Morayetan Lawrence

# Nykyaikaisen web-järjestelmän arkkitehtuuri

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

01.06.2016

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Morayetan Lawrence<br>Nykyaikaisen web-järjestelmän arkkitehtuuri<br><br>36 sivua<br>01.06.2016 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Tietotekniikka |
| Suuntautumisvaihtoehto | Ohjelmistotekniikka |
| Ohjaaja(t) | Lehtori Juha Kämäri |

Työn tavoitteena oli tutkia ja toteuttaa nykyisellä webteknologialla verkkosovellus. Tarkoituksena oli tutkia syvällisemmin, mihin suuntaan verkkopalveluiden kehitys on mennyt.

Tässä työssä tavoitteena oli toteuttaa sovellus, joka helpottaa työvuorojen hallintaa ja varausta. Tarkoitus oli käyttää asiakaspuolella AngularJS sovelluskehystä ja REST-rajapinnan toteutuksessa CakePHP-sovelluskehystä toteutustekniikoina. Työssä käydään läpi MVC-arkkitehtuurin toimintaperiaatteet. Sen jälkeen perehdytään AngularJS ja CakePHP-sovelluskehyksiin.

Insinöörityössä keskitytään käytettyihin teknikoihin sekä verkkopalveluiden tietoturvallisuuksiin. Lopussa käydään toteutusvaiheet läpi.

| | |
|---|---|
| Avainsanat | AngularJS, MVC, REST API, CakePHP |

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Morayetan Lawrence<br>Modern Web Systems Architecture<br><br>36 pages<br>01 June 2016 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Specialisation option | Software Engineering |
| Instructor(s) | Juha Kämäri, Senior Lecturer |

The aim of this thesis is to study and implement web application with modern web technology. The purpose was to study deeply what direction web services development has gone.

In this work, the objective was to implement application, which makes it easy for shifts management and reservation. The aim was to use AngularJS framework for the client side and CakePHP framework as implementation techniques. The thesis covers the basics of MVC architecture and used programming languages and frameworks.

The thesis focuses on the techniques used and web information security. At the end, we'll go through the stages of implementation.

| | |
|---|---|
| Keywords | AngularJS, MVC, REST API, CakePHP |

## Sisällys

## Lyhenteet

| | |
|---|---|
| REST | Representational State Transfer on arkkitehtuurimalli ohjelmointirajapinnan toteuttamiselle. |
| LAMP | Kokoelma avoin lähdekoodi -ohjelmia, jolla voidaan toteuttaa web-sovelluksia. |
| MVC | Model-View-Controller on arkkitehtuurimalli, joka erottaa sovelluksen kolmeen eri komponenttiin. |
| CakePHP | CakePHP on sovelluskehys verkkosovellukselle. |
| AngularJS | JavaScript-sovelluskehys käyttöliittymälle. |
| HTML | Hypertext Markup Language on merkintäkieli, jolla tehdään kotisivuja ja erilaisia verkkosovelluksia. |
| PHP | Hypertext Preprocessor on ohjelmointikieli, jota käytetään web-palvelin ympäristöllä. |
| CSRF | Cross-site Request Forgery on tietoturvahaavoittuvuus, jossa hyökkääjä lähettää luvattomia komentoja käyttäjälle, johon web-sovellus luottaa. |
| SQL | Structured Query Language on kieli, jolla ohjemoidaan relaatiotietokantajärjestelmää. |
| XSS | Cross-site Scripting on tietoturvahaavoittuvuus, jossa hyökkääjä pääse laittamaan haitallista koodia käyttöliittymään. |
| CRUD | Perustoiminnot sovelluksissa Create, Remove, Update ja Delete. |
| ES6 | EcmaScript 6 on JavaScript-skriptikielen standardi. |
| NativeScript | NativeScript on avoimen lähdekoodin JavaScript-kehys, jolla voi rakentaa natiivipuhelinsovelluksen. |

| | |
|---|---|
| AJAX | Asynchronous JavaScript, jonka avulla sovelluksesta voidaan tehdä vuorovaikutteisempi. |
| URL | Uniform Resource Locator käytetään WWW-sivujen osoittamiseen. |
| DOM | Document Object Model on alustasta riippumaton rajapinta, jolla voi käsitellä HTML-elementtejä. |
| SOAP | Simple Object Access Protocol on XML:ään perustuva protokolla, jolla kommunikoidaan. |
| API | Application Programming Interface, jolla eri ohjelmat voivat pyytää ja vaihtaa tietoa. |
| XML | Extensible Markup Language käytetään tietojen tallentamiseen ja siirtämiseen. |
| HTTP | Hypertext Transfer Protocol on sovellusprotokolla, joka määrittelee viestin muotoa ja yhteyttä. |
| SMTP | Simple Mail Transfer Protocol on protokolla, jota yleisimmin käytetään sähköpostin lähettämiseen. |
| URI | Uniform Resource Identifier on merkkijono, jota käytetään resurssien löytämiseen. |
| HATEOAS | Hypertext As The Engine Of Application State on linkki, jota käytetään navigoimiseen REST-rajapinnalla. |
| CORS | Cross-origin resource sharing on mekanismi, joka antaa toisen domainin pyytää rajoitettuja resursseja. |
| ECMA | European Computer Manufacturers Association on toisella nimellä JavaScript. |
| HTTPS | Hypertext Transfer Protocol Secure on sovellusprotokolla, joka salaa yhteyden määritetylle viestille. |

