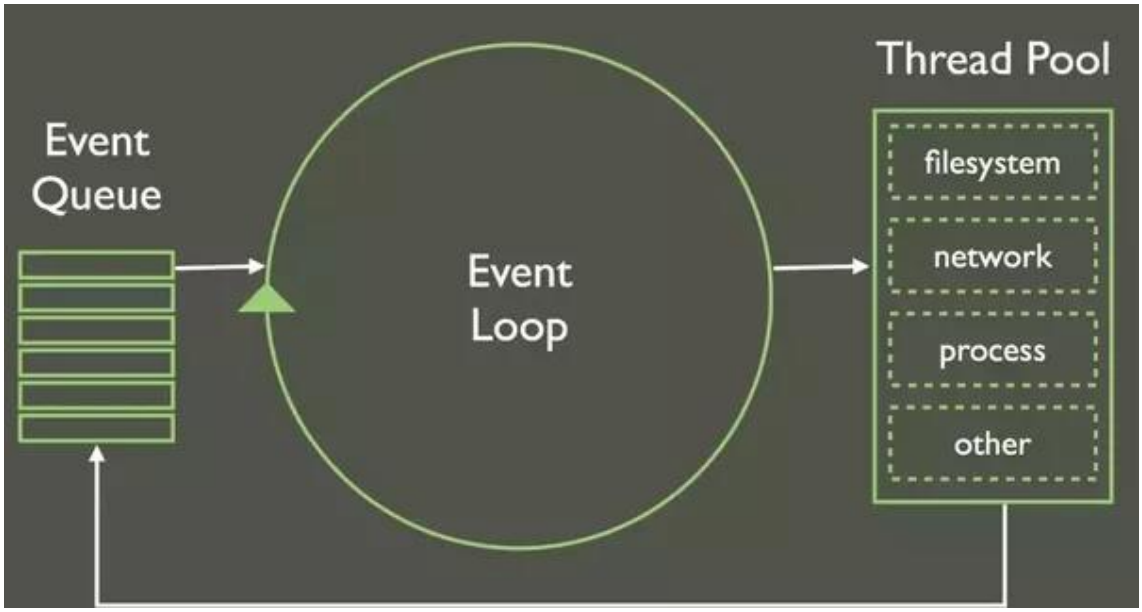
React JS seuraa komponenttien tilojen vaihtumista, minkä jälkeen koko komponentti puun juuresta lähtien piirretään uudelleen. Tämä ei ole suorituskykyisin tapa muokata DOM:a, ja siksi React tuo siihen erilaisia komponenttiratkaisuja. Vuen oma järjestelmä seuraa automaattisesti komponenttien riippuvuuksista ja niiden tiloista, mikä komponentti pitää piirtää uudelleen, kun komponentin tila muuttuu.

2.4 Node JS -palvelinsovelluskehys

Node.js on Ryan Dahlin kehittämä avoimeen lähdekoodiin perustuva palvelimessa suoritettava Javascript-ajoympäristö, joka sisältää http-palvelinympäristön, jonka Ryan Dahl onnistui liittämään Googlen kehittämään V8-Javascript-moottoriin. Javascript-moottori mahdollisti sen, että palvelimissa ja selaimissa pystyttiin käyttämään samaa ohjelmointikieltä. V8-Javascript-moottorin lähtökohtainen käyttötarkoitus oli parantaa selaimen Javascript-suorituskykyä. Sovellusten suorituskyky on myös parantunut palvelimissa, kun verrataan perinteiseen palvelimeen, jossa voidaan käsitellä vain yksi käsky kerrallan. Node.js:llä voidaan käsitellä monta käskyä kerrallaan.

Node.js eroaa muista, perinteisimmistä sovelluskehyksistä nopeuden ja suorituskyvyn osalta. Nopeuden ja suorituskyvyn mahdollistaa Node.js:n käyttämä tapahtumasilmukka- eli ”event loop” -toiminto, joka toimii V8-Javascript-moottorilla. Tapahtumasilmukka on ohjelman sisällä oleva yhdellä säikeellä toimiva rakennelma, joka vastaanottaa ja käsittelee tapahtumia. Node.js:n taustalla käytetään libuv (Unicorn Velociraptor Library) -kirjastoa, jonka avulla Node.js:n I/O-operaatiot suoritetaan asynkronisesti http-palvelun ulkopuolella antaen mahdollisuuden Node.js:lle vastaanottamaan lisää

http-pyyntöjä. Kun taustalle viedyt I/O tapahtumat ovat valmiita, palautetaan I/O-tapahtuman tulokset takaisin tapahtumasilmukkaan, ja tapahtumasilmukka palauttaa operaatiota vastaavan tuloksen takaisin asiakkaalle. [Kuva 2.]



Kuva 2. Node.JS:n tapahtumasilmukka [14].

Asynkronisuudella voidaan siis luoda rinnakkaisajoja ohjelmakoodissa, mikä tarkoittaa sitä, että ohjelmakoodin suorituksessa ei tarvitse jäädä odottamaan yksittäisen ohjelmalohkon suoriutumista vaan voidaan aloittaa jo seuraava suoritettava ohjelmalohko. Kun asynkroninen ohjelmalohko on suoriutunut loppuun asti, suoritetaan takaisinkutsufunktio. Takaisinkutsufunktiota ei suinkaan suoriteta heti, vaan vasta kun Node.js:n tapahtumasilmukka on I/O-takaisinkutsusuoritusvaiheessa. Tässä tilassa Node.js suorittaa takaisinkutsufunktioita niin kauan, kuin niitä riittää tai takaisinkutsusuoritusten määrä ylittyy. [11.]

3 Hybridisovellus

Mobiilisovelluksien kehityksessä käytetään hyvin usein eri laitteille omaa kehitysympäristöä ja työkaluja sekä käännettävää ohjelmointikieltä. Android-alustoilla käytetään Android Studio SDK -kehitysympäristöä [18], Windows-alustoilla Visual Studiota [19] ja Apple-alustoille Xcodea [20]. Android-alustoille kirjoitetaan hyvin usein Android Java-ohjelmointikielellä. Windows alustoille käytetään usein C#- tai VB.NET-kieltä. Apple on tuo-

nut alustoilleen Swift-kielen. Hybridisovelluksen tai Cross-platformin kehittämiseen tarkoitettuja työkaluja on monia: Ionic, Xamarin, React Native, Nativescript, PhoneGap ja Cordova. Xamarin oli ainoa, jolla kirjoitettiin Javascriptin sijasta C#-ohjelmointikielellä. React Nativella voidaan rakentaa täysiverinen natiivisovellus Javascriptillä. Ionic ja Nativescript luottivat AngularJS:iin tai Typescriptiin. PhoneGap on jakeluversio Apache Cordovasta, eli voidaan kuvitella, että Cordova toimii PhoneGapin moottorina [22].

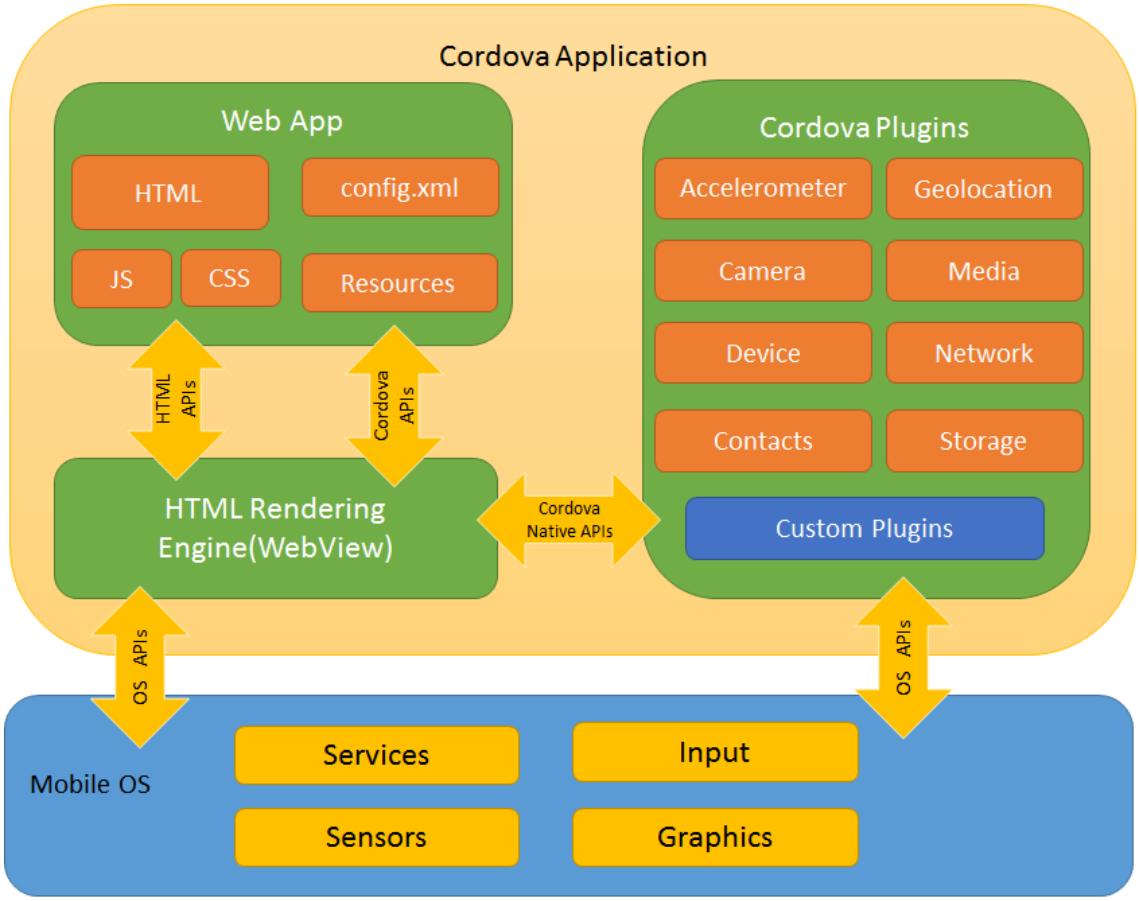
Mobiilisovellukset voidaan jaotella kolmeen kategoriaan: natiivisovellukset, verkkosovellukset ja hybridisovellukset. Natiivisovellus tarkoittaa juuri niitä sovelluksia, jotka on tehty yksinomaan esimerkiksi Android-alustalle tai iOS-alustalle ja jotka ovat ladattavissa niiden kaupoista. Verkkosovellus toimii selaimessa verkkosivuna, mutta se toimii sovelluksena kuten Office 365 tai Facebook. Hybridisovellus on sekoitus natiivisovelluksesta ja verkkosovelluksesta, jolloin sovellus voidaan ladata kaupasta ja asentaa mobiililaitteelle, mutta sovellus on rakennettu verkkosovellustekniikalla. Hybridisovellusta voidaan laajentaa aina tarvittaessa helposti ja nopeasti, ja laitteisto-ominaisuuksiakin voidaan käyttää. [13, s. 28.]

