SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

# VIRTUAALIAVUSTAJA - SOVELLUS

TEKIJÄ/T: Joonas Hakkarainen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

Tiivistelmä

| | | | |
|---|---|---|---|
| Koulutusala<br>Tekniikan ja liikenteen ala | | | |
| Koulutusohjelma/Tutkinto-ohjelma<br>Tietotekniikan koulutusohjelma | | | |
| Työn tekijä(t)<br>Joonas Hakkarainen | | | |
| Työn nimi<br>Virtuaaliavustaja - sovellus | | | |
| Päiväys | 27.4.2019 | Sivumäärä/Liitteet | 37 |
| Ohjaaja(t)<br>Jussi Koistinen Lehtori, Keijo Kuosmanen Lehtori | | | |
| Toimeksiantaja/Yhteistyökumppani(t)<br>VideoVisit Oy | | | |


Tiivistelmä

Opinnäytetyön aiheena oli kehittää VideoVisit Oy:lle web-sovellus. Kohderyhmänä toimivat vanhukset ja muistisairaat ihmiset. Sovelluksen tarkoituksena oli tuottaa videopohjaisia ilmoitus- ja muistutusviestejä. Toiminnallisuuksiin kuului ennalta suunnitellun päiväohjelman näyttäminen selainpohjaisessa käyttöliittymässä ja käyttäjän vastausten tallennus yrityksen tietokantaan. Videoviestit oli nauhoitettu ennalta prototyyppiä varten suunnittellun päiväohjelman mukaisesti ja ne oli määrä näyttää käyttäjällä päiväohjelman mukaiseen kellonaikaan.

VideoVisit Oy:lla oli olemassa oleva tietokantapalvelin ja back end jota hyödynnettiin sovelluksen kehityksessä. Tietokantana toimi MySQL, back end kielenä PHP ja data migraatiokirjastona Doctrine.
Olemassa olevaan kantaan tuli tehdä tietokantasuunnitelma uusia tauluja varten ja back endiin piti luoda modelit tietokantakyselyitä varten. Front endiä varten täytyi kirjoittaa uusi REST API rajapinta yirtyksen back endiin.

Front end toteutettiin ReactJS kielellä. Käyttäjä tunnistautui Axioksen kautta API rajapinnan yli yrityksen back endiin, josta hänelle noudettiin kyseisen päivän ohjelma JSON muodossa. Palautunut ohjelma siirrettiin Reduxin tietovarastoon. Javascriptillä haettiin seuraava ohjelman mukainen video. Jos yksikään video ei ollut käynnissä, laitettiin sovellus stand-by-tilaan. Videoiden avulla kerättiin käyttäjästä tietoja, jotka tallennettiin yrityksen tietokantaan. Valmis projekti käännettiin Webpackin avulla yhdeksi tiedostoksi.

Lopputuloksena syntyi selainpohjainen sovellus, jonka avulla vanhuksia ja muistisairaita ihmisiä voitiin muistuttaa ja tiedottaa päivän tapahtumista. Sovellus pystyi toimimaan itsenäisesti tai verkkosivulle upotetuna komponenttina.



Avainsanat

ReactJS, Webpack, PHP, Doctrine


Julkinen

SAVONIA UNIVERSITY OF APPLIED SCIENCES THESIS

Abstract

| | | | |
|---|---|---|---|
| Field of Study<br>Technology, Communication and Transport | | | |
| Degree Programme<br>Degree Programme in Information Technology | | | |
| Author(s)<br>Joonas Hakkarainen | | | |
| Title of Thesis<br>Virtual Assistant Application for Elderly and Memory Disordered People | | | |
| Date | 27 April 2019 | Pages/Appendices | 37 |
| Supervisor(s)<br>Mr Jussi Koistinen, Senior Lecturer and Mr Keijo Kuosmanen, Senior Lecturer | | | |
| Client Organisation /Partners<br>VideoVisit Oy | | | |


Abstract

The aim of the thesis was to develop a web-based application. The project was ordered by VideoVisit Oy. The target group for the application was the elderly and memory disordered people. The purpose of the application was to display a video-based notification and reminder messages for the users. The messages were part of a daily schedule that was predetermined for the users. The messages collected feedback through user interactions. Afterwards they were saved to the company's database. The videos were prerecorded to match the messages in the daily schedule.

The application was developed to VideoVisit's existing back end and database. Back end was written in PHP and the database engine was MySQL. Data migrations were handled through Doctrine libraries. New API interface as well as required database tables were designed and developed to the company's server.

Front end was written in ReactJS programming language. User authentication to the new API interface was handled by Axios HTTP client library. Hash string was used as the primary authentication method. The HTTP request returned the user's daily schedule which then was saved to Redux state container. If none of the daily videos were playing the application was in stand-by-mode. The final version was translated to a minified Javascript file using Webpack.

As a result of this thesis, a web-based application for the elderly and memory disordered people was developed. The application can be deployed as a modular component or as an embedded subcomponent.



Keywords

ReactJS, Webpack, PHP, Doctrine


Public

SISÄLTÖ

# 1 TERMIT JA LYHENTEET

Tässä kappaleessa käsitellään opinnäytetyöhän liivyiä käsitteitä ja termejä.

## 1.1 ReactJS

Javascript kirjasto, joka on kehitetty käyttöliittymien rakentamista varten. Sen ylläpitäjänä toimii Facebook. (Wikipedia, ReactJS)

## 1.2 Javascript

Pääasiassa web-ympäristössä käytettävä dynaaminen komentosarjakieli.
Javascriptiä käytetään myös palvelinten verkko-ohjelmoinnissa. (Wikipedia, Javascript)

## 1.3 Redux

Ennustettava tilamääritelmä Javascript sovelluksille. (Redux, About)

## 1.4 ES6

ECMAScript on ohjelmointikieli määritelmä, jonka on standardoinut Ecma International. Numero 6 tarkoittaa, että kyseessä on standardin kuudes versio. (Wikipedia, ES6)

## 1.5 HTTP

Hypertext Transfer Protocol on palvelinten ja selainten käyttämä tiedonsiirtoprotokolla. (Wikipedia, HTTP)

## 1.6 Node.js

Avoimen lähdekoodin Javascript runtime-ympäristö. Sen avulla voidaan kirjoittaa Javascript koodia palvelimella. (Wikipedia, Node.js)

## 1.7 Axios

Promise pohjainen HTTP asiakasohjelma selaimelle ja Node.js ohjelmointikielelle. (Github, Axios)

### 1.8 Webpack

Avoimen lähdekoodin Javascript-pohjainen sovellus, jonka avulla paketoidaan ja yhdistetään erilaisia kirjastoja moduuleiksi. (Wikipedia, Webpack)

### 1.9 Babel

Avoimen lähdekoodin Javascript-kääntäjä, jota käytetään web-sovelluksissa. (Wikipedia, Babel)

### 1.10 Sass

Syntactically awesome style sheets on tyylitiedostokieli, joka käännetään Cascadin Style Sheets kielelle. Sen avulla luodaan tyylejä web-pohjaiselle sovellukselle. (Wikipedia, Sass)

### 1.11 NGINX

Yksi maailman suosituimmista WWW- ja proxy-palvelimista. (Wikipedia, NGINX)

### 1.12 PHP

Ohjelmointikieli, jota käytetään erityisesti palvelinympäristössä. Sisältää myös laajan luokkakirjaston. Lyhenne tulee sanoista 'Hypertext preprocessor'. (Wikipedia, PHP)

### 1.13 Doctrine

Kokoelma erilaisia PHP-kirjastoja, joiden pääpaino on tietokannan varastoinnissa ja objektien arkistoinnissa. (Doctrine Project, Doctrine)

### 1.14 REST

Representational State Transfer on HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen. (Wikipedia, REST)

### 1.15 API

Application programming interface on ohjelmointirapapinta. Sen avulla voidaan määritellä kuinka eri ohjelmat voivat tehdä pyyntöjä ja vaihtaa tietoja eli keskustella keskenään (Wikipedia, API)

### 1.16 Vagrant

Avoimen lähdekoodin sovellus, jonka avulla voidaan luoda ja ylläpitää virtuaalisia kehitysympäristöjä. (Wikipedia, Vagrant)

### 1.17 VirtualBox

Käyttöjärjestelmien ja ohjelmian virtualisoimiseen suunniteltu sovellus. Se tukee lähes kaikkia PC-pohjaisia käyttöjärjestelmiä. (Wikipedia, VirtualBox)

### 1.18 Back end

Palvelumalli, jonka avulla web- ja mobiilisovellukset voivat yhdistää palvelunsa serveriin ja API-rajapintaan. (Wikipedia, Back end)

### 1.19 Front end

Sovelluksen näkyvä kerros, jossa sijaitsee käyttöliittymä. Hakee palvelimelta tietoja ja esittää sen käyttäjälle. (Wikipedia, Front end)

### 1.20 VAU

Opinnäytetyöstä käytettiin tilaajayrityksen sisällä projektinimeä 'VAU'. Raportissa viitataan usein 'VAU'-termiin, mikä on lyhenne virtuaaliavustajasta. Virtuaaliavustaja on videokomponentti, joka toistaa käyttäjälle hänelle määrätyn päiväohjelman.