| | |
|---|---|
| MySQL | MySQL on avoin lähdekoodin relaatiotietokanta. |
| Composer | Paketinhallinta PHP-verkkosovelluksille. |

# 1 Johdanto

Rajapinnan kehittäjien, jotka tarjoavat rajapintaa kehittäjille, on tärkeä pysyä uusimpien ohjelmointi-innovaatioiden mukana. Kuten monet muutkin API-kehittäjät ovat valinneet tämän lähestymistavan, yksinkertaistetun tietojen validointirajapinnan REST. REST-rajapinnalla tietoja tuotetaan ja kulutetaan. REST on selkeää, helposti ymmärrettävää ja sen toteuttaminen on helppoa.

Tämä insinöörityö käsittelee modernia web-järjestelmäarkkitehtuuria. Työn tarkoituksena on perehtyä web-ohjelmointikehyksiin, REST-arkkitehtuurimalliin sekä tutkia, millainen hyöty niistä on verkkosovelluksen kehityksessä. Työssä toteutetaan työvuorosovellus hyödyntämällä LAMP:ia. Sovellus helpottaa vuorojen lisäämisen ja varaamisen esimerkiksi pien yrityksille.

Tässä työssä tutustaan MVC-malliin ja miten se toimii verkkosovelluksessa. Sen jälkeen perehdytään lyhyesti CakePHP-sovelluskehyksen komponentteihin ja sen mukana tuleviin nimeämissääntöihin. Sen jälkeen käydään läpi AngularJS-sovelluskehystä ja sen toiminnallisuutta. Sitten perehdytään REST-arkkitehtuurimalliin ja sen rajoitteisiin. Lopuksi käydään läpi työvuorosovelluksen toteutus vaiheittain.

## 2 MVC-malli web-sovelluksessa

MVC on arkkitehtuurimalli, joka on monessa suosituissa ohjelmointikielissä. Domaini logiikka on toiminto, joka käsittelee tiedonvaihdon käyttöliittymän ja tietokannan välillä. Siksi pystyy muokkaamaan domainin logiikka. Tärkeintä suunnittelijoille on, että käyttöliittymä on erillinen. Tämä vähentää sekaannuksen ja yksinkertaistaa koko kehitysprosessin. Kun ymmärtää, miten MVC-malli toimii, niin PHP-kehyksestä tulee olemaan paljon helpompi ja selkeämpi käyttää. (1.)

MVC-lyhenne tulee:

- Malli (Model) hallinnoi dataa, logiikkaa ja sovelluksen sääntöjä.
- Näkymä (View), pitää huolen käyttöliittymän ulkoasusta.
- Ohjain (Controller) hyväksyy sisään tulevat datat ja käskyn tullessa se muuntaa tiedon mallille tai näkymälle.

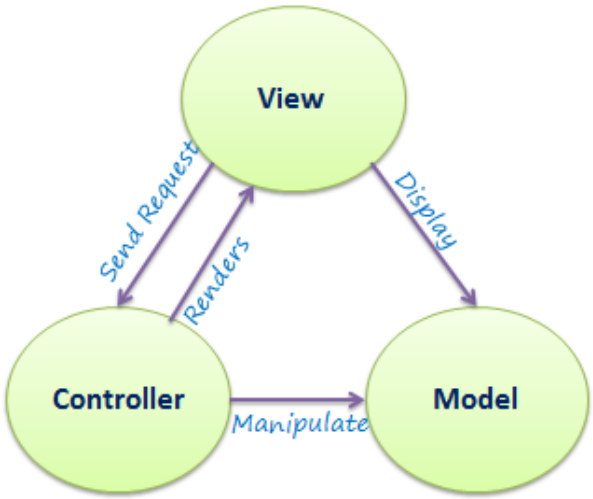
MVC-mallia on suunniteltu työpöytäsovellusten graafisiin käyttöliittymiin, mutta siitä on tullut myös erittäin suosittu web-sovelluksissa. (2.)