Hybridisovelluksen etuna on, että eri alustoille ei tarvitse luoda erikseen omia sovelluksia, vain yksi riittää. Näin saadaan säästettyä aikaa. Myöskään kehittäjän ei tarvitse olla kaikkien alustojen asiantuntija. Hybridisovelluksen heikkoutena on sovelluksen suorituskyky, sillä WebView-rajapinta hoitaa käyttöliittymän piirtämisen, Javascriptin suorittamisen ja rajapinnan tarjoamisesta myös laitteistoon asti. WebView tarjoaa mobiililaitteen natiivisovelluksen toiminnallisuudet verkkosovellukselle, ja antaa näin vaihtoehtoisen tavan luoda sovellus myös web-standardien mukaisesti.

Ei alustojen käyttöliittymien ulkoasu erot ovat myös hankalia käyttäjien tottumusten takia. Suosituimpien alustojen, Applen iOS ja Goolgen Android, välillä on isoja eroja perustoimintojen ulkoasuissa ja toimintalogiikkojen välillä.

Apache Cordova -Javascript-sovelluskehys on suunniteltu cross-platform-hybridisovelluksien kehitykseen käyttäen HTML5-, CSS- ja Javascript-web-tekniikkaa. Cordova API antaa kehittäjälle mahdollisuuden käyttää mobiililaitteiden ominaisuuksia, kuten sensoreita, tiedostoja ja puhelinnumeroita sekä seurata, käytetäänkö verkkoliikenteessä mobiiliverkkoa vai langatonta lähiverkkoa. Tällä hetkellä Apache Cordova -sovelluskehys tukee seitsemää (7) eri alustaa, Andoid, Blackberry 10, iOS, OS X, Ubuntu ja Windows.

Cordova-hybridisovellus koostuu verkkosovelluksesta, jota laajennetaan Cordova-lisäosilla, jotka luovat rajapinnan mobiililaitteen natiivikomponentteihin. Kommunikointi verkkosovelluksen ja mobiililaitteen välillä tapahtuu Cordova-rajapinnan kautta. [Kuva 3.]



Kuva 3. Cordova-sovellusarkkitehtuuri [15].

Cordovalle voi myös itse kehittää lisäosia, mikäli sen omasta laajasta kirjastosta ei löydy tarvittavaa ominaisuutta valmiina. Cordovan rajapinnan ideana on, että kehittäjän ei tarvitse huolehtia monen eri alustan rajapinnoista tai sovelluksen uudelleen kirjoittamisesta jokaiselle alustalle erikseen.

4 Chat-sovelluksen rakentaminen

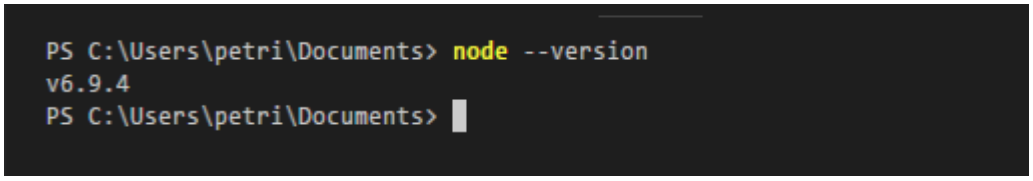
Insinöörityössä oli tarkoitus luoda prototyyppiversio chat-sovelluksesta, jolla pystytään kokeilemaan laitteisto- ja ympäristöriippumattomuus. Tarkoituksena ei ollut rakentaa täy-

siveristä sovellusta vaan tehdä puhdas esittelyversio. Sovelluksen avulla voidaan todentaa ja antaa vastauksia siihen, kuinka voidaan luoda sovellus käyttäen eri laitteistojen ominaisuuksia vain yhdellä versiolla ohjelmistosta.

Työn idea sai alkuunsa, kun it-alan ohjelmistotalot, kuten Microsoft ja Canoncial Ltd, esittelivät konsepteja ekosysteemistä, jossa mobiilikäyttöjärjestelmät ja tietokoneiden käyttöjärjestelmät yhdistetään tai niille tuodaan yhdenmukaisuutta. Microsoft toi Windows 8:n myötä uuden tiilikäyttöliittymän, joka oli optimoitu taulutietokoneille. Uudistuksen ideana oli tuoda helpotusta kosketusnäytöille kosketusnäyttöteknologian suosion kasvaessa. Canonical oli myös aloittanut Ubuntu Touch -projektin, joka tulisi kilpailemaan iPhonen iOS- sekä Android-laitteiden käyttöjärjestelmien kanssa, sillä eroavaisuudella, että Ubuntu Touch olisi täysiverinen PC-käyttöjärjestelmä, johon tehdään mobiililaitteille sopiva käyttöliittymä.

4.1 Asennus

Chat-sovellusprojektin asennus lähti liikkeelle Node.js -Javascript-palvelimen pystyttämisestä Windowsille käyttäen Windows-käyttöjärjestelmälle tarkoitettua asennuspakettia. Node.js:n onnistunut asennus voidaan todeta kirjoittamalla komentokehotteessa `node –version`, joka tulostaa Node.js:n versionumeron. [Kuva 4.]

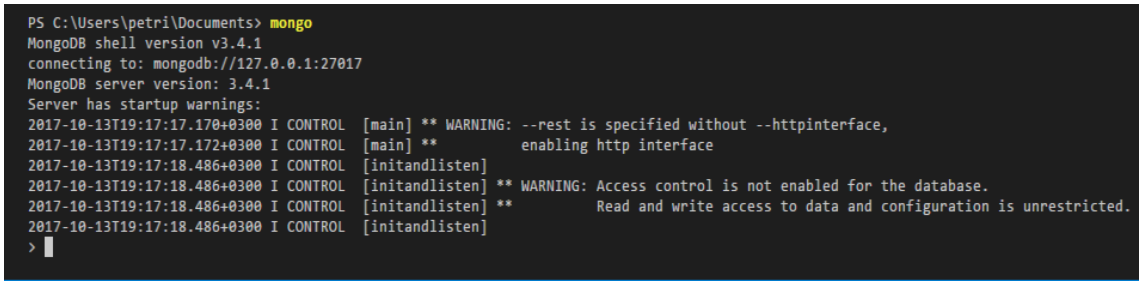


Kuva 4. Komentokehotteessa saadaan Node.js-versionumero.

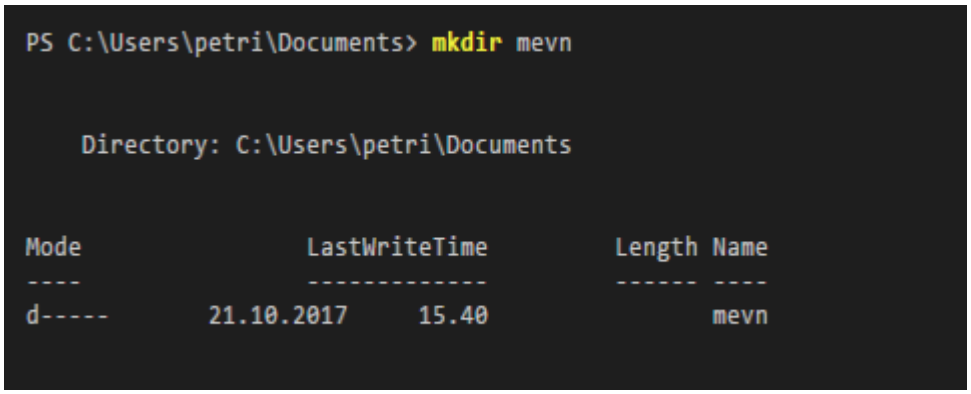
Node.js tuo myös mukanaan NPM-pakettihallintatyökalun, jolla voidaan jakaa, asentaa ja päivittää omia tai muiden kehittäjien rakentamia koodikirjastoja tai -paketteja. Verkkoohjelmoinnissa näitä kirjastoja saattaa olla kymmenittäin, ellei jopa satoja pieniä paketteja. Paketit ovat yleensä pieniä siksi, että niiden on tarkoitus tuoda vain yhteen ongelmaan ratkaisu. Tällä tavoin on helpompaa pitää yksittäisiä paketteja päivitettyinä ja tarkkailla yhteensopivuuksia eri pakettien välillä esimerkiksi omaan projektiin nähden. Pakettien paikalliseen hallintaan käytetään package.json-tiedostoa. Se tarjoaa kehittäjälle dokumentaation projektin riippuvuuksista, eli lista mitä paketteja projekti tarvitsee ja mitä versiota voidaan käyttää.

Seuraavaksi projektia varten tarvittiin MongoDB-tietokanta, ja asennuspaketti löytyy sen kotisivuilta. MongoDB vaatii asennuksessa kansion, johon tallennetaan kaikki tietokannan tiedostot. Oletuksena kansiohakemisto Windowsissa on \data\db. Jotta MongoDB:n käyttö olisi mahdollisimman vaivatonta komentokehotteessa, asetetaan Windows-ympäristömuuttujiin MongoDB:n asennuskansio. Tämän jälkeen komentokehotteeseen voidaan kirjoittaa `mongo` ja saadaan näkyviin muun muassa versionumeroinnit ja tietokannan verkko-osoite. [Kuva 5.]

Kuva 5. Komentokehotteessa suoritetaan mongo-komento.



Seuraavaksi projektille luotiin kansio, johon luodaan projektin ohjelmakoodit ja NPM-pakettihallinnan kautta ladattuja paketteja. Komentokehotteeseen kirjoitettiin `mkdir mevn` uuden kansion tekemiseksi [Kuva 6]. Projekti jaettiin kolmeen alahakemistoon: api-server, vue-app ja cordova-vueapp.



Kuva 6. Uuden kansion luonti komentokehotteessa.

Api-server-kansioon luotiin kaikki palvelimella suoritettavat koodit. Vue-app-kansioon luotiin sovelluksen käyttöliittymä. Cordova-vueapp-kansioon ladattiin Cordova npm-pakettihallinnan kautta sekä valmis Vue-sovellus.

4.2 Palvelinsovellus

Palvelinsovelluksen rakentaminen lähti käyntiin komennolla komentokehotteessa `npm init` api-server-kansiossa. Komennolla luotiin package.json-tiedostoa annettiin projektille nimi, versionumerointi, kuvaus, lisenssin tyyppi ja luoja. [kuva 7.]

```
PS C:\Users\petri\Documents\mevn\api-server> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (api-server)
version: (1.0.0)
description:
git repository:
keywords:
license: (ISC)
About to write to C:\Users\petri\Documents\mevn\api-server\package.json:

{
  "name": "api-server",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "test"
  },
  "author": "Petri Väänänen",
  "license": "ISC",
  "description": ""
}


Is this ok? (yes)
PS C:\Users\petri\Documents\mevn\api-server>
```

Kuva 7. Komentokehotteessa suoritetaan npm init.