# 2 JOHDANTO

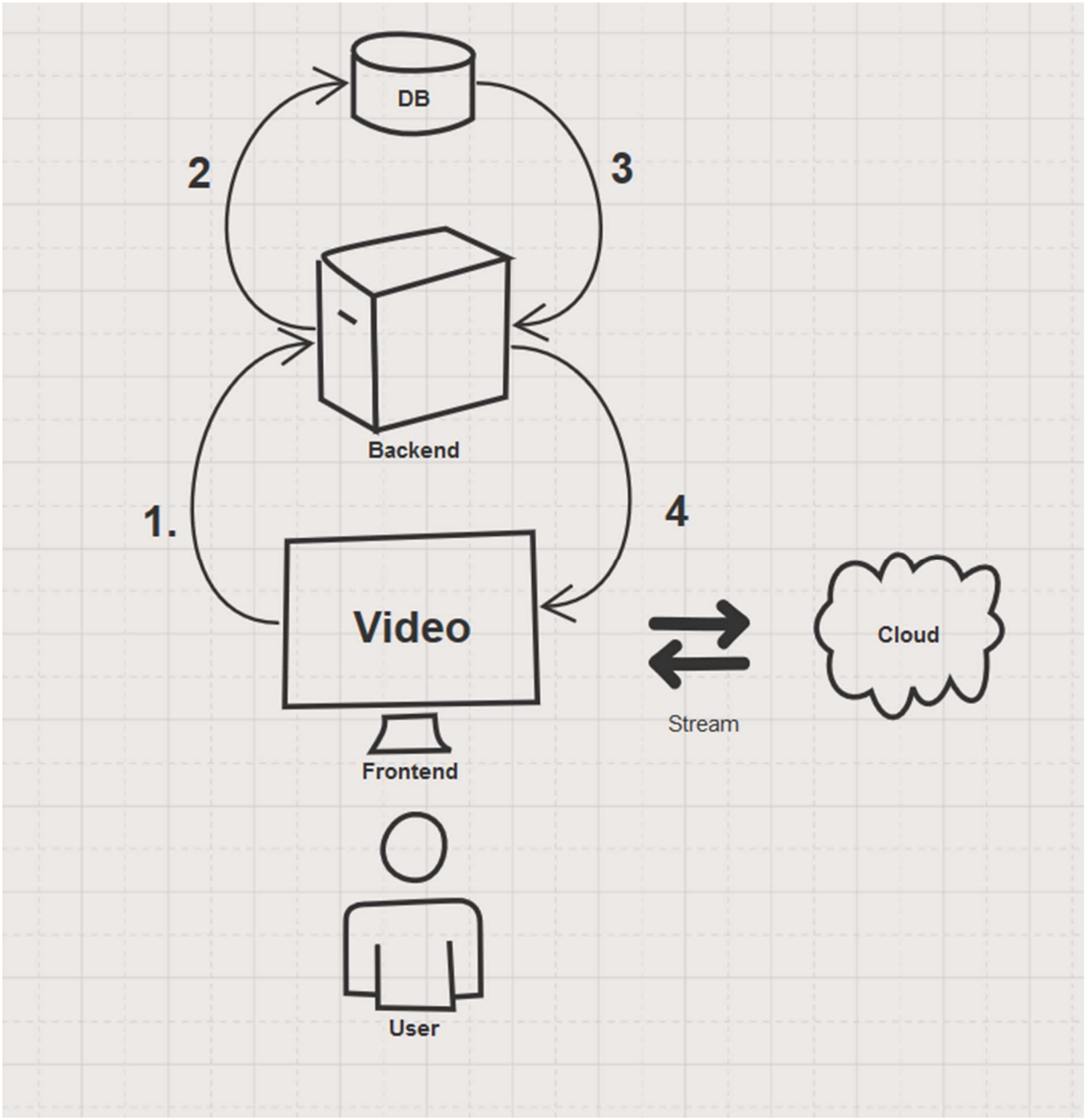
Opinnäytetyön tilaajana oli VideoVisit Oy. Työ toteutettiin vuosien 2018 ja 2019 välisenä aikana. Tavoitteena oli luoda vanhuksille ja muistisaraille ihmisille kohdistetu web-sovellus. Kotihoidon piirissä olevat vanhukset tarvitsevat apua jokapäiväisissä askareissa. Hoitajat eivät pysty olemaan aina läsnä varmistamassa, että vanhukset muistavat syödä ja ottaa heille määrätyt lääkkeet. Kotihoidossa olevien vanhusten valvonnan tueksi on tarve kehittää sovellus. Sovelluksen tarkoitus oli tuottaa käyttäjälle videopohjaisia ilmoituksia ja muistutusviestejä. Videoiden sisältö suunniteltiin päiväohjelma formaattiin. Päiväohjelma koostui ajastetuista videoviesteistä, joiden avulla käyttäjästä kerättiin tietoja. Tiedot tallennettiin yrityksen tietokantaan raportointia ja tiedon analyosintia varten. Työssä hyödynnettiin tilaajan jo olemassa olevaa back end ympäristöä. Back end oli toteutettu PHP-ohjelmointikielellä ja tietokantamigraatiot Doctrine kirjaston avulla. Tilaajan back endiin tehtiin suunnitelma uuden API rajapinnan ja ja tietokantataulujen integroimiseksi. Front end toteutettiin ReactJS-ohjelmointikielellä ja siinä hyödynnettiin valmiita komponenttikirjastoja, kuten ‘Material-UI’ ja ‘ReacPlayer’. Sovelluksesta haluttiin täysin itsenäinen komponentti, jonka voisi upottaa asiakkaan verkkosivustolle. Valmis sovellus käännettiin Webpackin avulla yhdeksi Javascript-tiedostoksi, joka sisälti koko projektirakenteen.

# 3 SUUNNITTELU

Ennen kehitystyön aloitusta tuli laatia suunnitelma prosessista, tietokannasta, käyttöliittymästä ja sovellusprototyypin päiväohjelmasta.

## 3.1 Prosessikuvaus

Opinnäytetyö koostui neljästä eri osasta: Käyttäjän laitteesta, tilaajan portal-palvelimesta, tietokantapalvelimesta ja pilvipalvelusta (KUVA 1). Prosessi lähti liikkeelle käyttäjän laitteesta, kun sovellus avattiin selaimessa. Tämä voidaan jakaa viiteen eri vaiheeseen. Ensimmäisessä vaiheessa käyttäjän selain lähetti HTTP pyynnön API-rajapinnan avulla portal-palvelimelle. Palvelin tunnisti käyttäjän tämän lähettämän hash stringin avulla, jonka jälkeen noudettiin tietokannasta käyttäjälle määritetty päiväohjelma. Tulokset palautuivat tietokannasta palvelimelle, joka välitti ne edelleen JSON muodossa käyttäjän selaimelle. Front end haki päivän seuraavan videon palautuneiden tietojen listalta. Kun ohjelman oli määrä alkaa, niiin se striimattiin pilvipalvelun ylitse käyttäjän ruudulle.




KUVA 1. Prosessikuvaus

## 3.2 Tietokantasuunnitelma

Opinnäytetyötä varten tuli suunnittella yrityksen olemassa olevaan tietokantaan sovelluksen määritelmän mukaiset taulut. Suunnitelmassa keskeisimmäksi tauluksi muodostui 'question_group'-niminen taulu (KUVA 2). Sinne luotiin ryhmät, joiden avulla voitiin yhdistää joukko haluttuja tapahtumia yhden nimikkeen alle. Kantaan luotiin ryhmä jokaista viikonpäivää kohden. Lisäksi luotiin ns. päivittäinen ryhmä, jonka sisältämät tapahtumat toistuvat jokaisena viikonpäivänä.

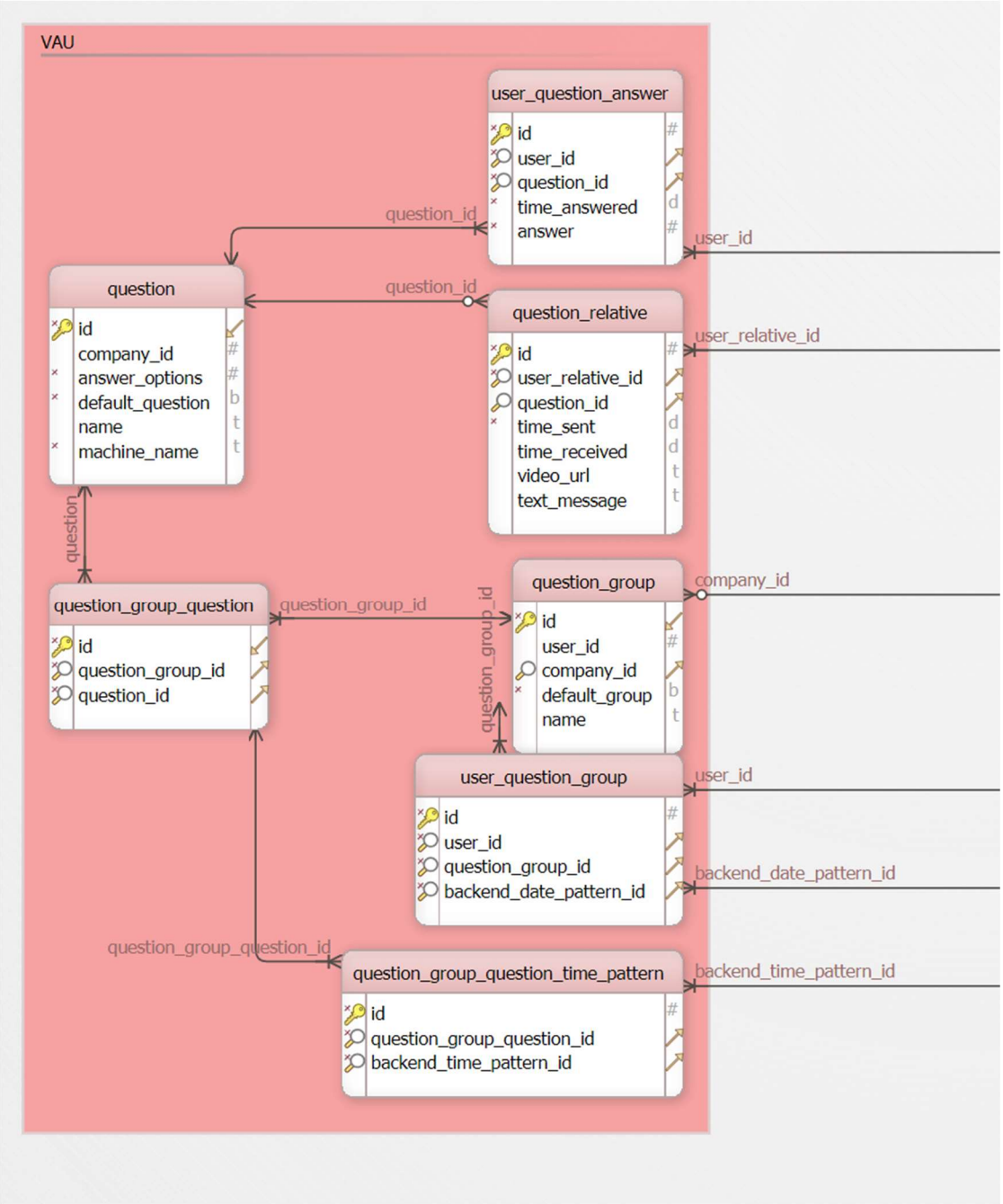
'User_question_group'-taulun (KUVA 2) tehtävä oli nimetä kullekin käyttäjälle hänelle määrätyt ryhmät, sekä sitoa kyseinen ryhmä tiettyyn ajankohtaan.