MVC-mallin tärkeimmät hyödyt ovat:

- nopeampi kehitysprosessi
    - MVC tukee nopeaa ja rinnakkaista kehitystä. Jos MVC-mallia käytetään verkkosovelluksen kehitykseen, niin yksi ohjelmoija voi työskennellä näkymässä ja toinen samaan aikaan ohjaimessa.
- kyky tarjota useita näkymiä
    - MVC-mallissa voi luoda useita näkymiä mallille. Lisäksi tässä menetelmässä koodin duplikaatteja on hyvin vähän, koska se erottaa datan logiikasta.
- tukee asynkronista tekniikkaa
    - MVC-arkkitehtuuria voidaan integroida JavaScript-sovelluskehyksen kanssa, joka mahdollistaa nopean asynkronisen sovelluksen kehittämisen.
- muutos ei vaikuta koko malliin
    - Uusien näkymien lisääminen on helppoa MVC-mallissa, koska mallin osa ei ole riippuvainen näkymästä.

- palauttaa datan ilman muotoilua.
    - Samat komponentit MVC-mallissa voidaan käyttää ja kutsua mistä tahansa käyttöliittymästä

(3.)



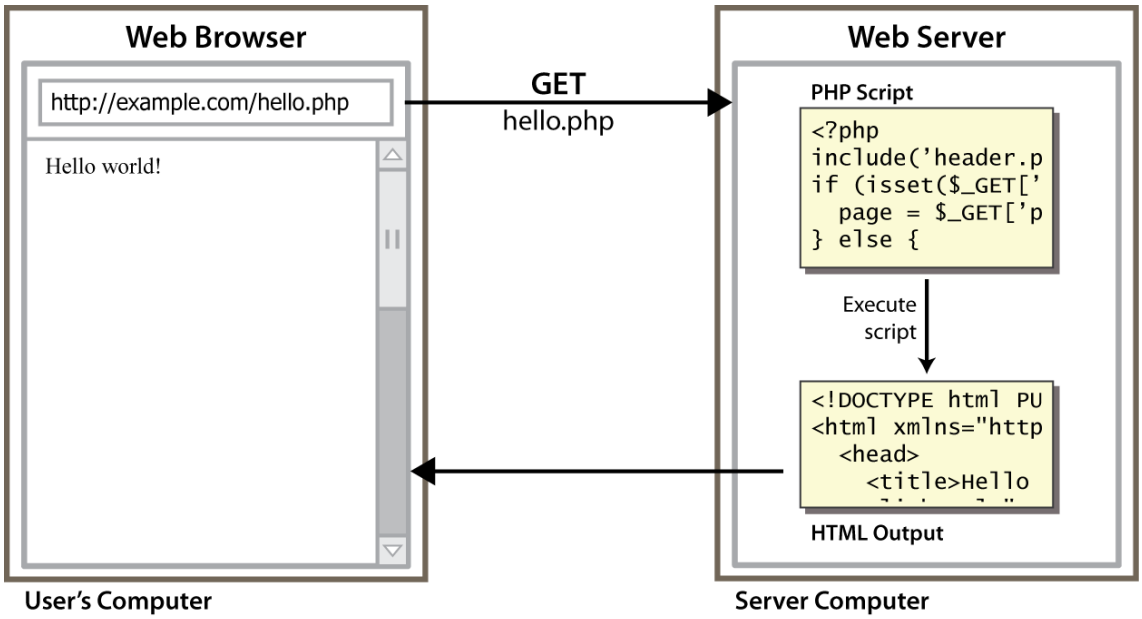

Kuva 1. Kaavio MVC-arkkitehtuurista. (4.)

Näkymässä näkyvät kaikki mallin datat. Tietojen päivittämisessä näkymä lähettää pyynnön ohjaimelle, joka manipuloi mallin datat ja päivittää sen näkymään. Mallissa on tarvittavat logiikat, jotka hoitavat tietojen käsittelyn.

# 3 Käytetyt tekniikat

## 3.1 CakePHP

PHP on laajasti käytetty avoin lähdeskriptikieli, jolla lähinnä keskitytään palvelinpuolen skriptaukseen. Sopii erityisesti web-kehitykseen ja voidaan upottaa HTML: ään. Se, mikä erottaa PHP:n käyttöliittymä Javascriptista, on koodi, jota suoritetaan palvelimen puolelta. PHP kehittää HTML-koodit ja lähettää sen käyttöliittymälle. Asiakas näkee koodin tuloksen, mutta ei pysty tietämään, millaista koodia on taustalla. (5.)




Kuva 2. Selain pyytää ”.php”-tiedoston, palvelin lukee sen. PHP suorittaa koodit, jotka ovat tiedoston sisällä ja lähettävät tuloksen verkon kautta takaisin selaimelle. (6.)