Package.json:n luonnissa tallentuivat perustiedot projektista, kuten projektin nimi ja lisenssi sekä sen luoja. Seuraavaksi pakettihallinnasta asennettiin komennolla paketteja npm install ja paketin nimi, kuten esimerkiksi `npm install express –save`. Parametri `–save` tallentaa uuden riippuvuuden projektiin, paketin package.json-tiedostoon riippuvuuslistaan nimeltä express. [Kuva 8.]

```
PS C:\Users\petri\Documents\mevn\api-server> npm install express --save
api-server@1.0.0 C:\Users\petri\Documents\mevn\api-server
`-- express@4.16.2
  +-- accepts@1.3.4
```

Kuva 8. NPM-pakettihallinnan avulla asennetaan express.js.

Parametri `--dev` lisää riippuvuuden Node.js-projektin, joka on voimassa vain kehitysympäristössä eikä vaikuta tuotantoon vietäville koodeille. Jos projekti aloitetaan jo valmiin package.json -tiedoston avulla, riittänee riippuvuuksien asentamiseen pelkästään `npm install` -komento. Tähän projektiin tarvittiin kuusi tuotantoon tarvittavaa pakettia ja viisi pakettia kehitysympäristöön [kuva 9].

```
{
  "name": "api-server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon -w src --exec \"babel-node src --presets es2015,stage-0\"",
    "build": "babel src -s -D -d dist --presets es2015,stage-0",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^1.0.3",
    "body-parser": "^1.17.2",
    "cors": "^2.8.3",
    "express": "^4.15.3",
    "gridfs-stream": "^1.1.1",
    "jsonwebtoken": "^7.4.1",
    "mongoose": "^4.11.1",
    "multer": "^1.3.0"
  },
  "devDependencies": {
    "babel-cli": "^6.24.1",
    "babel-core": "^6.9.0",
    "babel-preset-es2015": "^6.9.0",
    "babel-preset-stage-0": "^6.5.0",
    "nodemon": "^1.11.0"
  }
}
```

Kuva 9. Projektin package.json-tiedosto.

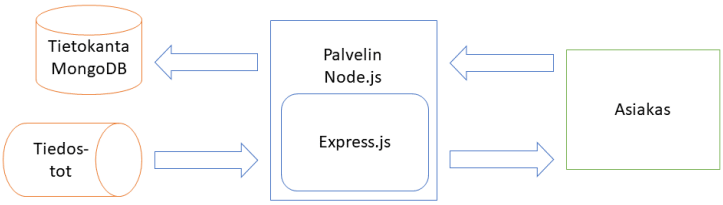
Tuotannon riippuvuuksiin kuuluvat bcrypt, body-parser, cors, express, gridfs-stream, jsonwebtoken, mongoose sekä multer. Bcryptillä salataan salasanat. Body-parser käsittelee http-pyyntöjä purkamalla ja jakamalla post-tiedon / tiedostot req-objekista req.body-objektiin. Cors-paketin nimi tulee sanoista Cross-Origin Resource Sharing. Esimerkiksi API:t ja XMLHttpRequest-pyynnöt, jotka tulevat eri domainista, protokollasta tai verkkoportista, kuin pyydettävä tieto sijaitsee, merkitään cross-origin-http-pyynnöksi. Gridfs-stream antaa valmiin työkalun liitetiedostojen tallentamiseen MongoDB-tietokantaan sekä työkalun liitetiedoston lataamista varten MongoDB-tietokannasta. Jsonwebtokenilla luodaan väliaikaisia avaimia, jotka jaetaan käyttäjälle tämän tunnistamiseksi palvelimella. Tämän ansiosta käyttäjätunnuksia ja salasanoja ei tarvitse siirtää jokaisessa http-kyselyssä. Mongoose-paketti antaa mainion työkalun MongoDB-tietokannan kanssa kommunikointiin. Sen avulla voidaan suoraan lisätä, päivittää tai poistaa dokumentteja.

Kehitysympäristössä on käytössä babel-paketteja, joilla saadaan Javascriptin ECMAScript 6-standardit käännettyä ECMAScript 5-standardiin. Vaikka ECMAScript 6-version spesifikaatio on ollutkin jo tovin standardoitu Javascript, harva selain tukee kuitenkaan sitä suoraan, Google Chromea lukuun ottamatta. Nodemon-työkalu valvoo jokaista muutosta tiedostoissa, ja tiedoston muuttuessa nodemon uudelleenkäynnistää Node.js-palvelimen.

4.2.1 Palvelimen arkkitehtuuri

Projektissa palvelimen pääasiallinen toiminta on tarjota RESTful-rajapinta ja verkkosovellus [Kuva 10]. REST-rajapinta ja verkkosovellus keskustelevat keskenään http:n avulla käyttäen json-muotoisia tiedostoja. REST (Representational State Transfer) on arkkitehtuurimalli, joka pohjautuu http-protokollan ominaisuuksiin ja jonka avulla voidaan suorittaa kokonaisuuksien luominen, lukeminen, päivittäminen ja poistaminen. Yleisesti käytettyjä http-menetelmiä ovat GET, PUT, POST, DELETE ja OPTIONS. [12.]

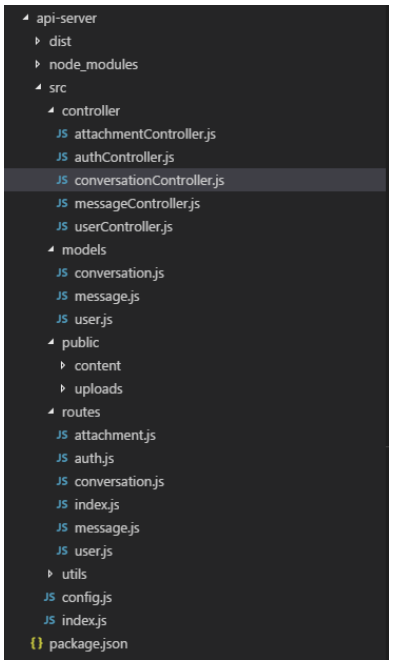
- GET-menetelmä palauttaa kokonaisuuksia lukuoikeudella.
- PUT-menetelmää käytetään kokonaisuuden päivittämiseen.
- POST-menetelmällä lähetetään uusi kokonaisuus luomiseksi.
- DELETE-menetelmällä voidaan poistaa kokonaisuus.
- OPTIONS-menetelmä palauttaa listan tuetuista toiminnoista verkkopalvelussa edellä mainituista menetelmistä.



Kuva 10. Sovelluksen arkkitehtuuri.

Tiedostorakenteessa on noudatettu MVC-suunnittelumallia, eli malli, näkymä ja käsittelijä (model, view ja control). Models-kansiossa on kaikki MongoDB-tietokannan mallit medialiitteistä, keskusteluista, viesteistä, käyttäjien rekisteröintikutsuista ja käyttäjistä.

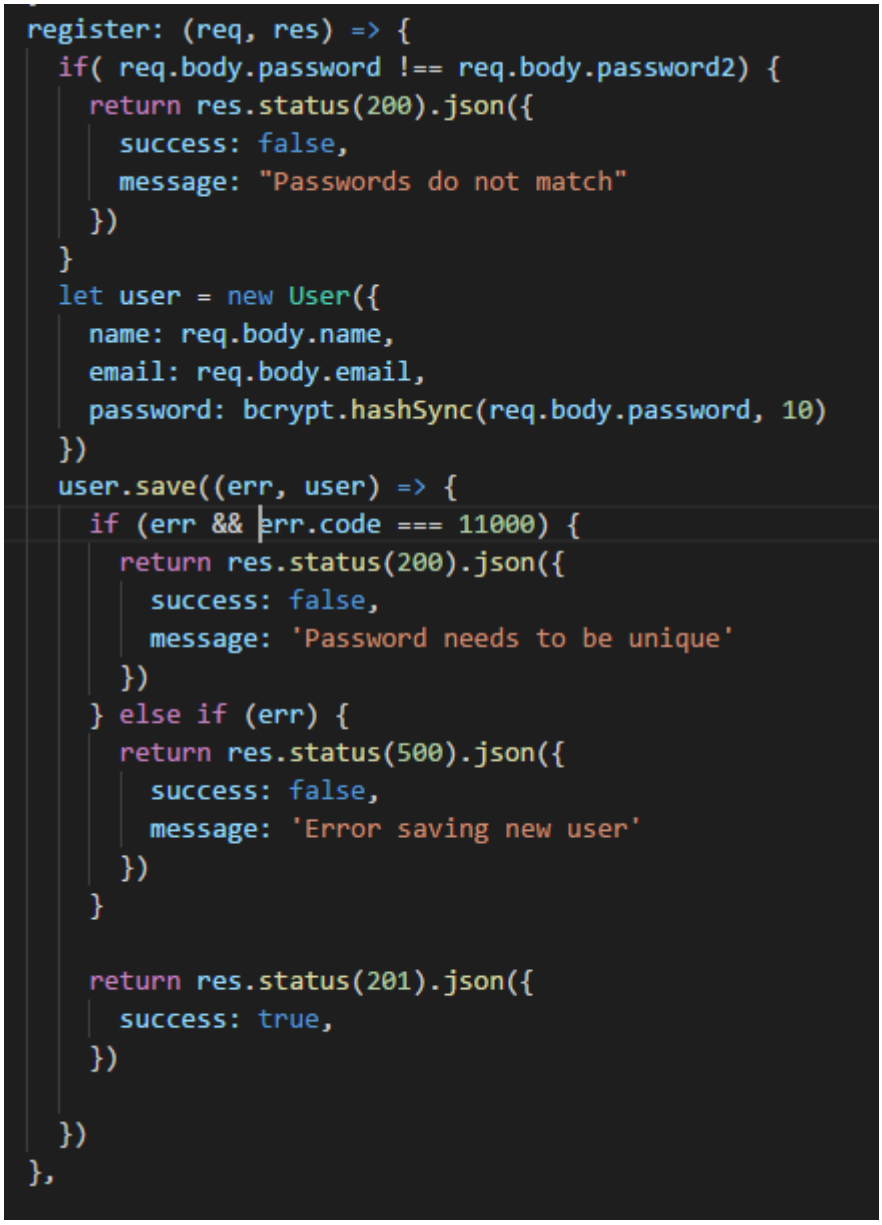
Näkymänä toimi public-kansio, johon tuodaan Vuella luodut käyttöliittymän ohjelmakoodit ja mediatiedostot. Käsittelijän koodit sijaitsevat controllers- sekä routes-kansiossa. Routes-kansio sisältää reitityksiä, joiden avulla ohjataan asiakkaan kysely oikeaan käsittelijään, minkä jälkeen palautetaan arvot asiakkaalle käsittelijän palauttamalla arvolla. Controllers-kansio sisältää käsittelijät, joilla luodaan tietokantakyselyt mallien avulla. [Kuva 11.]



Kuva 11. Sovelluksen kansiorakenne.