'Question_group_question'-taulu puolestaan yhdisti tapahtuman tiettyyn ryhmään.

'Question'-taulu sisälti itse tapahtumat ja kysymykset, joita käyttäjälle haluttiin esittää. Kysymys piti voida tarvittaessa asettaa ryhmän oletuskysymykseksi 'default_question' kentän avulla. Oletus kysymys tarkoitti tapahtumaa, mikä käyttäjälle näytettiin, kun yhtään ajastettua tapahtumaa ei ollut käynnissä.

'User_question_answer'-tauluun oli tarkoitus tallentaa kaikki käyttäjien vastaukset heille tapahtuman yhteydessä esitetystä kysymyksistä.

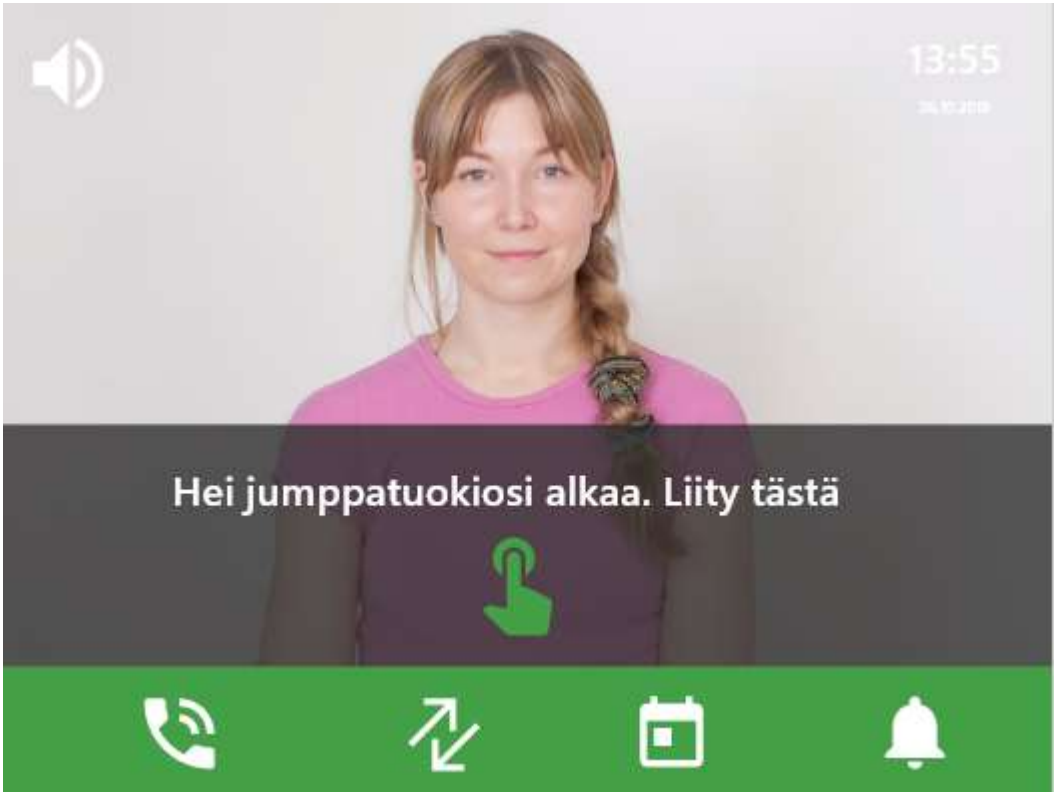
'Question_relative'-tauluun voitiin tallentaa tapahtumaan liittyvän videon sijainti pilvipalvelimella, valinnainen tekstiviesti, kuittaus tapahtuman alkamisesta ja omaiseksi määrätyn henkilön id-tunnus.




KUVA 2. Päiväohjelman tietokantasuunnitelma

## 3.3 Käyttöliittymä

Käyttöliittymä tuli suunnitella selainpohjaiseksi. Sen piti olla responsiivinen ja helposti muokattavissa oleva, mahdollista jatkokehitystä varten. Käyttöliittymään UI-suunnitelmasta piirrettiin mallikuvat Adobe XD-ohjelmalla (KUVA 3). Videotapahtumien tuli olla keskeisessä osassa käyttöliitymää, mutta tilaa piti silti jättää mahdollisille laajennuksille.




KUVA 3. Käyttöliittymän UI-suunnitelma.

## 3.4 Päiväohjelma

Opinnäytetyön päiväohjelmaksi valittiin yksinkertainen malli (KUVA 3), jonka avulla oli helppo esitellä sovelluksen keskeisimpiä toiminnallisuuksia. Tapahtumia oli kolmenlaisia: Ilmoituksia, muistutuksia ja stand-by-tapahtumia.

Ilmoitusten suurin eroavaisuus muistutuksiin oli se, että ne eivät sisältäneet kysymyksiä. Niiden avulla käyttäjää pyrittiin muistuttamaan syömisistä ja lääkkeenottamisista. Ilmoitukset odottivat ainoastaan käyttäjän kuittausta viestin lukemisesta.

Muistutukset olivat tapahtumia, joidenka avulla käyttäjistä ja heidän hyvinvoinnistaan voitiin kerätä tietoja erilaisten kysymysten muodossa.

Stand by tarkoitti hetkeä, jolloin ruudulla ei pyörinyt mitään ajastettuja tapahtumia.

Opinnäytetyön päiväohjelman suunnitelma sisälti yhdeksän ajastettua tapahtumaa pitkin päivää. Sovellukselle oli tarve kehittää tila, mikä aktivoituisi, kun yksikään tapahtuma ei olisi käynnissä. Tässä tilassa käyttäjä voisi itse aktivoida ohjelman koskettamalla tai klikkaamalla ruutua.

Prototyyppiversiota varten luotiin viikon jokaiselle päivälle sama ohjelma.

| | Maanantai | Tiistai |
|---|---|---|
| 6.00<br>7.00 | Aamunavaus | Aamunavaus |
| 8.00<br>9.00 | Lääkkeenotto & aamupala | Lääkkeenotto & aamupala |
| 10.00<br>11.00 | Stand by - aamu | Stand by - aamu |
| 12.00<br>13.00 | Lounasaika | Lounasaika |
| | Oletteko muistaneet syödä? | Oletteko muistaneet syödä? |
| 14.00<br>15.00 | Stand by - iltapäivä | Stand by - iltapäivä |
| 16.00<br>17.00 | Päivällinen | Päivällinen |
| 18.00 | Oletteko muistaneet syödä? | Oletteko muistaneet syödä? |
| 19.00<br>20.00 | Iltapala | Iltapala |
| 21.00 | Oletteko syöneet jo iltalääkkeen? | Oletteko syöneet jo iltalääkkeen? |
| 22.00 | Hyvää yötä! | Hyvää yötä! |
| 23.00<br>24.00 | Stand by - Yö | Stand by - Yö |

**Erilaisia viestityyppejä:**

| | *Ajastettuja tapahtumia* | *Näillä kerätään tietoja* |
|---|---|---|
| | Ilmoitukset | Muistutukset |
| 1 | Aamunavaus | oletteko jo syöneet aamupalan ? |
| 2 | Aamupalan aika | oletteko jo ottaneet aamulääkkeen? |
| 3 | Keskipäivä & lounasaika | Oletteko jo syöneet lounaan ? |
| 4 | Iltapäivä ja välipala | Oletteko jo syöneet päivällisen ? |
| 5 | Iltapala | Oletteko jo ottaneet iltalääkkeen ? |
| 6 | Lääkkeenotto | |
| 7 | Hyvää yötä | |