CakePHP on avoimen lähdekoodin web-sovelluskehys, joka seuraa MVC-lähestymistapaa ja on kirjoitettu PHP:lla. Se tekee web-sovelluksen kehityksestä yksinkertaisen, nopean ja vähentää koodien kirjoittamista. Sen mukana tulee sisäänrakennettuja työkaluja ja niiden avulla pystyy suojautumaan CSRF-hyökkäyksiltä, SQL-injektiolta, XSS ja tarkistamaan syötteitä. Parasta CakePHP:ssa on se, että se tarjoaa joustavan tietokantojen käyttö kerroksen ja koodigeneraattorilla tekemään prototyyppejä. (7.)

### 3.1.1 Nimeämissäännöt

CakePHP tarjoaa perusjärjestelmällisen rakenteen, joka kattaa luokan nimiä, tiedoston nimiä, tietokannan taulukoiden nimet ja muut yleissäännöt. Sääntöjen oppiminen kestää jonkin aikaa. (7.) CakePHP-osien nimeäminen toimii seuraavalla tavalla:

- Tietokannassa
    - Taulujen nimet CakePHP:ssa ovat monikossa ja alleviivattuja.
    - Taustalla olevien taulujen nimet olisi people, big_people ja really_big_people.
- Mallissa ja ohjaimessa
    - Nimet ovat yksittäisiä tai sitten CamelCase.

    - Nimeämissääntöjen esimerkkeihin kuuluvat Person, BigPerson, Really-BigPerson.

- Ohjaimessa
    - Ohjainten luokkien nimet ovat monikossa, CameCasella ja loppuun lisätty Controller.
    - Esimerkiksi PeopleController-luokka, joka on nimetty people-tietokanta taulun nimen mukaan.
- Näkymässä
    - Mallitiedosto on nimetty ohjaimien metodien mukaan alleviivatussa muodossa.
    - GetReady()-funktio PeopleController-luokassa etsii näkymämallia tiedostoapolusta /app/View/People/get_ready.ctp. (8.)

Seuraamalla yleissääntöjä voi välttää tarpeettomia asetelmia ja tehdä yhtenäisen sovellusstruktuurin, joka tekee eri projekteissa työskentelystä helpompaa. (7.)

### 3.1.2 Komponentit

CakePHP:n mukana tulee joukko ydinkomponentteja, jotka voi käyttää avuksi yleisten tehtävien suorittamisessa. Siitä on myös mahdollista luoda oma komponentti. Jos löytää itsensä kopiomassa ja liittämässä samankaltaisia asiota ohjaimien kesken, kannattaa luoda komponentti. CakePHP:ssa on sisäänrakennettu seuraavanlaiset komponentit:

- Authentication
    - Käyttäjien todennuksen ja valtuutuksen hallintaan.
- Cookie
    - Evästeiden hallintaan
- Cross Site Request Forgery
    - Suojaa CSRF-hyökkäyksiltä.
- Flash
    - Tarjoaa ilmoitusponnahduksen käyttöliittymälle.
- Security

- Käsittelee perusturvatoimia, kuten tarjoaa menetelmiä tiivisteille ja salauksille.

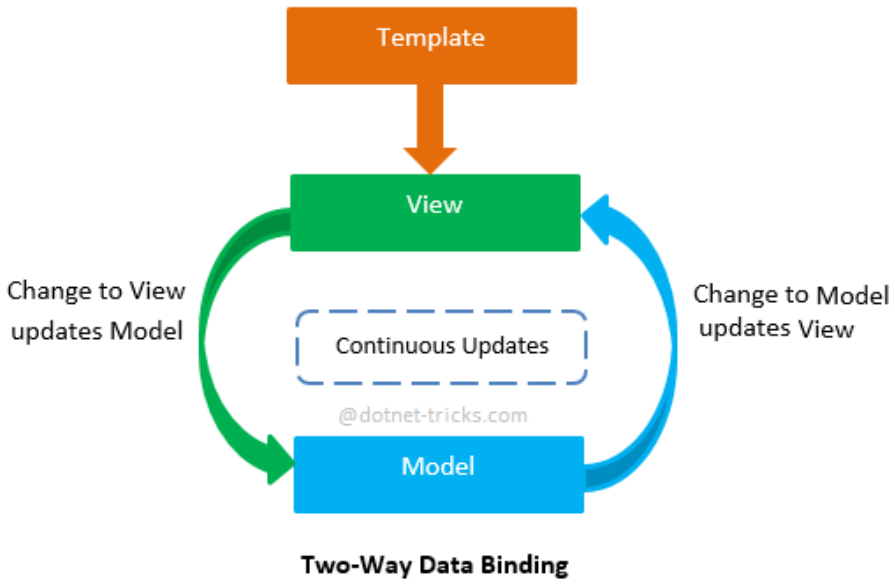
- Pagination
  - Pitää huolen tietojen näyttämisen ruudulla.
- Request Handling
  - Käsittelee palvelimelle tulevia pyyntöjä.