4.2.2 Käyttäjän lisääminen

Käyttäjän rekisteröinti suoritettiin lähettämällä post-pyyntö palvelimelle osoitteeseen /api/authentication/signup. Kyselyllä lähetettiin json-tiedosto, joka sisälsi käyttäjänimen, sähköpostiosoitteen ja salasanan.



```
register: (req, res) => {
  if( req.body.password !== req.body.password2) {
    return res.status(200).json({
      success: false,
      message: "Passwords do not match"
    })
  }
  let user = new User({
    name: req.body.name,
    email: req.body.email,
    password: bcrypt.hashSync(req.body.password, 10)
  })
  user.save((err, user) => {
    if (err && err.code === 11000) {
      return res.status(200).json({
        success: false,
        message: 'Password needs to be unique'
      })
    } else if (err) {
      return res.status(500).json({
        success: false,
        message: 'Error saving new user'
      })
    }

    return res.status(201).json({
      success: true,
    })

  })
},
```

Kuva 12. Ohjelmakoodia tiedostosta authController.js.

Palvelimen vastaanottaessa http post -pyynnön sovellus tarkistaa ensin salasanojen olevan yhteneväiset, minkä jälkeen käyttäjä luodaan lähetetyillä tiedoilla kryptaten salasana. Virhetilanteessa palvelin lähetti tapahtuneesta virheestä ilmoituksen. [Kuva 12.]

4.2.3 Sisäänkirjautuminen järjestelmään

Sovelluksen tarjoamaa rajapinnan käyttöä rajoitettiin vain pääsyoikeusavaimen haltijoille, eli käyttäjän piti aina lähettää tunnistautumiseen pääsyoikeusavaimensa käyttäessään rajapintaa. Kirjautuminen tapahtui lähettämällä http post -kysely osoitteeseen /api/authentication/signin mukanaan json-tiedosto. Tiedosto sisälsi sähköpostiosoitteen

ja salasanan. Palvelimen vastaanottaessa post-pyynnön tietokannasta haettiin ja verrattiin vastaanotettu sähköposti ja salasana toisiinsa. [Kuva 13.]

```
login: (req, res) => {
  User.findOne({
    email: req.body.email
  },(err, user) => {
    if(err){
      return res.status(500).json({
        success: false,
        message: "Authentication failed."
      })
    }
    if(!user){
      return res.status(401).json({
        success: false,
        message: 'Authentication failed. User or Password is wrong.'
      })
    }

    bcrypt.compare(req.body.password, user.password, (err, isvalid) => {
      if(!isvalid){
        return res.status(401).json({
          success: false,
          message: 'Authentication failed. Email or Password is wrong.'
        })
      }
      const payload = {
        admin: user.admin
      }
      let token = jwt.sign(payload, config.secret, {
        expiresIn: '24h' // 24 hours
      })
      return res.json({
        success: true,
        items: {
          expiresIn: 1,
          token: token,
          user: {
            _id: user._id,
            email: user.email,
            name: user.name,
            conversations: user.conversations
          }
        }
      })
    })
  })
}
```

Kuva 13. Ohjelmakoodia tiedostosta authController.js.

Kun käyttäjä löytyi ja salasana täsmäsi, luotiin 24 tunnin pituinen pääsyoikeusavain. Tämän jälkeen asiakkaalle palautettiin käyttäjätiedot. Muussa tapauksessa lähetettiin virheilmoitus.

4.2.4 Keskustelun luominen

Keskustelu voitiin luoda lähettämällä rajapinnalle post-kysely osoitteeseen /api/conversations. Kysely sisälsi json-tiedoston, johon sisällytettiin keskustelun nimi, kuvaus ja keskusteluun halutut käyttäjät. Palvelimen vastaanottaessa järjestelmä haki tietokannasta ensin kaikki käyttäjät varmistaen, ettei uuteen keskusteluun tuotu tietokannan ulkopuolelta käyttäjiä [Kuva 14]. Seuraavaksi luotiin uusi keskustelu tietokantaan http-pyynnön vastaanotetuilla arvoilla lisäten jokaisen käyttäjän objektin avain keskustelun metatietoihin sekä keskustelun objektin avain käyttäjän metatietoihin.

```
User.find( { _id: { $in: users } }, (err, docs) => {
  if( err ){
    return res.status(500).json({
      success: false,
      message: "There is user that isn't exist at all"
    })
  }

  // Create new conversation
  let conversation = new Conversation({
      name: req.body.name,
      description: req.body.description,
      users: docs
  })

  // Save conversation to database with users
  conversation.save( (err, conversation) => {
    if( err ){
      return res.status(500).json({
        success: false,
        message: 'Error creating new conversation.'
      })
    }

    users.forEach(function(element) {
      User.findById( {_id: element}, (err, user) => {
        if( err ){
          return res.status(500).json({
            success: false,
            message: 'Error to finding users.'
          })
        }

        user.conversations.push(conversation)
        user.save( (err) => {
          if( err ){
            return res.status(500).json({
              success: false,
              message: 'Error to saving conversation to user.'
            })
          }
        })
      })
    });
    return res.status(200).json({
```

Kuva 14. Ohjelmakoodia tiedostosta conversationController.js.

4.2.5 Keskustelujen listaaminen

Kaikki keskustelut saatiin rajapinnasta get-pyynnöllä osoitteesta /api/conversations. Rajapinta palauttaa json-tiedoston, joka sisälsi listan keskusteluita. Keskustelut haettiin tietokannasta keskustelu-id:llä. Keskustelu-objektiin lisättiin tietokantahaun aikana keskusteluun lisätyt käyttäjät. Mikäli keskusteluita ei löytynyt tai tietokantaan ei saatu yhteyttä, rajapinta palautti virheilmoituksen [Kuva 15].

```
list: (req, res) => {
  const arr = req.params.ids.split(",")
  Conversation.
    find( {_id: {
      $in: arr.map(function(o){ return mongoose.Types.ObjectId(o); })
    }}).
    populate({path:'users', select: 'name' }).
    exec( (err, conversations ) => {
    if(err) {
      return res.status(500).json({
        success: false,
        message: 'Error getting any conversation.'
      })
    }

    if (conversations.length === 0) {
      return res.status(200).json({
        success: false,
        message: 'No conversations.'
      })
    }

    return res.json({
      success: true,
      items: conversations
    })
  })
},
```

Kuva 15. Ohjelmakoodia tiedostosta conversationController.js.

4.2.6 Keskustelun viestit

Keskusteluihin kuuluvat myös viestit. Viestit saatiin rajapinnan avulla pyytämällä get-kyselyn /api/message/:id, jossa id vastaa keskustelun id-arvoa. Viestit haettiin keskustelun id:n avulla tietokannasta. Viesteihin liitettiin tietokantahaun aikana viestin käyttäjän id ja nimi sekä mahdollinen kuva, jos sellainen oli liitetty viestiin mukaan. Virhetilanteessa rajapinta palautti virheilmoituksen. [Kuva 16.]

```
show: (req, res) => {
  let id = req.params.id

  Message.find({conversation_id: id}).
    sort({created: 1}).
    limit(100).
    populate({path:'user_id', select: 'name' }).
    exec( (err, messages ) => {
    if( err ) {
      return res.status(500).json({
        success: false,
        message: 'Error getting a messages.'
      })
    }
    if( !messages ){
      return res.status(200).json({
        success: false,
        message: 'No messages found'
      })
    }

    return res.json({
      success: true,
      items: messages
    })
  })
},
```

Kuva 16. Ohjelmakoodia tiedostosta messageController.js.

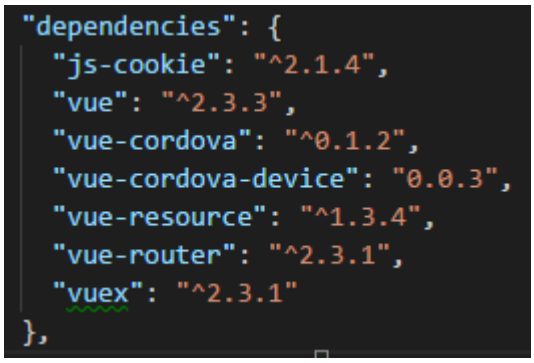
4.3 Käyttöliittymäsovellus

Perinteisen verkkosovelluksen käyttöliittymän asennukseen käytettiin webpack-asennuspakettia. Vue asennettiin ensin NPM-pakettihallinnalla, kun kirjoitettiin komentokehotteeseen `npm install vue`. Seuraavaksi asennettiin Vuen komentorivirajapinta vue-cli komennolla `npm install -global vue-cli`, josta tehtiin globaali asennus. Vuen webpack asennus suoritettiin komennolla `vue init webpack [kansion nimi]`. Tämän jälkeen seurattiin vue-asennuksen ohjeita. [Kuva 17.]

```
PS C:\Users\petri\Documents\mevn> vue init webpack vue-app

? Project name vue-app
? Project description A Vue.js project
? Author Petri Väänänen [redacted]
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? No
? Setup unit tests with Karma + Mocha? No
? Setup e2e tests with Nightwatch? No

   vue-cli · Generated "vue-app".

   To get started:

     cd vue-app
     npm install
     npm run dev

   Documentation can be found at https://vuejs-templates.github.io/webpack
```

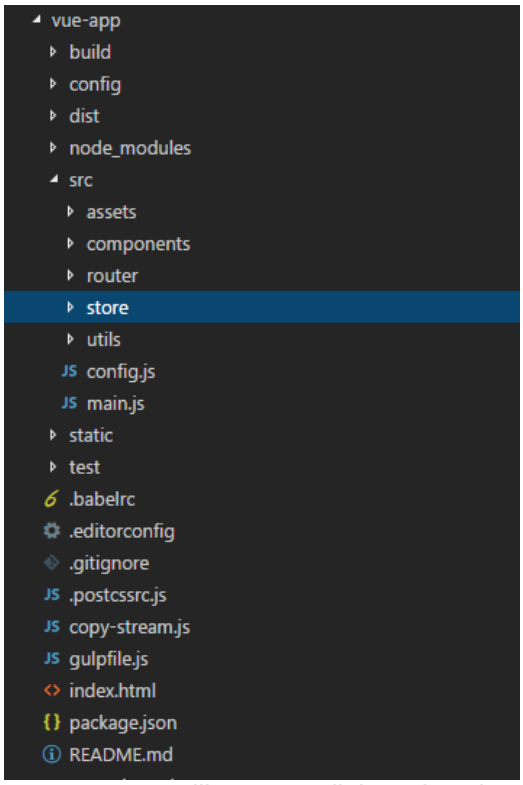
Kuva 17. Vue.js webpack -asennus.

Webpack asensi paljon erilaisia riippuvuuksia [Kuva 18]. Js-cookie antaa helpot työkalut hallita selaimen evästeitä. Vue-paketti sisältää Vue-sovelluskehyksen ydintoiminnot. Vue-cordova tuo työkalut Cordovan ja Vuen kommunikointiin. Vue-cordova-device tuo mukanaan rajapinnan Cordovan laitteistolisäosille. Vue-resource käsittelee verkkokyselyitä joko käyttämällä XMLHttpRequestia tai JSONP:a. Vue-router hallinnoi asiakkaan päässä tapahtuvia sivunvaihtoja yksisivuisussa verkkosovelluksissa. Se ohjaa URL:n perusteella oikeaan Vuen komponentteihin, jotka piirretään verkkosivulle. Vuex tuo asiakkaan sovellukseen oman pienen tietokannan tai väliaikaisen tallennuksen. Sen avulla voidaan myös tehdä API-kyselyitä palvelimelle ja palautettu arvo voidaan säilöä Vuexin tarjoamaan tallennustilaan. Tallennukset kuitenkin häviävät, jos käyttäjä pakottaa verkkosivun päivittämisen.