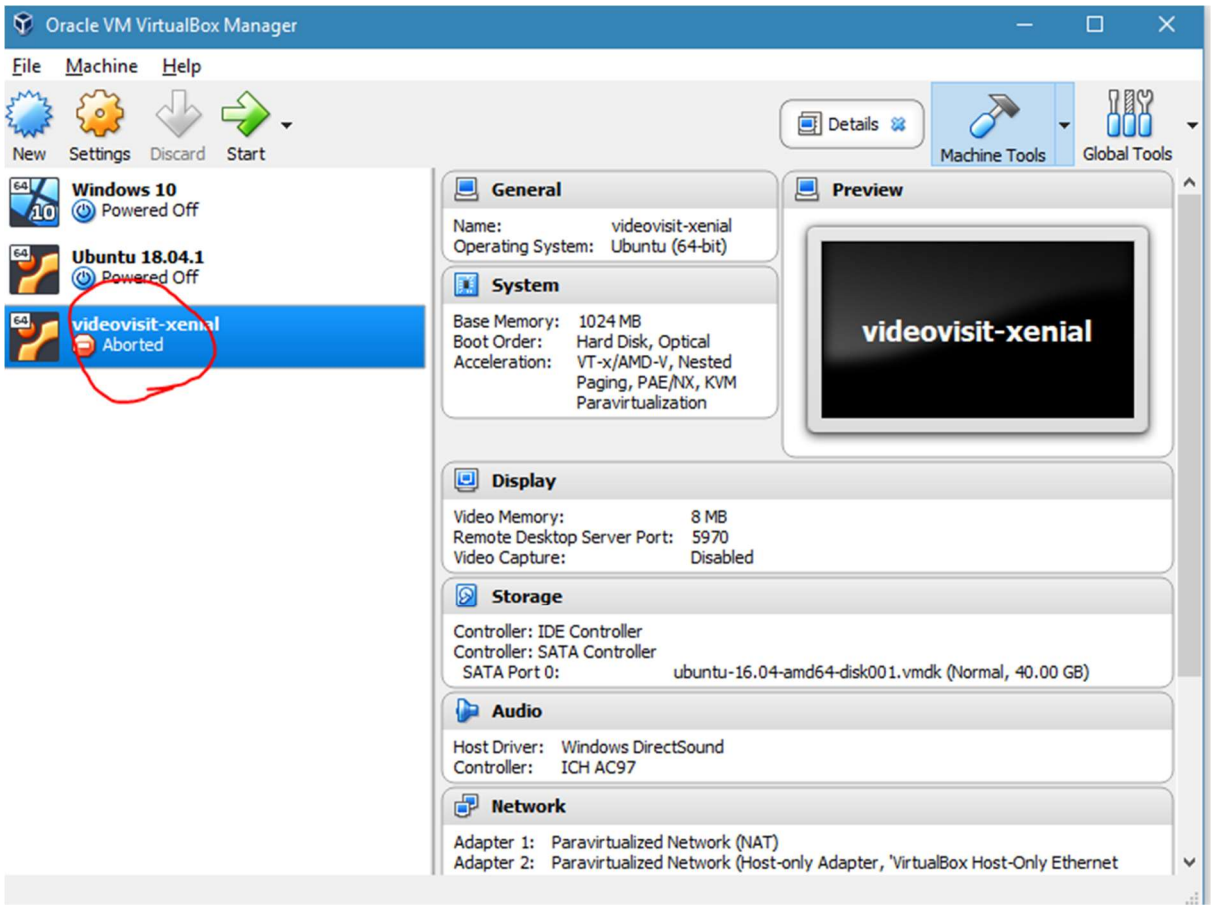
KUVA 3. Opinnäytetyötä varten suunniteltu päiväohjelma

# 4 TOTEUTUS

Opinnäytetyötä kehitettiin vuosien 2018 ja 2019 välisenä aikana. Työn tilaajan kanssa pidettiin kerran viikossa välikatsaus, jossa käytiin läpi työn etenemisen vaiheita. Seuraavissa kappaleissa käsitellään toteutuksen etenemistä aihe kerrallaan.

## 4.1 Kehitysympäristö

Tietokanta rakennettiin ensin lokaaliin ympäristöön kehitystyötä varten. Lokaalin ympäristön tuli vastata yrityksen tuotannon ympäristöä. Sen avulla oli turvallista kehittää ja testata projektia opinnäytetyön aikana. Kehitysympäristönä toimi vagrantin avulla luotu Ubuntu 16.04 Server, johon oli asennettu NGXIN, MySQL ja PHP. Serveriä käytettiin komentorivin kautta ssh-yhteyden ylitse. Serveri asennettiin omalla koneella olevalle VirtualBoxille (KUVA 4).




KUVA 4. Ubuntu Server asennettuna VirtualBoxille

## 4.2 Doctrine

Valmis viikko-ohjelman suunnitelma kirjoitettiin ylös tietokannan datatiedostoon. Tämän jälkeen tietokantasuunnitelman taulut kirjoitettiin Doctrinen skeematiedostoon (KUVA 5). Seuraavaksi tietokanta tuli rakentaan uudelleen, jotta opinnäytetyön taulut ja testidata ilmestyisivät lokaalin kehitysympäristön MysSQL tietokantaan.

Tätä varten tuli kirjautua ssh-yhteyden avulla palvelimella ja Doctrinelle syötettiin komento: 'PHP doctrine-cli build-all-reload'.

Doctrine tuhosi ensin vanhan tietokannan, jonka jälkeen kanta rakennettiin uudelleen aiemmin muokatun skeematiedoston mukaisesti. Tämän jälkeen Doctrine syötti tietokantaan tietoa datatiedoston avulla.

Lopuksi kirjauduttiin vielä palvelimen MySQL tietokantaan ja testattiin yksinkertaisella haulla päiväohjelman olemassaolo (KUVA 6).

```
VauQuestionRelative:
    columns:
        id:
            primary: true
            autoincrement: true
            type: integer(10)
        ecare_contact_id: integer(10)
        vau_question_id: integer(10)
        time_sent: timestamp
        time_received: timestamp
        video_url: string(1000)
        text_message: string(1000)

VauUserQuestionGroup:
    columns:
        id:
            primary: true
            autoincrement: true
            type: integer(10)
        user_id: integer(10)
        vau_question_group_id: integer(10)
        time_pattern_id: integer(10)

VauUserQuestionAnswer:
    columns:
        id:
            primary: true
            autoincrement: true
            type: integer(10)
        user_id: integer(10)
        vau_question_id: integer(10)
        time_answered: timestamp
        answer: integer(2)
```

KUVA 5. Ote Doctrine skeemasta.

```
mysql> select * from vau_question_group;
+----+---------+------------+---------------+-----------------------------+
| id | user_id | company_id | default_group | name                        |
+----+---------+------------+---------------+-----------------------------+
|  1 |       1 |          1 |             1 | StandBy kysymykset          |
|  2 |       1 |          1 |             1 | Maanantain ohjelma          |
|  3 |       1 |          1 |             1 | Tiistain ohjelma            |
|  4 |       1 |          1 |             1 | Keskiviikon ohjelma         |
|  5 |       1 |          1 |             1 | Torstain ohjelma            |
|  6 |       1 |          1 |             1 | Perjantain ohjelma          |
|  7 |       1 |          1 |             1 | Lauantain ohjelma           |
|  8 |       1 |          1 |             1 | Sunnuntain ohjelma          |
|  9 |       1 |          1 |             1 | JokapÃ¤ivÃ¤inen ohjelma     |
+----+---------+------------+---------------+-----------------------------+
9 rows in set (0.00 sec)
```

Kuva 6. MySQL tietokannan testaus. ‘Question_group’-taulun sisältö

## 4.3 Mallit

Jokaista tietokannan uutta taulua kohden kirjoitettiin PHP:lla model eli mallitiedosto (KUVA 7). Näiden mallien avulla Doctrine suoritti tauluun kohdistuvat tietokantahaut.

Tiedosto oli käytännössä PHP-luokka, jonka sisältämät funktiot suorittivat MySQL-kyselyitä. Funktioita kutsuttiin controller-tiedostosta käsin käyttäjän lähettämien API-kutsujen mukaisesti. mallit suorittivat haun ja palauttivat sen tulokset controller-tiedostolle.

GetRows-funktion avulla (KUVA 8) käyttäjälle haettiin tietokannasta hänelle asetettu päiväohjelman tiedot. Oletuksena funktio palautti kannasta koko viikon ohjelman. Funktiolle oli tarve kirjotitaa tuki erilaisia suodattimia varten. Suodattimien avulla voitiin hakea esimerkiksi vain tietyn päivän ohjelman.

```php
<?php

/**
 * Model_VauQuestionGroup
 *
 * This class has been auto-generated by the Doctrine ORM Framework
 *
 * @package     ##PACKAGE##
 * @subpackage ##SUBPACKAGE##
 * @author      ##NAME## <##EMAIL##>
 * @version     SVN: $Id: Builder.php 6716 2009-11-12 19:26:28Z jwage $
 */
class Model_VauQuestionGroup extends Model_Base_VauQuestionGroup
{
    public static function insert($data)
    {
        if (!empty($data['companyId']) && !empty($data['userId']) && !empty($data['name']))
        {
            $question                   = new Model_VauQuestionGroup();
            $question->user_id    = $data['userId'];
            $question->company_id       = $data['companyId'];
            $question->default_group = $data['defaultGroup'];
            $question->name             = $data['name'];

            $question->save();
            return $question->get('id');
        }
        return false;
    }
```

KUVA 7. Insert funktion avulla voidaan lisätä ryhmiä tietokantaan.

```
public static function getRows($filters)
{
    $q = Doctrine_Query::create()
            ->select('vauqg.id, vauqg.user_id, vauqg.company_id, vauqg.default_group,
                vauqg.name, vauuqg.id,
                tp.id, tp.show_mon, tp.show_tue, tp.show_wed, tp.show_thu, tp.show_fri,
                    tp.show_sat, tp.show_sun')
            ->from('Model_VauQuestionGroup vauqg')
            ->innerJoin('vauqg.VauUserQuestionGroup vauuqg ON (vauqg.id =
                vauuqg.vau_question_group_id)')
            ->innerJoin('vauuqg.TimePattern tp ON (vauuqg.time_pattern_id = tp.id)');


    if (!empty($filters))
    {
        if (!empty($filters['id']))
        {
            $q->andWhere('vauqg.id IN (?)', $filters['id']);
        }
        if (!empty($filters['user_id']))
        {
            $q->andWhere('vauqg.user_id IN (?)', $filters['user_id']);
        }
        if (!empty($filters['companyId']))
        {
            $q->andWhere('vauqg.company_id IN (?)', $filters['companyId']);
        }
        if (isset($filters['defaultGroup']))
        {
            $q->andWhere('vauqg.default_group = ?', $filters['defaultGroup']);
        }
        if (isset($filters['dayOfWeek']))
        {
            $q->andWhere('tp.' . $filters['dayOfWeek'] . ' = ?', true);
        }
    }

    $result = $q->fetchArray();
    return $result;
}
```

KUVA 8. GetRows-funktion avulla noudetaan käyttäjän päiväohjelma

## 4.4 Controller-tiedosto

Jotta käyttäjän selain pystyisi ottamaan yhteyden yrityksen palvelimeen, tuli tätä varten kehittää uusi API rajapinta. Tätä varten luotiin uusi controller-tiedosto (KUVA 10).