(9.)

## 3.2 AngularJS

### 3.2.1 AngularJS yleisesti

Angular on täydellinen JavaScript-pohjainen avoimen lähdekoodin front-end web-sovelluskehys, jota Google pääosin ylläpitää. (10.) Se on rakenteellinen kehys dynaamisille web-sovelluksille. Se antaa mahdollisuuden käyttää HTML:ää mallikielenä ja antaa mahdollisuuden laajentaa HTML-syntaksia. Nämä kaikki tapahtuvat selaimen sisällä, joten se soveltuu hyvin minkä tahansa web-palvelimen kanssa.




Kuva 3. Kaavio kaksisuuntaisesta datan yhdistyksestä. (11.)

Datan yhdistys sovelluskehyksessä synkronoi mallin ja näkymän. Kun tiedot mallissa tai näkymässä muuttuvat, niin tiedot päivittyvät kumpaankin. Tämä prosessi tapahtuu saman tien automaattisesti, mikä varmistaa mallin ja näkymän päivittymistä.

Angular yksinkertaistaa sovelluksenkehityksen tuomalla korkean abstraktion kehittäjälle. Kuten kaikki abstraktiot niissä on omat puolensa. Toisin sanoen jokainen sovellus ei sovi Angularille. Angular on rakennettu CRUD (Create, Remove, Update & Delete)-sovelluksia varten, jota moni web-sovelluksista edustaa. (12.)

### 3.2.2 Ero AngularJS:n ja Angular 2:n välillä

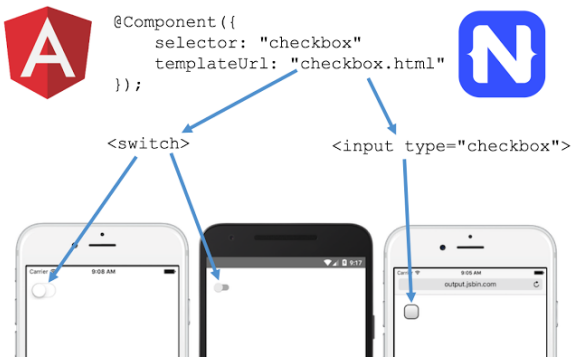
Angularista on kaksi versiota: AngularJS 1 ja Angular 2. Angular 2 julkistettiin ng-Europe konferenssissa 22-23. syyskuuta 2014. Muutos oli varsin suuri, se aiheutti väittelyä kehittäjien keskuudessa. Vakaa versio julkaistiin 14. syyskuuta 2016. Angular 2 ei ole päivitysversio, vaan se on uusi versio. (10.) Tästä huolimatta AngularJS 1-versiossa pystyy hyödyntämään ES6-omanisuuksia. Angular 2 on suunniteltu hyödyntämään näitä tekniikoita, jotka tekevät siitä, nopean, tehokkaan, parhaan ja alustasta riippumattoman sovelluskehyksen. (13.) Ensisijaiset erot Angular 2:ssa on:

- Modulaarisuus - ydintoiminnot ovat siirtyneet moduuleihin, joka keventää ja nopeuttaa ytimen
- Se vähentää tarvetta selvitellä selaimen yhteensopivuutta
- Microsoftin TypeScript-kieli, jota suositellaan koodikielenä Angular 2:ssa
- Yksinkertaisempi reititys
- Ohjaimia (Controller) ja $scope on paikattu komponenteilla ja direktiiveillä.

(10.)

Tästä huolimatta Angular 1-versiota vielä ylläpidetään. Useimmilla yrityksillä on järjestelmä, joka käyttää Angular 1:tä ja luultavasti tulee olemaan pitkään. Tämä tarkoittaa, että Angular 1 tulee tarvitsemaan ylläpitoa ja jopa päivitystä Angular 2:een. (14.)

Angular 1.x on luotu responsiiviseksi ja kaksisuuntaisen datan yhdistämistä varten, mutta siinä ei ole mobiilitukea. Angular 2 sisältää mahdollisuuden toteuttaa mobiiliorientoitua arkkitehtia. On olemassa JavaScript-kirjastoja esim. NativeScript, joka nopeuttaa Angular 2 -mobiilikehitystä. (15.)




Kuva 4. NativeScriptin mobiilituki, joka asettaa eri käyttöjärjestelmäkohtaiset painikkeet. (16.)