Kuva 18. Vue.js-sovelluksen package.json-tiedosto.

Vuen kansiorakenne on pilkottu pieniin osiin [Kuva 19]. Build-kansioon luotiin valmis käyttöliittymä paketti. Config- ja dist kansio luotiin Vuen asennuksen yhteydessä. Ne sisältävät webpack-asetuksia, joilla rakennettiin käyttöliittymäpaketti. Myös Static-kansio luotiin Vuen asennuksen yhteydessä. Tähän kansioon luotiin staattiset tiedostot, kuten valmiit tyylitiedostot ja kuvat. Src-kansioon rakennettiin itse käyttöliittymä. Sen alle luotiin assets-, compontents-, router-, store- ja utils-kansiot. Assets-kansioon luotiin tyylitiedostot. Components-kansioon luotiin käyttöliittymäkomponenttipalasia, joita yhdistämällä saatiin tehtyä kokonainen sivu.

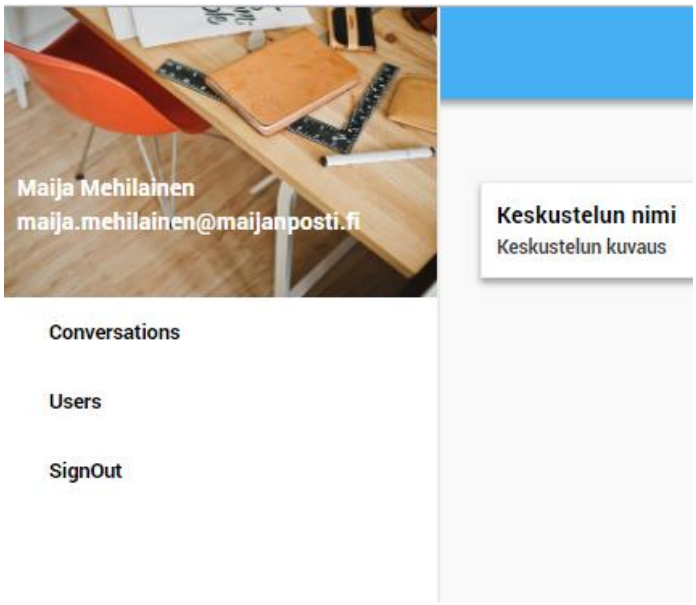


Kuva 19. Käyttöliittymäsovelluksen kansiorakenne.

Komponentit jaoteltiin pieniin osiin komponenttien osien uudelleenkäytön takia. Router-kansioon luotiin ohjaukset yksisivuisen sovelluksen sisäisiin ohjauksiin. Store-kansioon luotiin ohjelmakoodit, jotka tapahtuvat taustalla ja jakavat funktioita koko projektin käyttöön. Sen avulla kaikki API-toiminnot saatiin erotettua käyttöliittymäkomponenteista pois, ja sen funktioita pystyttiin uudelleenkäyttämään eri käyttöliittymäkomponenteissa. Utils-kansioon luotiin avustavia muuttujia ja funktioita, joita voitiin uudelleen käyttää useamman kerran.

4.3.1 Käyttöliittymän sivupalkki

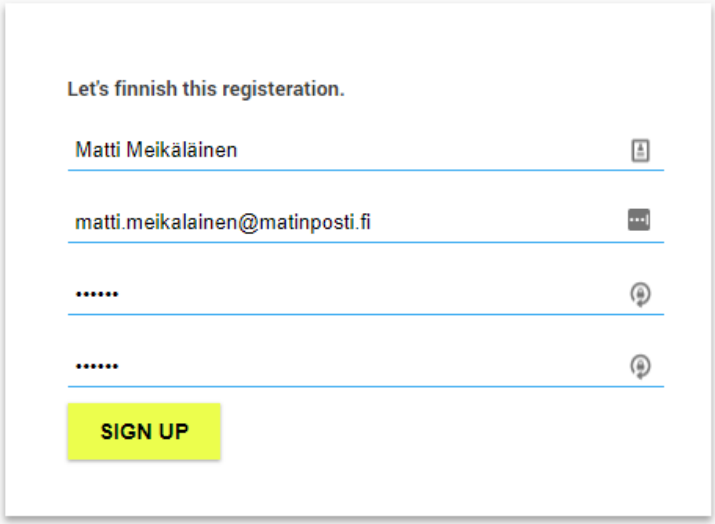
Chat-sovelluksen käyttöliittymän sivupalkissa on mobiililaitteista tuttu sivupalkki, joka toimii navigaationa sivustolla. Raahaamalla sivupalkkia vasemmalle voitiin elementti poistaa näkyvistä. Sivupalkki saadaan näkyviin painamalla hampurilaisvalikkopainiketta. Sivupalkkiin kuuluu pieni kuva ja kirjautuneen käyttäjän nimi ja sähköpostiosoite. Sen alla on listattu keskustelut, käyttäjät ja kirjaudu ulos -painikkeet. [Kuva 20.] Mikäli käyttäjä ei ole kirjautunut, näkyy sivupalkissa vain rekisteröityminen ja kirjaudu sisään -painikkeet.



Kuva 20. Käyttöliittymän sivupalkki.

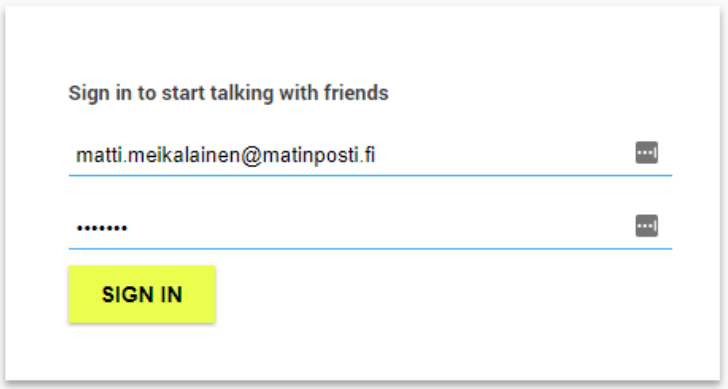
4.3.2 Rekisteröityminen ja sisäänkirjautuminen järjestelmään

Uuden käyttäjän rekisteröinti voitiin tehdä klikkaamalla sivupalkista Signup tai lisäämällä selaimen osoiteriville /signup domain-osoitteen perään [kuva 21]. Rekisteröintilomakkeeseen täytettiin käyttäjän koko nimi, sähköpostiosoite ja salasana sekä salasanan vahvistus.



Kuva 218. Käyttöliittymän rekisteröintinäkymä.

Painamalla Sign up -nappia sovellus lähettää API-kyselyn palvelimelle. Kun käyttäjätili on onnistuneesti lisätty järjestelmään, palautetaan käyttöliittymälle hyväksytty arvo, minkä jälkeen käyttöliittymä ohjaa käyttäjän signin-sivulle. Sisäänkirjautumisessa pyydetään käyttäjän sähköpostiosoitetta ja salasanaa. [Kuva 22.]

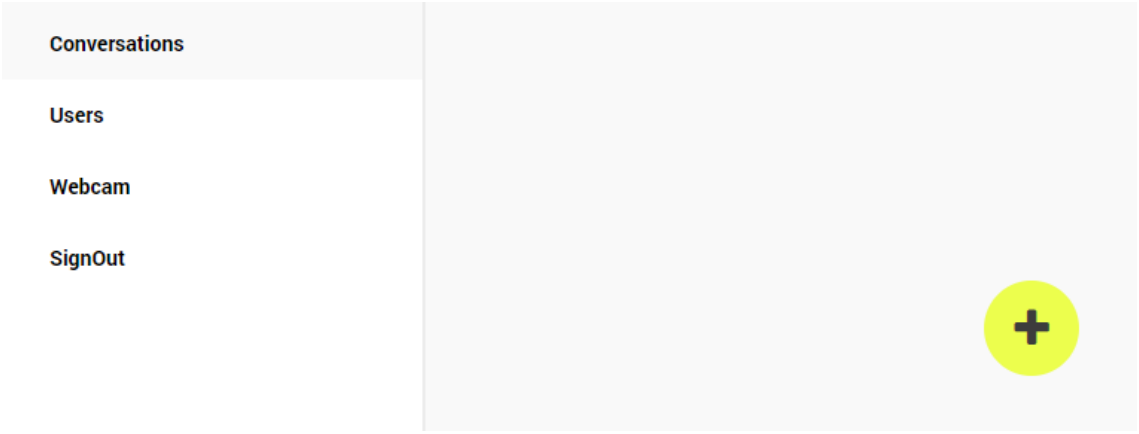


Kuva 22. Käyttöliittymän sisäänkirjautumisen näkymä.

Painamalla Sign In -nappulaa tarkistetaan käyttöliittymäsovelluksessa ensin, että kentillä on arvot. Tämän jälkeen sovellus lähettää palvelimelle API-kyselyn. Kun kirjautuminen on onnistunut, palvelin palauttaa käyttöliittymäsovellukselle hyväksytty-tiedon ja käyttäjälle pääsyoikeusavaimen. Pääsyoikeusavain tallennetaan sovellukseen ja selaimen evästeisiin. Sovellus lukee avaimen pääasiassa sovelluksen muistista. Mikäli arvoa ei löydy sovelluksen muistista, ladataan arvo selaimen evästeistä.

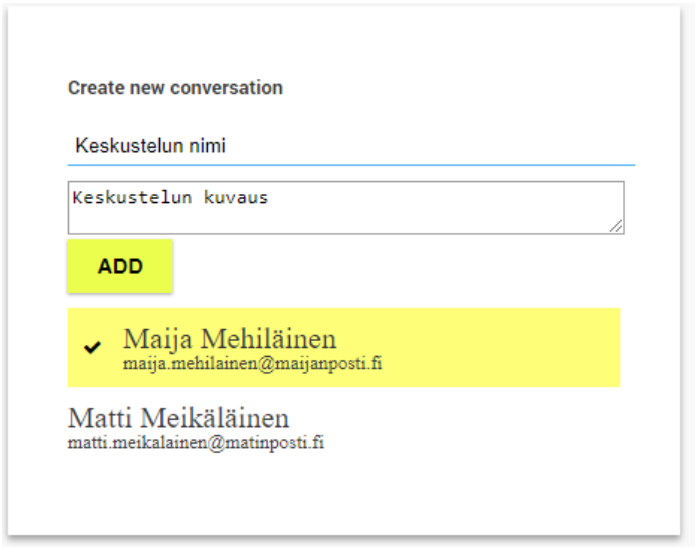
4.3.3 Keskustelut

Uusi keskustelu voidaan luoda käyttöliittymäsovelluksessa menemällä osoitteeseen /conversation/new tai klikkaamalla sivupalkin navigaatiosta Conversations ja oikeassa alakulmassa olevaa isoa plus -merkillä varustettua nappia. [Kuva 23.] Sovellus uudelleenohjaa /conversation/new -osoitteeseen.