Controller on PHP-luokka, jonka sisältämät funktiot määrittävät pyynnöt, joita selain voi palvelimelle esittää HTTP-requestin muodossa.

Luokka sisältää myös init-funktion, joka ajetaan ensin, aina kun käyttäjä lähettää pyynnön palvelimelle. Init-funktion avulla tunnistetaan käyttäjän oikeudet suorittaa komentoja palvelimen API-rajapinnassa.

Käyttäjän tunnistautuminen tapahtui Doctrinen ylitse tietokantahaulla, jossa verrattiin käyttäjältä saatua hash-tunnistetta tietokannasta löytyvään hash-tunnisteeseen. Hash-tunniste annetaan vain tilaajayrityksen tietokantaan rekisteröidyille käyttäjille. Opinnäytetyössä oletetaan, että käyttäjä on jo aiemmin tunnistaunut yrityksen sivustolle ja saanut tätä kautta haltuunsa sovelluksen käyttöön

vaadittavan hash-tunnisteen. Tunniste salattu merkkijono, joka muodostetaan käyttäjälle, kun hänelle on myönnetty oikeudet sivuston käyttöön. Tunniste syötetään muuttujana sivustolle opinnäytetyön alustavaan scriptaan (KUVA 9).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/javascript" src="./assets/bundle.js"></script>
    <script>
    vau(
      'http://nettineukkari.local', //base url
      '2c46f2178b06a497b3fe026efde5ccc3ad69f0asf', //hash
      'https://s3.eu-west-2.amazonaws.com/vau-videos/' //s3 base url
    );
    </script>
  </body>
</html>
```

KUVA 9. Sovellus alustetaan kutsumalla vau-nimistä funktiota.

Mikäli tietokanasta löytyi käyttäjän hash-tunnistetta vastaava käyttäjä, niin annettiin hänen jatkaa API rajapinnan käyttöä.
Mikäli käyttäjän lähettämä hash-tunniste ei vastannut yhtään tietokannasta löytyvää tunnistetta, niin estettiin API:n käyttö ja lähetettiin käyttäjälle tästä palautuksena virheilmoitus.

```php
<?php

class Api_VauController extends App_Controller_Action
{

    public function init()
    {
        parent::init();

        $this->_helper->layout->disableLayout();

        if ($this->getRequest()->isPost())
        {

            $post = $this->getRequest()->getPost();

            $rawPostData = false;

            $this->postData = $rawPostData !== false ? $rawPostData : $post;
            $user           = false;
            if (!empty($this->postData['hash']))
            {
                $user = Model_User::getByEcareHash($this->postData['hash'], true);
            }
            else if (!empty($this->user->id))
            {
                $user = Model_User::getUserData($this->user->id);
            }
            if ($user)
                $this->APIUser = $user;
            else
            {
                $this->_helper->json(array('error' => 'NO_ACCESS'));
            }
        }
    }
```

KUVA 10. API rajapinnan määrittävä controller-tiedosto.

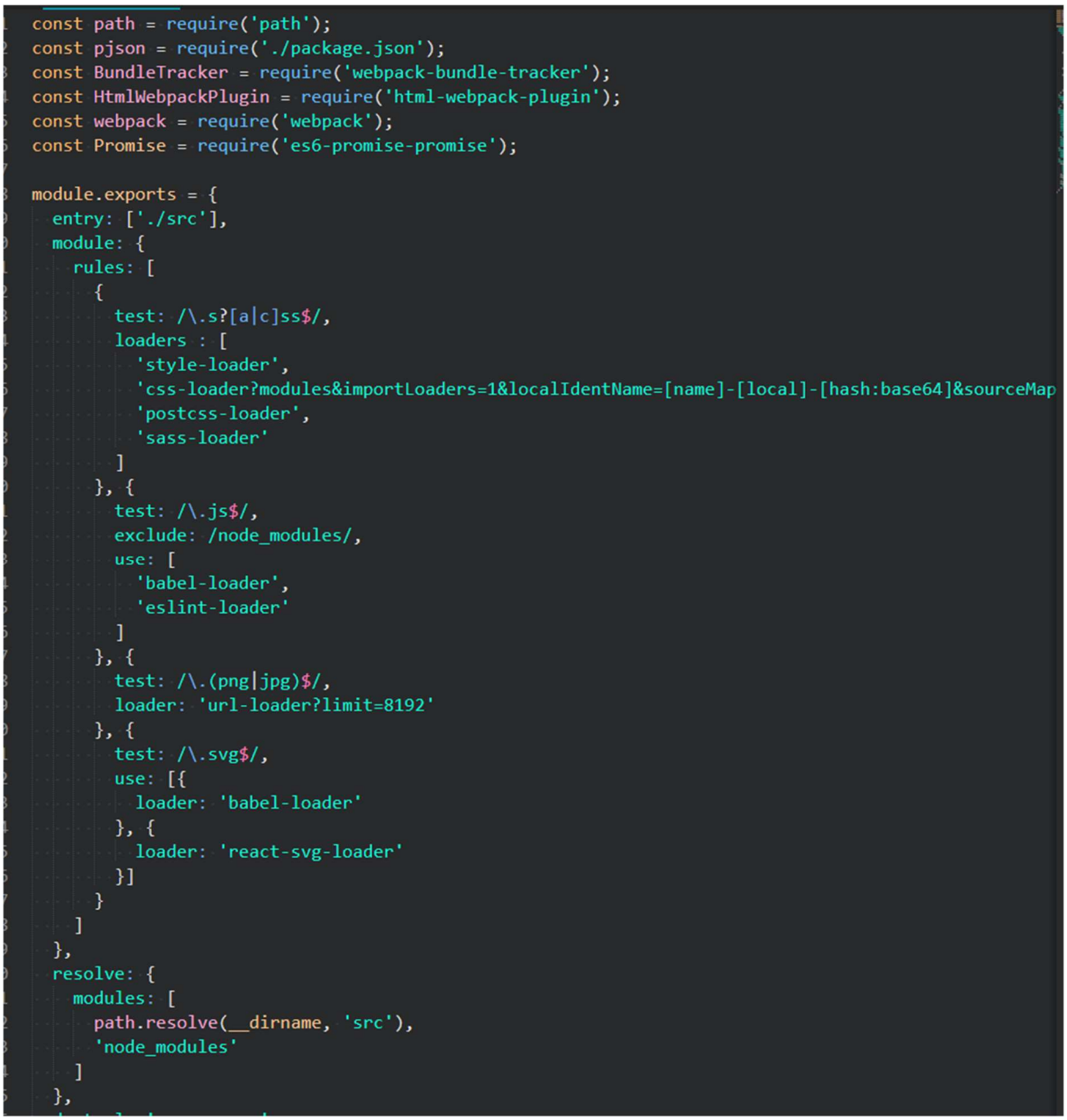
Päiväohjelman hakua varten kirjoitettiin controller-tiedostoon funktio nimeltä ‘getdailyroutinesAction’ (KUVA 11). Funktion otti vastaan tiedon sen hetkisestä viikonpäivästä. Viikonpäivää käytettiin ’getRows’-mallifunktion tietokantakyselyssä palauttamaan oikean päivän ohjelma.
Kun ’getRows’-funktio palautti kyselyn tuloksen, niin sen sisältämät ryhmät käytiin yksitellen läpi. for-loopin avulla. Opinnäytetyössä jokaiselle päivälle oli määritelty kolme eri ryhmää: Päivää osoittava ryhmä, stand by-ryhmä ja joka päivä toistuva ryhmä. Kullekin ryhmälle haettiin kannasta siihen kuuluvat kysymykset samalla tapaa kuin aiemmin, eli kutsumalla kohteen taulua vastaavaa mallitiedostossa sijaitsevaa funktiota. Tämän jälkeen haun tulokset palautettiin käyttäjän selaimeen JSON-muodossa.