NativeScript on avoimen lähdekoodin JavaScript-kehys, jolla voi rakentaa natiivipuhelinsovelluksia yhden koodin perustalla. NativeScript toimii hyödyntämällä JavaScript-virtuaalikoneita. Kun luodaan yhteys natiivikäyttöliittymä- ja JavaScript koodiin välille, hyödynnetään JavaScriptCore- ja V8-yhteyttä. (17.)

### 3.2.3 Scope

Scope on sitova osa HTML:n (näkymä) ja JavaScriptin (ohjain) välillä. Scope on olio, jonka mukana on saatavilla asetuksia ja metodeja. Se on saatavilla näkymässä ja ohjaimessa. AngularJS-sovellus koostuu

- näkymästä, joka on HTML.
- mallista, jossa on näkymälle saatavia tietoja
- ohjaimesta, joka on JavaScript-funktio, jolla voi lisätä/muokata/poistaa/ohjata datoja

Scope on sitten malli. (18.)

Jokainen AngularJS-sovellus pitää sisällään $rootScopen, joka on HTML:ään luotu Scope, mikä sisältää ng-app-direktiivin. $rootScope on saatavilla ympäri sovelluksen.

Jos paikallisella muuttujalla on sama nimi kuin $rootScopen muuttujalla, sovellus käyttää paikallista scopea. (18.)

### 3.2.4 Injektio

Riippuvuus-injektio on suunnittelumalli, joka käsittelee, miten komponentit saavat käsiinsä riippuvuudet. AngularJS-injektoriosajärjestelmä on vastuussa komponenttien luomisesta, riippuvuuksien ratkaisemisesta ja sen tarjoaminen muidenkin komponenttien pyytäessä. Riippuvuusinjektio on käytettävissä ympäri sovelluskehystä.

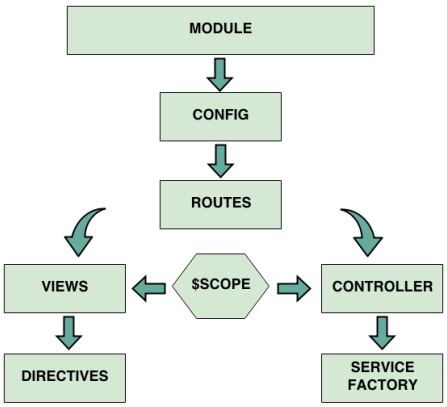
- Komponentit kuten palvelut, direktiivit, suodattimet ja animaatiot on määritelty injektoitavalla tehtaan metodilla tai alustajafunktiolla.
- Ohjaimet on määritelty alustajafunktiolla, joka sitä voi injektoida millä tahansa ”palvelu”-ja ”arvo”-komponenteilla.
- Run-metodi hyväksyy injektion funktioon ”palvelu”-, ”arvo”- ja ”vakio”-komponenttien riippuvuuksilla.
- Config-menetelmä hyväksyy injektion ”palveluntarjoaja”- ja ”vakio”-komponenttien riippuvuuksilla. (19.)

### 3.2.5 Moduuli ja komponentit

Moduuli on kokoelma palveluja, direktiivejä, ohjaimia, suodattimia ja kokoonpanotietoja. Moduulia käytetään injektion konfiguroimiseen. Useimmissa sovelluksessa on päämetodi, joka käynnistyy ja yhdistää sovelluksen eri osat. AngularJS-sovelluksessa ei ole päämetodia. (20.)

```
var app = angular.module('myApp', []);
```

Esimerkkikoodi 1. Moduuli sovellusta varten.




Kuva 5. Kaavio AngularJS-arkkitehtuurista. (21.)

Moduuleihin voidaan tarvittaessa luetella muita moduuliriippuvuuksia riippuen, mitä moduuleita halutaan ladata ennen päämoduulia. Toisin sanoen vaadittavat kokoonpanolohkot suoritetaan ennen moduulin kokoonpanolohkoa, joka edellyttää riippuvuuksien suorittamista. Sama pätee Run-lohkoille. Jokaista moduulia voidaan ladata vain kerran, vaikka moni moduuli on riippuvainen siitä. (20.)

Ohjaimet

Ohjain (Controller) on määritelty JavaScript-rakentajafunktiolla, joka tehostaa AngularJS Scopea. Komponentin tarkoituksena on määritellä scope-muuttujia ja tarvittaessa toteuttaa toiminnallisuudet. (15.)

```
app.controller('myCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
```

Esimerkkikoodi 2. AngularJS-ohjain, jossa toteutetaan toiminnallisuudet.

Palvelut

AngularJS:n palvelut ovat korvattavia objekteja, jotka on kytketty toisiinsa käyttämällä riippuvuusinjektio. Palveluilla voi organisoida ja jakaa koodia ympäri sovelluksen. (23.) AngularJS tarjoaa erilaisia sisäänrakennettuja palveluita, joita ovat esimerkiksi

- $http pitää huolta pyynnön (AJAX) lähettämisestä ja vastaanottamisesta.