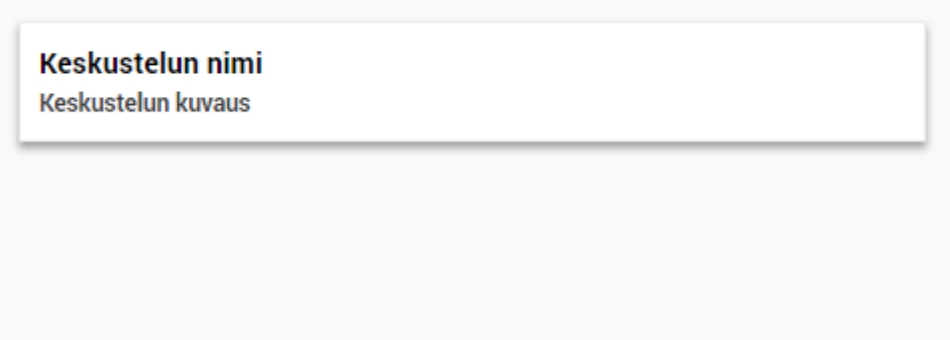
Kuva 9. Käyttöliittymän keskustelutnäkymä tyhjänä ja iso plus -merkillä varustettu nappi.

Keskustelun lisäämisessä vaaditaan keskustelun nimi. Keskustelun kuvaus on vapaaehtoinen. Näiden lisäksi keskusteluun on lisättävä käyttäjät, jotka halutaan keskusteluun mukaan, valitsemalla käyttäjän nimi. Sovellus näyttää valitut käyttäjät keltaisella taustavärillä. [Kuva 24.] Painamalla Add-nappia lähetetään palvelimelle annetut tiedot.



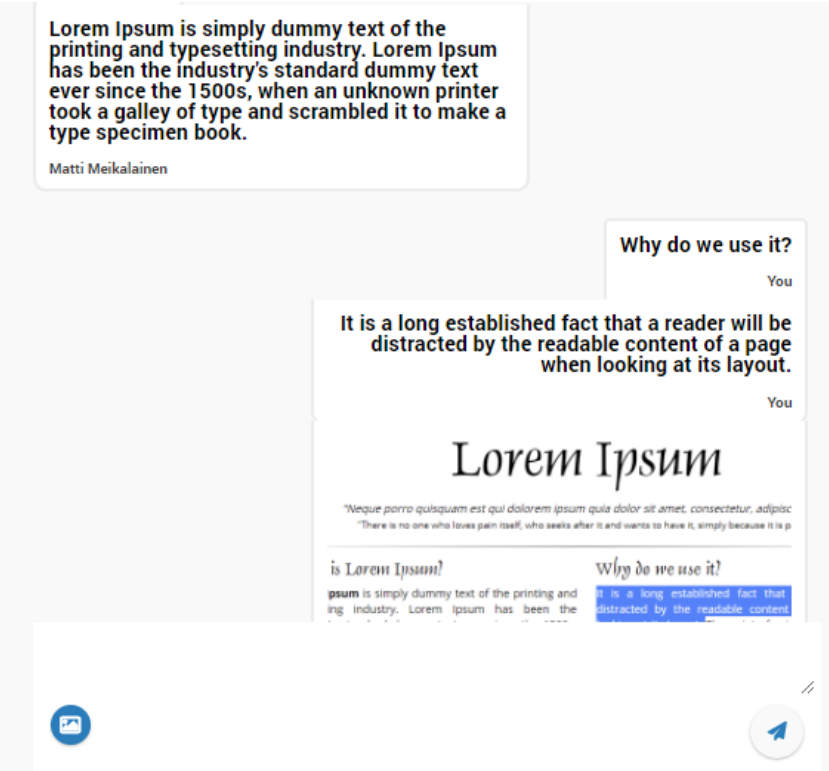
Kuva 24. Keskustelun lisääminen.

Onnistuneen keskustelun lisäyksen jälkeen käyttäjä ohjataan käyttöliittymäsovelluksessa keskustelutnäkymään. Samanaikaisesti sovellus päivittää keskustelu listauksen annetuille käyttäjille. Keskusteluiden listausnäkymässä näytetään keskustelun nimi ja kuvaus. Keskustelua painettaessa päästään keskustelun sisälle. Yksittäiseen keskusteluun pääsee myös kirjoittamalla selaimen osoitekentään /conversation/:keskustelun yksilöivä avain. [Kuva 25.]



Kuva 25. Keskustelujen näkymä keskustelun kanssa.

Keskustelussa käyttäjä voi lähettää tekstiä tai kuvia tietokoneella tai mobiililaitteella [kuva 26]. Mobiililaitteella voidaan myös ottaa kameralla kuva ja lähettää kuva oton jälkeen. Käyttäjän itse lähettämä kuva näytetään ruudun oikeassa reunassa ja muiden lähettämät viestit vasemmalla puolella. Keskustelunäkymän alalaidassa sijaitsee tekstikenttä ja sen vasemmassa alalaidasta löytyy kuvahakutoiminnallisuus. Mobiililaitteissa sen viereen tuodaan näkyviin kameralla varustettu nappi, vain jos sovellus osaa tunnistaa mobiililaitteen kameran. Oikeassa reunasta löytyy viestin lähetyspainike paperilentokonekuvakkeena. Viestit päivitetään sekunnin välein käyttäjille.



Kuva 106. Viestit keskustelussa, joka sisältää myös yhden kuvan.

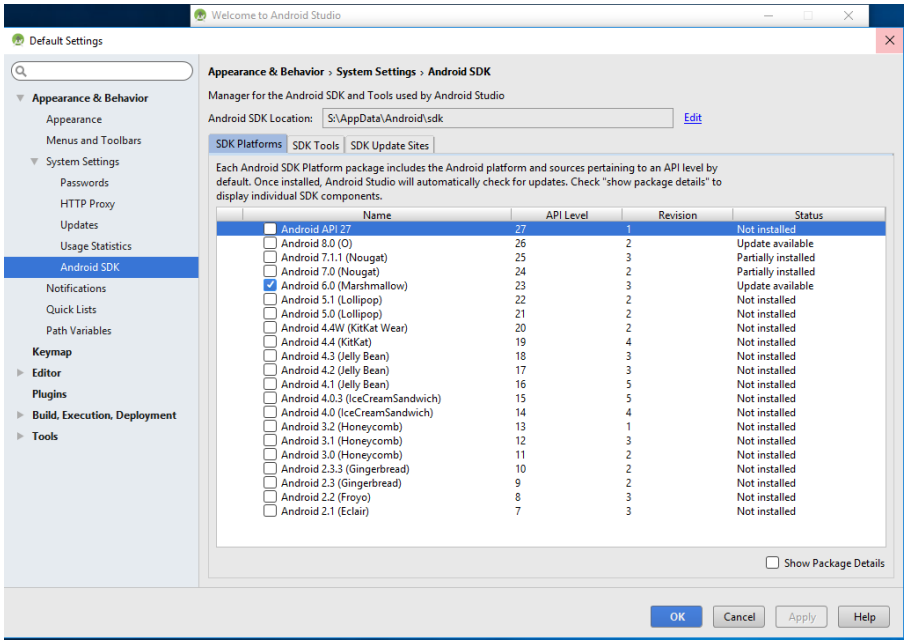
4.4 Käyttöliittymäsovellus mobiililaitteisiin

Käyttöliittymäsovellus asennettiin mobiililaitteeseen Cordovan kehitystyökalun avulla. Cordovan komentorivirajapinta asennettiin globaaliksi NPM-pakettihallinnan avulla `npm install cordova -g` -komennolla. Cordova-komentorivirajapinnan avulla pystyttiin luomaan uusi Cordova-projekti komennolla `cordova create [asennus kansio]`. Tämän jälkeen siirryttiin cordova-app-kansioon, jossa lisättiin Android-alusta, jonne haluttiin asentaa käyttöliittymäsovellus `komennolla cordova platform add android`. [Kuva 27.]

```
PS C:\Users\petri\Documents\mevn> cordova create cordova-app
Creating a new cordova project.
PS C:\Users\petri\Documents\mevn> cd .\cordova-app\
PS C:\Users\petri\Documents\mevn\cordova-app> cordova platform add android
Using cordova-fetch for cordova-android@~6.2.2
Adding android project...
Creating Cordova project for the Android platform:
        Path: platforms\android
        Package: io.cordova.hellocordova
        Name: HelloCordova
        Activity: MainActivity
        Android target: android-25
Subproject Path: CordovaLib
Android project created with cordova-android@6.2.3
Discovered plugin "cordova-plugin-whitelist" in config.xml. Adding it to the project
Installing "cordova-plugin-whitelist" for android

               This plugin is only applicable for versions of cordova-android greater than 4.0. If you have
```

Kuva 27. Cordova webpack -asennus.

Jotta Cordovalla voidaan luoda Android-alustalle sovelluksia, pitää Windows-ympäristöön asentaa Oraclen Java SE JDK (Stantard Edition Java Development Kit) -kehitystyökalut. Tämän jälkeen asennettiin Android SDK, joka antaa tarvittavat kirjastot ja työkalut Android-sovelluksen rakentamiseen. Seuraavaksi luotiin Windows-ympäristömuuttujiin ANDROID_HOME-muuttuja ja asennuspoluksi annettiin android-sdk-asennuskansio. Android SDK:n yhteydessä tulleen Android SDK Managerin avulla tarkastettiin ja asennettiin Android Marshallow 6.0-alustalle API ja Android 6.0:n tarvitsemat SDK-työkalut. [Kuva 28.]



Kuva 28. Android Studio -asetukset.

Tämän jälkeen tarkistettiin, että kaikki oli asennettu oikein ja ympäristömuuttujat toimivat oikein komennolla cordova requirements. Vaatimuksiin lukeutuivat Java JDK-, Android SDK- ja Android API-asennukset sekä Gradle, jonka avulla sovellus käännetään Androidille toimivaksi sovellukseksi. [Kuva 29.]

```
PS C:\Users\petri\Documents\mevn\cordova-app> cordova requirements

Requirements check results for android:
Java JDK: installed 1.8.0
Android SDK: installed true
Android target: installed android-25,android-24,Google Inc.:Google APIs:24,android-23
Gradle: installed C:\Users\petri\scoop\shims\gradle.cmd
PS C:\Users\petri\Documents\mevn\cordova-app>
```

Kuva 29. Komentokehotteessa Cordovan vaatimukset.