```
        $question_groups = Model_VauQuestionGroup::getRows($filters);

        for ($i = 0; $i < sizeof($question_groups); $i++) {

            $question_group_questions_filter['vauQuestionGroupId'] = $question_groups[$i]['id'
                ;
            $question_group_questions = Model_VauQuestionGroupQuestion::getRows($
                question_group_questions_filter);

            $question_groups[$i]['question'] = [];

            foreach($question_group_questions as $key => $question_group_question) {

                $question_filter['id'] = $question_group_question['vau_question_id'];
                $question_groups[$i]['question'] = array_merge($question_groups[$i]['question'
                    , Model_VauQuestion::getRows($question_filter));

                $question_group_question_time_pattern_filter['vauQuestionGroupQuestionId'] = $
                    question_group_question['id'];
                $question_group_question_time_pattern =
                    Model_VauQuestionGroupQuestionTimePattern::getRows($
                    question_group_question_time_pattern_filter);

                $question_groups[$i]['question'][$key]['backend_time_pattern'] =
                    Model_TimePattern::getTimePatternForVau($
                    question_group_question_time_pattern[0]['time_pattern_id'], true)[0];

                $question_relative_filter['vauQuestionId'] = $question_group_question['
                    vau_question_id'];
                $question_groups[$i]['question'][$key]['question_relative'] =
                    Model_VauQuestionRelative::getRows($question_relative_filter)[0];

                $question_groups[$i]['question'][$key]['default_question'] = (int)$
                    question_groups[$i]['question'][$key]['default_question'];
                $question_groups[$i]['question'][$key]['answer_options'] = (int)$
                    question_groups[$i]['question'][$key]['answer_options'];
                $question_groups[$i]['question'][$key]['id'] = (int)$question_groups[$i]['
                    question'][$key]['id'];
            }
        }
        $this->_helper->json(array('data' => $question_groups, 'user_id' => $filters['user_id'
            ));
    }
}
```

KUVA 11. ‘getdailyroutinesAction’-funktio.

## 4.5 Webpack

Front end toteuttiin selainpohjaisena web-sovelluksena ReactJS ohjelmointikielen avulla. Tyylitiedostot luotiin Sass-kielellä. Käytetyt kielet ja kirjastot kännettiin Webpackin avulla selaimen ymmärtämään muotoon ja paketoitiin lopuksi yhdeksi ‘bundle.js’ tiedostoksi. Jotta Webpack osasi asettaa oikeat kääntäjät eri kielille, niin piti sitä varten kirjoittaa oma Webpack asetustiedosto (KUVA 12). Asetustiedosto käy läpi projektin sisäisiä tiedostopäätteitä ja osoittaa niille oikeat loaderit. Esimerkiksi ReactJS käännetään Babel loaderin avulla.



```
const path = require('path');
const pjson = require('./package.json');
const BundleTracker = require('webpack-bundle-tracker');
const HtmlWebpackPlugin = require('html-webpack-plugin');
const webpack = require('webpack');
const Promise = require('es6-promise-promise');

module.exports = {
  entry: ['./src'],
  module: {
    rules: [
      {
        test: /\.s?[a|c]ss$/,
        loaders : [
          'style-loader',
          'css-loader?modules&importLoaders=1&localIdentName=[name]-[local]-[hash:base64]&sourceMap
          'postcss-loader',
          'sass-loader'
        ]
      }, {
        test: /\.js$/,
        exclude: /node_modules/,
        use: [
          'babel-loader',
          'eslint-loader'
        ]
      }, {
        test: /\.(png|jpg)$/,
        loader: 'url-loader?limit=8192'
      }, {
        test: /\.svg$/,
        use: [{
          loader: 'babel-loader'
        }, {
          loader: 'react-svg-loader'
        }]
      }
    ]
  },
  resolve: {
    modules: [
      path.resolve(__dirname, 'src'),
      'node_modules'
    ]
  },
```

Kuva 12. Webpackin asetustiedosto.

## 4.6 NPM

Ennen käyttöliittymän kehitystyön aloitusta tuli ensin asentaa projektin vaatimat NPM-paketit. Asennus tapahtui luomalla ‘package. json’-tiedosto (KUVA 13), johon tuli kirjoittaa kaikkien vaadittavien pakettien nimi ja haluttu versionumero. ‘package. json’-tiedostoon määritettiin myös projektin käyttämät käynnistys-, testaus- ja rakennuskomennot. Tämän jälkeen määritettiin koodin syntaxin laatua valvovat ‘eslint’ asetukset.

Lopuksi NPM:lle annettiin käsky asentaa vaaditut riippuvuudet komentokehotteen kautta suorittamalla ‘npm install’ – käsky. Asennuksen jälkeen sovellus voitiin käynnistää Webpackin avulla suorittamalla ‘npm start’ komento.

```
{
  "name": "VAU Prototype",
  "version": "0.0.1",
  "description": "VAU Prototype",
  "main": "index.js",
  "repository": "git@git.videovisit.mobi:videovisit/vau.git",
  "scripts": {
    "start": "webpack-dev-server --hot --inline",
    "build": "webpack -p --config webpack.config.js",
    "test": "jest"
  },
  "author": "Joonas Hakkarainen",
  "license": "MIT",
  "eslintConfig": {
    "extends": [
      "airbnb-base",
      "eslint:recommended",
      "plugin:react/recommended"
    ],
    "env": {
      "browser": true
    },
    "plugins": [
      "react"
    ],
    "parserOptions": {
      "ecmaFeatures": {
        "jsx": true
      }
    },
    "rules": {
      "no-console": 0,
      "import/no-extraneous-dependencies": 0,
      "import/extensions": 0
    },
    "settings": {
      "import/resolver": {
        "webpack": {
          "config": "webpack.config.js"
        }
      }
    }
  },
```

KUVA 13. Ote ‘package. json’ asetuksista

## 4.7 Redux

ReactJS tarjosi ainoastaan komponenttikohtaisen tilanhallintajärjestelmän.
Sovelluksen kehitystyötä varten haluttiin ottaa käyttöön koko ohjelman laajuinen tilanhallintajärjestelmä. Tätä varten oli olemassa Redux-niminen kirjasto.
Redux voidaan jakaa karkeasti kolmeen osaan, jotka ovat: action, store ja reducer.

### 4.7.1 Store

Store sisälti Reduxiin tallennetun tilan, eli staten. Tila oli muuttumaton objekti, mikä takoitti sitä, että sitä ei voida muuttaa, vaan storen päivitys palautti aina uuden tilan.
Työtä varten sovellukselle kirjoitettiin oletustila (KUVA 14), mikä aktivoitui, kun sovellus käynnistettiin. Tällä tavoin sovellus saatiin toimimaan, ennen kuin palvelin palautti tiedon päiväohjelmasta.
Kaikki palvelimelta palautunut tieto tallennettiin storeen, josta se saatiin sovelluksen käyttöön yhdistämällä React komponentti Reduxiin.

```
export default function vau(state = {
  loading: false,
  data: null,
  error: null,
  videoBaseUrl: null,
  triggerQuestion: [],
  defaultTemplate: {
    id: -1,
    message: "Hei, kosketa minua jos haluat apua",
    backGround: 'anna',
    answer_options: 1,
  }
}, action) {
  switch (action.type) {
    case FETCH_VAU_QUESTIONGROUPS_SUCCESS:
      return {
        ...state,
        data: action.payload,
        loading: false
      }
```

KUVA 14. Storen oletustila.

### 4.7.2 Action

Actionin tehtävä oli lähetää päivityskomentoja reducerille. Komennot sisältivät tyypin ja valinnaisen datan. Päiväohjelman hakua varten luotiin ’fetchVauQiestionGroups’-niminen funktio (KUVA 15). HTTP requestia varten käytettiin axios-nimistä HTTP-client sovellusta. Axios lähetti pyynnön portal-palvelimen controllerltiedoston funktiolle, joka palautti tietokannasta saadun päiväohjelman. Saatu onnistunut tulos otettiin kiiinni then-metodi avulla, josta se lähetettiin edelleen reducerille.
Mikäli palvelin palautti virheen, niin saatiin siitä tieto catch-metodin avulla, josta välitettiin myöskin tieto reducerille.

```
40
41  export function fetchVauQuestionGroups(hash, dayOfWeek) {
42    return (dispatch, getState, {api}) => {
43      return api
44
45        .post('/vv/api/vau/getdailyroutines', qs.stringify({
46          'hash': hash,
47          'dayOfWeek' : dayOfWeek
48        }))
49        .then(response => dispatch({
50          type: FETCH_VAU_QUESTIONGROUPS_SUCCESS,
51          payload: response.data.data,
52          data: response
53        }))
54        .catch(err => dispatch({type: FETCH_VAU_QUESTIONGROUPS_FAILED, payload: err}));
55    }
56  }
57
```

KUVA 15. Async action, joka noutaa päiväohjelman palvelimelta

### 4.7.3 Reducer

Reducer otti vastaan actionin lähettämät tiedot. Action sisälti type- ja payload-muuttujat. Type-muuttujan avulla voitiin nimetä actionin lähettämä tapahtuma. Tämä mahdollisti sen, että tietoa voitiin suodattaa nimen perusteella oikean reducerin käsiteltäväksi. Storen tilaa päivitettiin switch-lauseen avulla (KUVA 16). Uutta tietoa ei tallenneta vanhan päälle, vaan state-objektista palautetaan kokonaan uusi versio, johon päivitetty tila on tallennettu. Kullekin hetkelle saatiin siis luotua oma kopio sen hetkisestä storen tilasta. Storen tilan muutosta pystyttiin seuraamaan lineaarisesti selaimesta käsin. Tämä helpotti testausta ja vianmääritystä.