- URL hallitsee $locationin kautta. Tämä palvelu jäsentää URL-osoitteen selaimen osoiteriville ja sovelluksessa tekee URL osoitteesta käytettävän. Muutokset URL-osoitteessa näkyvät $location-palvelussa ja muutokset $location-palvelussa näkyy URL-osoitteessa. (24.)
- $filteriä käytetään datan muotoiluun tietojen näyttämistä varten. (25.)

AngularJS 1:ssä on yhteensä 30 sisäänrakennettua palvelua.

```
app.service('intToString', function() {
    this.convert = function (x) {
        return x.toString(16);
    }
});
```

Esimerkkikoodi 3. Palvelujen luominen ohjaimille.

Direktiivit

Direktiivi on merkitsijä DOM-elementeissä, jotka kertovat AngularJS:n HTML-kääntäjälle liittämään tietyn käyttäytymisen DOM-elementtiin tai jopa muuttavat DOM-elementin ja sen lapsen. AngularJS:n mukana tulee joukko sisäänrakennettuja direktiivejä kuten ngBind, ngModel ja ngClass. Aivan kuten luodaan kontrollereita ja palveluita, voi myös luoda direktiivejä. (26.)

```
app.directive("Direktiivi", function() {
    return {
        template : "<h1>Tehty direktiivillä!</h1>"
    };
});
```

Esimerkkikoodi 4. AngularJS-direktiivi, jolla voi liittää toiminnallisuuden DOM-elementtiin.

### 3.2.6 Paketinhallinta AngulariJS:n kanssa

AngulariJS:lle on paljon erillaisia lisäosia. Koodia ei tarvitse toistaa jatkuvasti sovelluksenkehyksen takia ja injektiolla saa liitettyä erilaisia lisäosia. AngularJS:lle löytyy paketinhallintasovelluksia, jotka nopeuttavat ja helpottavat sovelluskehitystä. Tämä tarkoittaa sitä, että ei tarvitse keksiä pyörää uudestaan.

NPM on paketinhallintaohjelma JavaScriptille, joka antaa kehittäjille mahdollisuuden jakaa ja käyttää uudelleen koodia. Selatessa sivua saattaa löytyä paketteja, jotka voivat auttaa sovelluksen kehityksessä. Paketteja asennetaan komentorivin kautta. On myös paljon paketteja, joissa on käytettävissä komentoriviltä ajettavia komentoja. (27.)

# 4 Verkkopalvelu

## 4.1 Palvelut ennen REST-määritelmää

Termi REST otettiin käyttöön ja määriteltiin vuonna 2000 Roy Fieldingin väitöskirjassa. (28.) Ennen sitä useimmat ohjelmistokehittäjät käyttivät SOAP-Simple Object Access-protokollaan integroidakseen rajapintaa. Sana ”Simple” on osa tätä lyhennettä eikä sitä pidä ottaa kirjaimellisesti.

SOAP ( Simple Object Access Protocol) on protokollamäärittely jäsenneltyjen tietojen vaihtamiseen verkkopalveluiden toteuttamisessa. Sen tarkoituksena on mahdollistaa laajentamista, neutraalisuutta ja riippumattomuutta. Se käyttää XML-datanvaihtomuotoa muotoa. SOAP nojautuu useimmiten Hypertext Transfare Protocoliin (HTTP) tai Simple Mail Transfare Protocoliin (SMTP) viestiä käsitellessä/lähettäessä. (29.)

SOAP on suunniteltu ennen mobiilisovelluksia, joten siinä on huonoja puolia kuten:

- Palveluiden uudelleen käyttö modernismissa web-sovelluksissa, jotka käyttävät AJAX:ia ja mukailtuja RESTful-palveluita hyödyntäen JSON-datatyyppiä. Mahdollisuudet uudelleenkäyttöön rajoittuu, jos mobiilisovellukset käyttävät RESTin sijaan SOAP:ia.
- Palveluiden vaihtaminen tarkoittaa monimutkaista koodin vaihtoa asiakkaan puolelta. Asiakkaan puolen ollessaan web-palvelimessa muokkaaminen ei ole hankalaa. Mobiilisovelluksessa asiakkaan puolen päivittäminen leviää helposti käsiin. (30.)