Seuraavaksi Cordovan komentorivirajapinnan avulla asennettiin Cordova-kameralisäosa. Lisäosa asennettiin `cordova plugin add cordova-plugin-camera` komennolla. Lisäosalla voitiin ottaa käyttöön Android-mobiililaitteen kamera käyttöliittymäsovelluksessa. [Kuva 30.]

```
PS C:\Users\petri\Documents\mevn\cordova-app> cordova plugin add cordova-plugin-camera
Installing "cordova-plugin-camera" for android
Installing "cordova-plugin-compat" for android
Subproject Path: CordovaLib
Adding cordova-plugin-camera to package.json
Saved plugin info for "cordova-plugin-camera" to config.xml
```

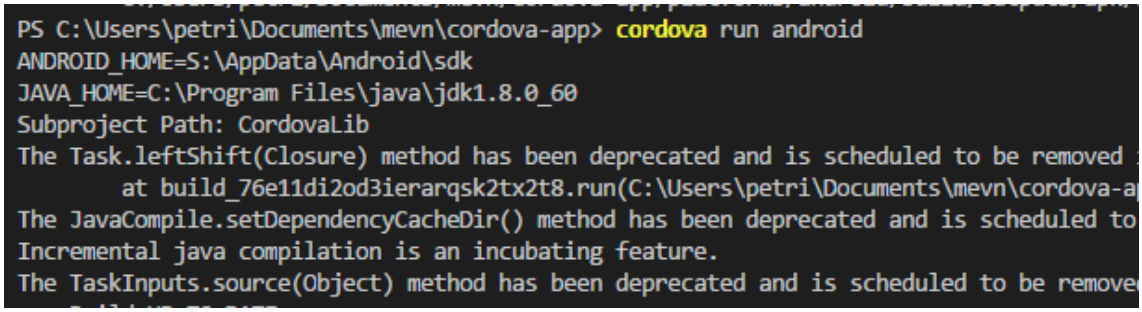
Kuva 30. Komentokehotteessa asennetaan Cordovalle lisäosia.

Seuraavaksi kopioitiin vue-app-kansiosta Vue-sovelluksen luotu valmis tuotantoversio cordova-app-hakemiston www-kansioon. Tämän jälkeen suoritettiin `cordova build android` -komento, joka käänsi koko koodikannan Android-alustalle suoritettavaksi sovellukseksi. [Kuva 31.]

```
PS C:\Users\petri\Documents\mevn\cordova-app> cordova build android
ANDROID_HOME=S:\AppData\Android\sdk
JAVA_HOME=C:\Program Files\java\jdk1.8.0_60
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 8s
1 actionable task: 1 executed
Subproject Path: CordovaLib
Starting a Gradle Daemon (subsequent builds will be faster)
The Task.leftShift(Closure) method has been deprecated and is scheduled to be removed in Gradle 5.0.
        at build_76e11di2od3ierarqsk2tx2t8.run(C:\Users\petri\Documents\mevn\cordova-app\platforms\a
The JavaCompile.setDependencyCacheDir() method has been deprecated and is scheduled to be removed in
Incremental java compilation is an incubating feature.
```

Kuva 31. Komentokehotteessa luodaan Vue.js:llä luotu sovellus Android-sovellukseksi.

Sovelluksen saamiseksi puhelimeen liitettiin puhelin koneeseen kiinni, minkä jälkeen kirjoitettiin komentokehotteeseen `cordova run android`. Komento kopioi tiedostot puhelimeen samalla tavalla, kuin jos käytettäisiin Android SDK -kehitysympäristöä, minkä jälkeen sovellus käynnistyy, mikäli asennus puhelimeen onnistui ja sovellus on virheetön. [Kuva 32.]



Kuva 32. Komentokehotteessa siirretään käännetty sovellus Android-puhelimeen.

4.5 Ohjelmistotestaus

Ohjelmistotestausta voidaan nykyteknologian avulla automatisoida helposti. Käyttöliittymäsovelluksien rakentamisessa Vue.js:n webpack tuo valmiina erilaisia linter-työkaluja, jotka ilmoittavat reaaliaikaisesti virheistä ohjelmistossa, yleensä kirjoitusvirheistä. Myös selaimilla voidaan tutkia ja testata käyttöliittymäsovelluksia. Muun muassa Google Chrome- ja Firefox-selaimille on luotu Vue.js-kehitystyökalulisäosa, jonka avulla voidaan tutkia ja selvittää, mitä Vue.js-komponenteissa tapahtuu missäkin vaiheessa. Tästä oli suuresti hyötyä koko projektin aikana varsinkin Vuex-väliaikaistallennuksia tutkiessa.

Yksikkötestaaminen on myös mahdollista Vue.js -komponenteille ja Vuex-tapahtumahallintamallityökalulle Karma-työkalun avulla. Karma on asennettavissa node.js:n NPM-pakettihallinta-työkalun avulla. Karma-työkalua käytin pelkästään ensimmäisen tietokantayhteyden testaamiseen Vue-sovelluksen avulla. Ensimmäiset kokemukset olivat, että Karmasta jäi valtavat määrät konfiguroitavaa suhteessa siihen, mitä on testaamassa. Karman kokeilut jäivät siksi projektissa sikseen.

Postman

Palvelinsovellusta rakentaessa Postman nopeutti ja helpotti API-rajapinnan kehitystä. Sen graafisella käyttöliittymällä pystyi luomaan ketterästi http-pyyntöjä kuten, get, post, put ja delete. Tämän avulla voitiin testata rajapinnan toimintaa kullakin http-pyynnöllä.

Postmania käytettiin esimerkiksi tilanteissa, joissa haluttiin chat-sovelluksen rajapinnan palauttavan oikeanlaista tietoa. Esimerkiksi http-pyynön get-metodilla osoitteeseen /api/users tulisi palauttaa kaikki käyttäjät. Mikäli rajapinta esimerkiksi vaatii pääsyavaimen käyttöä, rajapinnan tuli palauttaa virheilmoitus json-muotoisena.

```
{
    "success": false,
    "message": "No token provided."
}
```

Http-pyyntöön lisätään headers-alueen avain-arvoparikenttiin x-access-token-avaimen ja arvoksi saatu pääsyoikeusavain. Pääsyoikeusavaimen avulla rajapinta sallii pyynnön suorittamisen ja palauttaa kaikki käyttäjät json-muotoisena. [Kuva 33.]

Kuva 11. Postmanin http-kyselyn vastaus.

Projektissa testejä ajettiin sitä mukaa, kuin sovellukseen tarvitsi rakentaa lisää rajapintakutsuja. Testaukset tehtiin yksittäisinä ajoina käsin. Aluksi testeillä haluttiin varmistua, että rajapinta osaa vastaanottaa annetut tiedot ja käsitellä niitä oikein vastaamalla aina halutulla tavalla. Kun Postmanin testaamisella saatiin haluttu tulos, käytettiin käytettyjä arvoja ja tuloksia vertailuna käyttöliittymän Ajax-pyyntöjen rakentamisessa. Tällä tavalla oli helppoa rajata ongelmakohdat joko käyttöliittymään tai rajapintaan. Yleisimmät virhetilanteet, joita ilmeni testatessa, olivat ohjelmointivirheet, jotka keskeyttivät ohjelmakoodin suorituksen kokonaan, tai osa sellaisista muuttujista puuttui ohjelmakoodista, joiden olisi pitänyt tallentua tietokantaan.

Rajapinnasta testattiin käyttäjän rekisteröiminen ja kirjautuminen ja käyttäjätietojen lukeminen, keskusteluiden listaaminen ja lisääminen, ja viestien lähettäminen ja vastaanottaminen oikeassa keskustelussa. Näiden lisäksi rajapinnan oli estettävä ulkopuolisten pääsy kaikkiin rajapinnan toiminnallisuuksiin pois lukien kirjautuminen ja rekisteröityminen. Rajoitettuja alueita oli päästävä käyttämään vain toimivalla pääsyoikeusavaimella. Kuvien lähettäminen ei onnistunut syystä tai toisesta Postmanilla. Testaaminen toteutettiin silloin curl-komentorivityökalulla `curl -X POST -F 'image=@/path/to/pictures/picture.jpg' http://domain/upload`.

5 Johtopäätökset

Insinöörityön lähtökohtana oli luoda prototyyppiversio chat-sovelluksesta, jolla pystyttäisiin esittämään laitteisto- ja ympäristöriippumattomuus. Insinöörityön tavoitteena oli chat-sovelluksen avulla tutkia, kuinka ketterää on rakentaa sovellus alustariippumattomaksi sekä perinteisille tietokoneille että mobiililaitteille.

Projektia aloittaessani tutustuin erilaisiin MEAN-kirjastokokoelmiin, joista lupaavimmaksi osoittautui MEAN.io-kirjastokokoelma. Kuitenkin hyvin pian selvisi, että ennen valmiin kirjastokokoelman käyttöä tai sen opettelua olisi hyvä opetella ensin jokainen osa-alue MEAN-kokoelmasta, eli MongoDB, Express, AngularJS ja Node.js. Tutkiessani syvemmin tätä maailmaa huomasin, kuinka nopeasti muutama vuosi sitten julkaistu esimerkkisovellus ohjeineen päivineen kävi vanhaksi. Perusperiaatteet pysyivät kuitenkin samoina, mutta kirjoitusasu ohjelmoinnissa saattoi hieman muuttua. Hankalimmaksi MEAN-kokoelmasta koin AngularJS-kirjaston. Tutkiessani vaihtoehtoisia kirjastoja ulkoasun rakentamiseen, kriteerinä oli, että voin tuoda valmiin sovelluksen tavalla tai toisella mobiililaitteseen. Vaihtoehdoiksi AngularJS 1:n rinnalle tuli AngularJS 2 betavaiheessa, React sekä VueJS:n toinen versio. Koska AngularJS 2 oli betavaiheessa, päätin jättää sen pois vaihtoehdoista ja antaa sen valmistua ensin. Reactin valtavasta suosiosta ja kehuista huolimatta jätin sen väliin yksinkertaisesti siitä syystä, että VueJS oli itselleni merkittävästi helpompi sisäistää kuin React. Vue on ominaisuuksiltaan yksinkertaisempi ja suoraviivaisempi kuin React. VueJS:n myötä kirjastokokelman nimitys muuttui MEAN:sta MEVN-nimeksi.

Tutkiessani työkalua käyttöliittymän rakentamiseen tutkin myös työkaluja, joilla saisi valmiin Javascript-sovelluksen mobiililaitteeseen. Yllätykseksi vaihtoehtoja löytyi paljon erilaisilla lähestymistavoilla. Valitsin Cordovan sen takia, että se ei ollut Javascript-kirjastosta tai sovelluskehysestä riippuvainen ja koska Vuelle oli luotu oma lisäosa helpottamaan Cordovan käyttöönottoa. Cordova tuki myös eri alustoja monipuolisemmin, vaikka tässä projektissa tarvitsin tuen vain Android-alustalle.

MEVN-kirjastokokoelman REST API -palvelun tietokantana toimi MongoDB, ja http-palvelua suoritettiin Node.js:llä. Testailun vuoksi asensin MongoDB-tietokannan eri alustoille, kuten Windows 10, Debian 7 VirtualBoxissa ja Raspberry Pi 2 ARM prosessorille Debian 7.11. Raspberry Pille MongoDB-asennuksen versio oli kuitenkin muutamia ver-

sionumeroita jäljessä, joten ongelmien minimoimiseksi jätin Raspberry Pin kanssa ohjelmoinnin sikseen. VirtualBoxiin asennetussa Debian 7:ssä ilmeni ajureihin liittyviä ongelmia, jotka sotkivat jatkuvasti joitain asetuksia jokaisen VirtualBoxin uudelleen käynnistyksen yhteydessä. Tässä kohtaa totesin Windows 10:n olevan sillä hetkellä kaikista helpoin ja vakain alusta prototyyppisovelluksen toteuttamiseksi.