```
}, action) {
  switch (action.type) {
    case FETCH_VAU_QUESTIONGROUPS_SUCCESS:
      return {
        ...state,
        data: action.payload,
        loading: false
      }
    case FETCH_VAU_QUESTIONGROUPS_STARTED:
      return {...state, loading: true}
    case FETCH_VAU_QUESTIONGROUPS_FAILED:
      return {
        ...state,
        error: action.payload,
        loading: false
      }
```

KUVA 16. Ote reducerista.

## 4.8 ReactJS

Projektissa käyettiin ReactJS versiota 16.8.6. Reduxin store kiinnitettiin ReactJs komponentteihin, mikä lisäsi sen Reactin elinkaariseurannan piiriin. Seuraavissa kappaleissa käsitellään työssä käytettyä React API: a.

## 4.9 Life cycle

React Component API:n metodeja kutsutaan 'life cycle'-metodeiksi, sillä ne käytännössä kuvaavat komponentin sen hetkisen elinkaaren tilaa. Näitä metodeja ovat esimerkiksi: componentDidMount, componentDidUpdate ja render. Aina, kun jokin Reactin life cyclen piirissä oleva muuttuja tai state päivittyy, niin komponentti renderöi itsensä uudestaan automaattisesti. Käyttäjälle tämä näkyy sivun päivityksenä. Opinnäytetyössä hyödynnettiin näitä metodeja seuraavalla tavalla

### 4.9.1 ComponentWillMount

'Component will mount' tarkoittaa hetkeä, kun React lataa komponentin. Tämä tapahtuu yleensä ainoastaan kerran, kun uutta komponenttia alustetaan. Viimeisimpien ReactJS päivitysten myötä

componenWillMount-metodi on vanhentunut ja luokan constructor-metodia suositellaan käytettäväksi sen sijasta. Opinnäytetyössä kuitenkin käytettiin vielä componenWillMount-metodia (KUVA 17). ComponentWillMount-metodin avulla kutsuttiin fetchUserModules-API-funktiota Reduxin actions tiedostosta. Lisäksi storeen tallennettiin sovelluksen vastaanottama hash-tunniste ja videoiden pilvipalvelun url-osoite.

```
componentWillMount() {
  const {
    setHash,
    setVauVideoUrl,
    startFetchingUserModules,
    fetchUserModules
  } = this.props.actions;

  setHash(this.props.hash);
  setVauVideoUrl(this.props.videoBaseUrl);
  startFetchingUserModules();
  setTimeout(() => {
    if (this.props.user.modules.data === null) {
      fetchUserModules();
    }
  }, 3000);
}
```

KUVA 17. componentWillMount-metodi

### 4.9.2 ComponentDidUpdate

React kutsuu componentDidUpdate-metodia aina kun komponentti on päivittynyt. Sen avulla kehittäjä voi suorittaa komponentin päivitysten välissä vaadittavia tarkituksia.
Opinnäytetyössä componentDidUpdate-metodin avulla tarkkailtiin, oliko sovelluksen lataus valmiina, eli voidaanko latauspalkki poistaa ja sovellus näyttää. Lataus katsottiin valmiiksi, kun storesta löytyi kaikki tarvittavat tiedot sovelluksen käynnistymiselle. Lataus merkittiin valmiiksi päivittämällä komponentin oman tilan muuttujan ’isLoadingComplete’ arvoa (KUVA 18).

```
componentDidUpdate() {
  if (isLoadingComplete(this.props) === true && this.state.isLoadingComplete === false) {
    this.setState({
      isLoadingComplete: true
    });
  }

  if (this.props.user.modules.data !== null && this.state.isLoadingComplete === false) {
    this.getModuleApiData();
  }
}
```

KUVA 18. componentDidUpdate-metodi tarkistaa onko sovelluksen lataus valmis.

### 4.9.3 Render

Render-metodin avulla (KUVA 19) voitiin määrittää mitä käyttäjä näkee, eli se sisältää komponentin visuaaliset osat. Visuaalisia osia ovat HTML-elementit ja mahdolliset alikomponentit.
Render kirjoitettiin palauttamaan, joko 'GridUI'-komponentti eli käyttöliittymä, tai latauspalkki. Javascriptin 'ternary operator' ilmaisun avulla käyttäjälle näytettiin latauspalkki, mikäli lataus oli kesken, tai käyttöliittymän sisältävä 'GridUI'-komponentti, mikäli lataus oli valmis. Header komponentin haluttiin aina olevan näkyvillä, jonka vuoksi se sijoitettiin 'ternary operator'-ilmaisun ulkopuolelle.

```
101   render() {
102     return (
103       <div className={styles.boilerplates}>
104         <Header controls={this.props.app.settings} />
105         {this.state.isLoadingComplete ?
106           (
107             <GridUI
108               layout={this.props.app.layout}
109               data={this.renderModules()}
110               settings={this.props.app.settings}
111             />
112           ) :
113           (
114             <Progress size={70} />
115           )
116         }
117       </div>
118     );
119   }
120 }
```

KUVA 19. Render-metodi.

## 4.10 Komponentit

Reactin renderöitävät osat koostuvat komponenteista. Komponentti on luokka tai funktio, joka palauttaa html-elementin, minkä React lisää domiin. Reactissa komponetteja ei jQuerylle tyypilliseen tapaan yleensä piiloteta, kun ne halutaan poistaa käyttöliittymästä, vaan ne poistetaan suoraan domista irroittamalla komponentti render-metodista.
Opinnäytetyössä käytettiin kolmea pääkomponenttia: lataus-, video- ja 'App' eli sovelluksen alustavaa komponenttia.

### 4.10.1 Constructor

ReactJS komponentit kirjoitettiin luokkapohjaisiksi. Luokille luotiin constructor-metodi, mikä käytännössä alusti luokan ja asetti sille oletusparametrit.
Opinnäytetyössä constructor-metodiin lisättiin komponentin tila-muuttuja ja kiinnitettiin komponentin käyttämät funktiot luokan 'this' -kontekstiin (KUVA 20).

```
export class App extends Component {
  constructor(props) {
    super(props);
    this.renderModules = this.renderModules.bind(this);
    this.getModuleApiData = this.getModuleApiData.bind(this);
    this.state = {
      isLoadingComplete: false
    };
  }
```

KUVA 20. App-komponentin constructor metodi.

### 4.10.2 Progress

Latauskomponentin (KUVA 21) kehityksessä käytettiin hyväksi Material-ui-nimistä komponenttikirjastoa. Material-ui:n kirjastosta haettiin 'CircularProgress'-niminen valmis komponentti, joka palautti ympyrän muotoisen latausindikaattorin.
Latauskomponentin ainut tehtävä oli palauttaa 'CircularProgress'-komponentti ennaltamääriteltyjen asetuksien perusteella. Asetuksia olivat: väri, luokan nimi ja koko.

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import CircularProgress from 'material-ui/CircularProgress';
import styles from './Progress.scss';

export default class Progress extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div className={styles.container}>
        <CircularProgress
          color={'#008000'}
          className={styles.progress}
          size={this.props.size}
        />
      </div>
    );
  }
}

Progress.propTypes = {
  size: PropTypes.number.isRequired
}
```

KUVA 21. Luokkapohjainen latauskomponentti.