REST API -palvelua kehittäessä ajatuksena oli pitää toiminnallisuudet mahdollisimman suoraviivaisina. Sovellukselle ei annettu muita tehtäviä kuin vastaanottaa vastaanotetut pyynnöt ja tallentaa tiedot tietokantaan ja vastaavasti lukea tietokannasta tiedot ja lähettää asiakkaalle takaisin. Siitä huolimatta callback hell -tilanteita tulee herkästi kirjoittaessa asynkronista Javascript-koodia. [Kuva 14.] Niin kutsuttu callback hell ilmenee, kun kirjoitetaan uudet callback-funktiot asynkronisissa funktioissa toistensa sisään, pyramidimaiseen malliin.

Vastaavia tilanteita tuli myös tässä projektissa muutamiin paikkoihin tallennettaessa tietoja tietokantaan, esimerkiksi uuden keskustelun luominen, johon tallennettiin myös siihen keskusteluun liitettyjä käyttäjiä selvittäen samalla käyttäjien id-arvot. REST API -palvelun kehittäminen Node.js:llä ensimmäistä kertaa vaati paljon tutkimista ja kokeiluja. Ohjelmakoodin rakenne muuttui muutamia kertoja sitä mukaa, kuin ymmärsi enemmän Javascriptin suorittamisesta palvelinmaailmassa. Haastavin osa oli kuvien vastaanottaminen ja tallentaminen sekä lähettäminen takaisinpäin asiakkaalle.

Ensimmäisessä versiossa tallennettiin kuva kansioon ja tietokantaan tallennettiin tiedoston sijainti sekä sen nimi. Tämä tapa osoittautui hieman yllättäen työlääksi, kun käytettiin pelkästään multer.js-nimistä kirjastoa. Oletuksena multer.js loi uniikin nimen tiedostolle, mutta jätti samalla tiedostopäätteen pois nimestä tallentaessaan kuvan kansioon. Tämän jälkeen kuvan tiedot oli tallennettava tietokantaan ja samalla piti tallentaa kuvan linkki viestiin. Seuraavassa ratkaisussa tallennettiin kuvat suoraan tietokantaan hyödyntäen gridfs-stream-kirjastoa multer.js:n rinnalla. Tämä kirjasto loi automaattisesti tietokantaan tarvittavat kokoelmat ja dokumentit kuville. Tämä antoi helpon tavan myös linkittää kuvat viestiin. Myös kuvien osoitteet generoituivat automaattisesti, eikä kuvien nimistä tarvinnut huolehtia.

Käyttöliittymän rakentaminen oli huomattavasti työläämpää kuin palvelimen tarjoama REST-rajapinnan kehittäminen. Käyttöliittymään kuuluu ulkoasun rakentamisen lisäksi taustalla tapahtuvien http-pyyntöjen lähettäminen ja vastaanottaminen palvelimelta sekä

niihin reagoimiset. Sen lisäksi ulkoasun minimivaatimuksiin kuului alustariippumattomuus. Käyttöliittymän tuli toimia tietokoneella ja mobiililaitteilla ja antaa samanlainen käyttökokemus.

Sovelluksen prototyypin rakentaminen MEVN-sovelluskokoelmaa käyttäen onnistui yllättävän hyvin. Käyttöliittymän suurimmat haasteet olivat mobiililaitteiden natiivitoimintojen tuominen mukaan. Niitä ovat esimerkiksi kameran toiminnot ja tiedostojen lukeminen. Ongelmallisin tilanne käyttöliittymäsovelluksen rakentamisessa oli se, mikäli sovelluksessa haluttiin käyttää mobiililaitteiston ominaisuuksia käyttäen Cordovan tarjoamaa rajapintaa, ennen kuin Cordovan rajapinta on valmis käytettäväksi. Rajapinta on valmis, kun mobiilialusta on laukaissut deviceready-tapahtuman, minkä jälkeen Cordovan rajapinta ilmoittaa Javascript-tapahtumakuulijalle oman deviceready-tapahtuman. Tämän jälkeen voidaan kutsua haluttuja rajapintakutsuja. Mikäli rajapintakutsut Cordovalle tehtiin ennenaikaisesti, vastauksena oli Javascript-virheilmoitus, että laitteeseen ei ole kytketty kyseistä moduulia tai rajapinta ei pysty lukemaan haluttua moduulia. Ratkaisu tällaiseen ongelmaan oli alustaa Vue-sovellus vasta siinä vaiheessa, kun Cordovan rajapinta oli laukaissut Javascript-tapahtumakuuntelijalle deviceready-tapahtuman.

Tämäntapaista sovellusta, jossa voidaan lähettää ja vastaanottaa viestejä Vuella, olisi ollut melkein mahdotonta saada valmiiksi inhimillisessä ajassa, mikäli yhtään lisäosaa ei olisi ollut saatavilla. Tärkeimpinä lisäosina pidin Vue Router -kirjastoa, joka auttaa ohjelmoijaa kertomaan sovellukselle, mitä komponentteja halutaan näyttää milläkin sivulla. Vuex lisäsi mahdollisuuden tallentaa helposti tietoja väliaikaisesti, eli http-pyynnöt saatiin minimoitua. Näiden lisäksi Vue Cordova -kirjasto helpotti Cordovan käyttöönottoa Vuen kanssa merkittävästi.

Sovelluksen REST-rajapinta suoriutui odotetulla tavalla Windows-ympäristössä. Rajapinta tarjoaa käyttäjän rekisteröinnin, sisään kirjautumisen, viestiryhmän luomisen sekä viestien lähettämisen ja vastaanottamisen. Myös mediatiedostojen lähetys ja vastaanottaminen viesteissä oli tuettuna.

Sovelluksen käyttöliittymä toimii verkkoselaimessa ja Android-mobiililaitteissa hybridisovelluksena käyttäen WebView-ominaisuutta. WebView antoi mahdollisuuden käyttää mobiililaitteen laitekomponentteja. Cordova loi rajapinnan kehittäjälle mahdollisuuden käyttää Javascriptiä laitteistokomponenttien ja natiiviominaisuuksien kutsumiseen. Corodova antoi myös laajan tuen laitteiden tilojen kuuntelemiseen, kuten milloin laitteen

omat rajapinnat ovat täysin käytettävissä sovelluksessa, kun sovellus on aktiivisessa tilassa, tai milloin sovellus siirretään tausta-ajoon. Alustoissa oli kuitenkin huomattavia eroja tilojen kuuntelijoiden tuessa. Esimerkiksi äänenvoimakkuuden painikkeiden tiloja voitiin kuunnella vain Android- ja BlackBerry-alustalla. IOS- ja Windows-alustoilla tämä ei onnistunut. Mikäli kaikkia laiteominaisuuksia ei tarvita sovelluksessa, on hybridisovelluksen rakentaminen oiva tapa rakentaa mobiilisovellus monelle mobiililaitteelle, esimerkiksi testiversio asiakkaalle.

Lähteet

1. Malathi, Nayak. 2017. Seventy-five percent of internet use in 2017 will be mobile: report. Verkkoaineisto. <https://www.reuters.com/article/us-internet-mobilephone/seventy-five-percent-of-internet-use-in-2017-will-be-mobile-report-idUSKCN12S29L>.
   Luettu 15.10.2017.

2. Doantam, Phan. 2016. Mobile-first Indexing. Verkkoaineisto. <https://webmasters.googleblog.com/2016/11/mobile-first-indexing.html>. Luettu 9.11.2017.

3. What is MongoDB? Verkkoaineisto. MongoDB, Inc. <https://www.mongodb.com/mongodb-architecture>. Luettu 10.2.2017.

4. Replica Set Data Synchronization. Verkkoaineisto. MongoDB, Inc. <https://docs.mongodb.com/manual/core/replica-set-sync>. Luettu 18.12.2017.

5. Replication Mysql. Verkkoaineisto. Oracle Corporation. <https://dev.mysql.com/doc/refman/5.7/en/replication.html>. Luettu 18.12.2017.

6. Read Preferences. Verkkoaineisto. MongoDB, Inc. <https://docs.mongodb.com/manual/core/read-preference/>. Luettu 19.12.2017.

7. Replication. Verkkoaineisto. MongoDB, Inc. <https://docs.mongodb.com/manual/replication/index.html>. Luettu 11.2.2017.

8. Data Model Desing. Verkkoaineisto. MongoDB, Inc. <https://docs.mongodb.com/manual/core/data-model-design/#data-modeling-embedding>. Luettu 6.1.2018.

9. MMAPv1 Storage Engine. Verkkoaineisto. MongoDB, Inc. <https://docs.mongodb.com/manual/core/mmapv1/#power-of-2-allocation>. Luettu 6.1.2018.

10. Comparison with Other Frameworks. Verkkoaineisto. <https://vuejs.org/v2/guide/comparison.html>. Luettu 15.3.2017.

11. What is the Event Loop? Verkkoaineisto. Node.js Foundation. <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick>. Luettu 10.9.2017.

12. Using HTTP Methods for RESTful Services. Verkkoaineisto. RestApiTutorial.com. <http://www.restapitutorial.com/lessons/httpmethods.html>. Luettu 7.1.2018.

13. Hazem, Saleh. 2014. JavaScript Mobile Application Development. Verkkoaineisto. <https://ebookcentral.proquest.com/lib/metropolia-ebooks/reader.action?docID=1822801&query=hybrid%20app>. Luettu 22.1.2018.

14. Node event loop. Verkkoaineisto. <https://qph.ec.quoracdn.net/main-qimg-ff39ad46d7fbc5cde9cb412ca6f57bd9>.

15. Cordova Architecture. Verkkoaineisto. The Apache Software Foundation. <https://cordova.apache.org/docs/en/latest/guide/overview/index.html#architecture>.

16. Vue.js Directives. Verkkoaineisto. <https://012.vuejs.org/guide/directives.html>.

17. Comparison with Other Frameworks. Verkkoaineisto. <https://vuejs.org/v2/guide/comparison.html#AngularJS-Angular-1>.

18. Android Studio. Verkkoaineisto. <https://developer.android.com/studio/index.html>.

19. Microsoft Visual Studio. Verkkoaineisto. <https://developer.microsoft.com/en-us/windows/downloads>.

20. Xcrode. Verkkoaineisto. <https://developer.apple.com/library/content/referencelibrary/GettingStarted/DevelopiOSAppsSwift/>.

21. NoSQL Databases Explained. Verkkoaineisto. MongoDB, Inc. <https://www.mongodb.com/nosql-explained>.

22. Cordova Tools. Verkkoaineisto. The Apache Software Foundation. <https://cordova.apache.org/>.

23. A Mobile Version of the Ubuntu Operating System. Verkkoaineisto. The UBports Team. <https://ubports.com/ubuntu-touch/about-ut>.