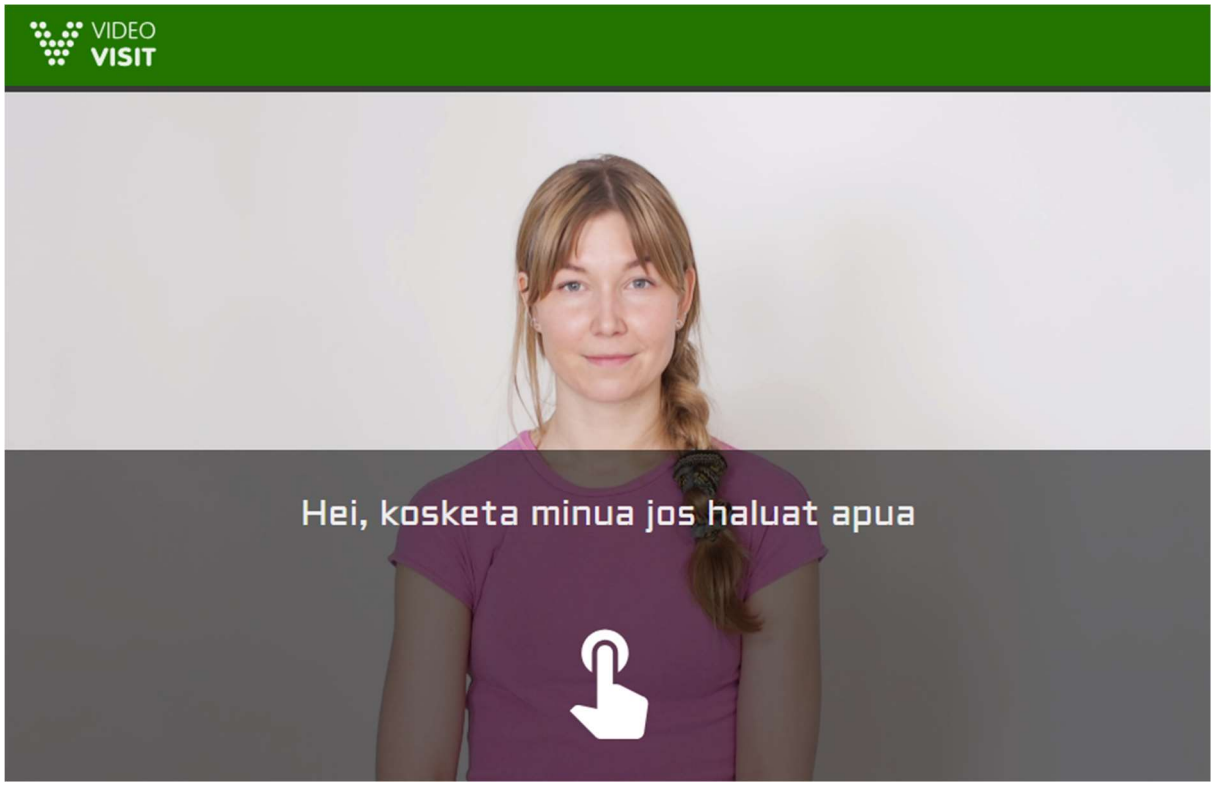
### 4.10.3 Video

Videokomponentti renderöi mediasoittimen. Opinnäytetyössä käytettiin 'ReactPlayer'-nimistä NPM-pakettia, joka sisälti valmiin komponentin videon toistamista varten (KUVA 22). Tämän avulla käyttäjälle toistettiin päivän tapahtumiin liittyvät videot. Mediasoittimelle määriteltiin käyttöön vielä halutut hallintapainikkeet, tapahtumatoiminnot, luokan nimi ja toistettavan videon osoite pilvipalvelimella. Videon osoite eli komponentin 'url'-muuttuja muodostettiin Amazon S3 pilvipalvelun perusosoitteesta ja kantaan tallennetun videon nimestä.

```
render() {
  const { url, playing, controls, loop, muted, volume } = this.state;
  return (
    <div className={styles.playerWrapper}>
      <ReactPlayer
        className={styles.videoplayer}
        url={url}
        playing={playing}
        controls={controls}
        loop={loop}
        muted={muted}
        volume={volume}
        onReady={() => this.onReady()}
        onClick={() => this.onPause()}
        onEnded={() => this.onEnded()}
        onPlay={() => this.onPlay()}
      />
    </div>
  );
  }
}
```

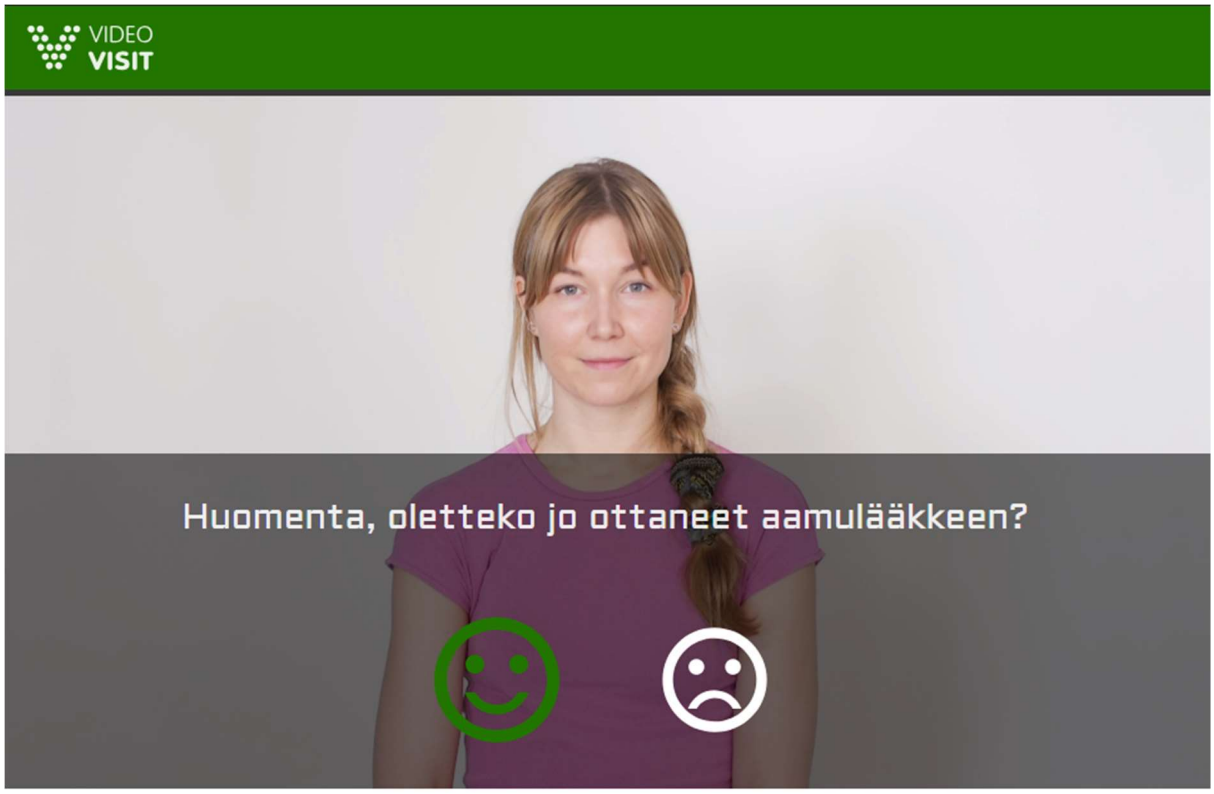
KUVA 22. Video-komponentin render metodi.

Komponentille tuli suunnitella kaksi eri tilaa: Stand-by- ja videotila. Stand-by-tilassa taustalla on videon sijasta valokuva (KUVA 23), jota klikkaamalla käyttäjä pystyi itse aktivoida videon. Itse aktivoidulla videolla käyttäjältä kysytään, tarvitseeko hän apua. Keskustelua ohjataan käyttäjän vastausten perusteella ja vastaukset tallennetaan tietokantaan




KUVA 23. Videokomponentin stand-by-tila

Videotila käynnistyi automaattisesti, kun päiväohjelmaan merkitty tapahtuma alkoi.
Sovellusta varten oli päiväohjelmaan suunniteltu muistutus lääkkeen ottamisesta (KUVA 24). Videolla ‘Anna’-niminen avustaja kysyi, onko käyttäjä muistanut ottaa aamulääkkeensä. Tämän jälkeen videon päälle tuotiin alue, jossa sama asia kysytään tekstimuodossa. Käyttäjältä pyydettiin vastausta kysymykseen hymynaamojen muodossa.
Mikäli käyttäjä ei vastannut kysymykseen kolmen minuutin kuluessa, niin siirrettiin tapahtuma jonoon 15:sta minuutin ajaksi. Tämän jälkeen tapahtuma esitettiin käyttäjälle uudestaan.
Kaikki vastaukset tallennettiin palvelimelle käyttäjän seurantaa varten. Myös tieto vastaamatta jättämisestä tallennetiin palvelimelle.
Vastaustietojen avulla omainen tai hoitaja voisi seurata käyttäjän päivän etenemistä ja yleistä tilaa.




KUVA 24. Muistutusvideo

# 5 YHTEENVETO

Opinnäytyön tavoitteena oli luoda VideoVisit Oy:lle videopohjaisia ilmoituksia tuottavat web-sovellus, vanhusten ja muistisairaiden tueksi. Tavoitteisiin päästiin ja sovellusta aletaan pilotoida yrityksen toimesta. Kehitystyö ei ole valmis ja sitä jatketaan opinnäytetyön jälkeen.

Opinnäytetyön jatkokehitys alkaa päivittämällä font end uusimpaan ReactJS versioon. Myös back end muutetaan vaihtamalla yrityksen portalin tilalle websocket serveri, joka tulee toimimaan front endin ja nykyisen back endin välissä. Tämä poistaa muun muassa selaimen tarpeen lähettää HTTP pyyntöjä palvelimelle tietyn väliajoin. Päiväohjelman suunnittelu ja laatiminen vaatii myös erillisen työkalun. Tämä tehdään erillisenä projektina jatkokehityksen yhteydessä.

Työn kehitys ei pysynyt aikataulussa, vaan venyi muiden työpaineiden takia. Määräajasta pystyttiin kuitenkin joustamaan, eikä viivästymisestä koitunut ongelmia.

Huonoa projektissa oli front endin kehitys vanhan ReactJS version päälle. Osa käytetyistä metodeista on vanhentunut, eikä niiden käyttöä enää suositella. Tämän vuoksi työ joudutaan kehittämään osittain uusiksi, kun ReactJS päivitetään uusimpaan version. Projektissa kytkettiin myös lähes kaikki komponentit kiinni Reduxiin, mikä on jälkeenpäin ajateltuna tarpeetonta. Riittäisi, että vain yksi pääkomponentti yhdistetään Reduxiin ja sen tilaa jaettaisiin eteenpäin sen kautta alikomponenteille.

Hyvää projektissa oli kehittää sovellus modulaariseksi. Tämä mahdollistaa sen, että sovellus ei ole sidottu mihinkään olemassa olevaan sivustoon, vaan se voidaan ottaa vapaasti käyttöön halutussa ympäristössä, esimerkiksi upotettuna asiakkaan verkkosivulle.

Projektin kehitystyö oli erinomainen oppitunti fullstack koodarille. Ohjelmointikielet ja tekniikat kehittyvät jatkuvasti ja kehittäjän täytyy pysyä niissä mukana. Projektia aloittaessani en tiennyt vielä React hooksien olemassa olosta, mutta nyt jos tekisin projektin uudestaan kirjoittasin sen täysin hook-pohjaiseksi ilman luokkapohjaisia komponetteja.

LÄHTEET JA TUOTETUT AINEISTOT

ReactJS Component [verkkosivu]. [viitattu 27.04.2019]
https://reactjs.org/docs/react-component.html
Material UI [verkkosivu]. [viitattu 27.04.2019]
https://material-ui.com/
Redux API [verkkosivu]. [viitattu 27.04.2019]
https://redux.js.org/api/api-reference
Webpack Documentation [verkkosivu]. [viitattu 27.04.2019]
https://webpack.js.org/concepts
Doctrine ORM [verkkosivu]. [viitattu 27.04.2019]
https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/index.html