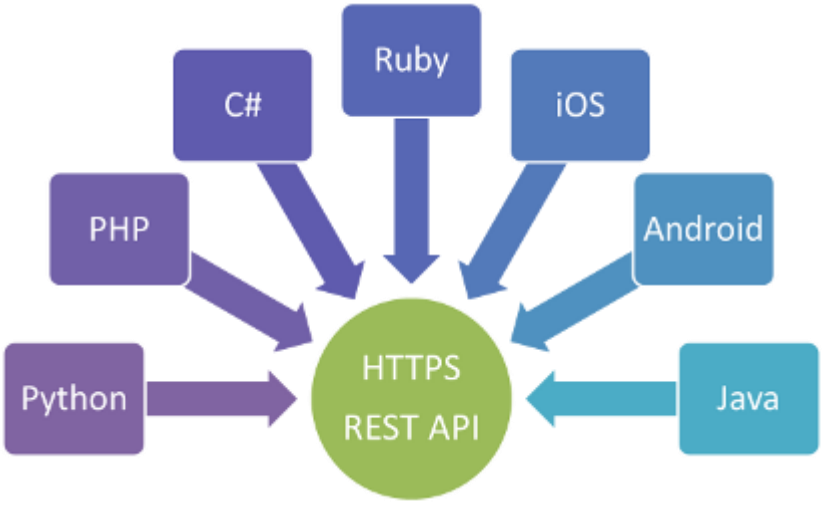
REST-verkkopalvelu on suositumpi kuin SOAP, koska sitä on helpompi toteuttaa ja se on halvempi.

## 4.2 REST yleisesti

REST (Representional state transfer) on yksinkertainen verkkopalvelu, joka on toteutettu käyttämällä http- ja REST- periaatteita. Sen resurssi koostuu neljästä määritetystä näkökohdista:

- URL-verkkopalvelun pohja, kuten http://esimerkki.fi/resurssit/.
- Internetmediatyyppien datat, jotka verkkopalvelut tukevat. Useimmiten JSON tai XML. Se voi olla mikä tahansa muu pätevä Internet mediatyppi.
- Joukko toimintoja, jotka verkkopalvelin tukee käyttämällä HTTP:n menetelmät (POST, GET, PUT tai DELETE).
- Sillä on kyky tallentaa välimuistiin ja hyödyntää kerrostettua järjestelmää.

REST-arkkitehtuurit sisältävät käyttöliittymät ja palvelimet. Käyttöliittymät lähettävät pyyntöjä palvelimille. Palvelin käsittelee pyynnön ja palauttaa asianmukaisen vastauksen. Pyyntö ja vastaus on rakennettu siirron edustusresurssin (transfer of representations of resources) ympärille. Resurssia voi olla olennaisesti mikä tahansa johdonmukainen ja mielekäs käsite, jota voi lähettää. Edustusresurssi on tyypillisesti dokumentti, joka tallentaa nykyisen tai aiotun tilan resurssin. (31.)




Kuva 6. REST on alustasta riippumaton verkkopalvelu. Asiakkaan puolen ohjelmat voidaan ohjelmoida erilaisilla ohjelmointikielillä, jotka tukevat http- tai HTTPS- pyyntöjen tekemisen. (32.)

Taulukko 1. RESTin CRUD operaatiot.

| Operaatio | SQL kielessä | HTTP/REST |
|---|---|---|
| Luo (Create) | INSERT | POST |
| Lue/Hae (Read/Retrieve) | SELECT | GET |
| Päivitä (Update) | UPDATE | PUT |
| Poista/Tuhoa (Delete/Destroy) | DELETE | DELETE |

Toteutuksessa REST:iä voidaan katsoa CRUD (Create, Read, Update ja Delete) -suunniteltuna arkkitehtuurina. URL tunnistaa resurssit, jotka kuvaavat minkä operaation haluaa suorittaa ja pyynnön tyyppi kuvaa operaation tyypin. Ohjelmistoa ei voida pitää täydellisenä ilman näitä neljää operaatiota, koska nämä operaatiot ovat perusteellisia. (31.)

Taulukko 2. HTTP-pyynnöt REST-tyylillä.

| Metodi | URL muoto | Tehtävä |
|---|---|---|
| HEAD | /kalenteri/(id)/ | Tarkistaa kalenterin olemassa oloa tunnuksen avulla. |
| HEAD | /kalenteri/(id)/liitteet | Laskee kalenterin liitteiden määrä. |
| GET | /kalenteri/(id)/ | Hakee kaikki tiedot kalenterista tunnuksen avulla. |
| GET | /kalenteri/(id)/liitteet | Listaa kaikki liitännäiset, jotka kuuluvat tiettyyn kalenteriin. |
| POST | /kalenteri/ | Luo uuden kalenterin ja palautta kalenterin tunnuksen(ID). |
| PUT | /kalenteri/(id)/ | Asettaa kalenterin ominaisuudet ja objekti tunnistetaan tunnuksella (ID). |
| DELETE | /kalenteri/(id)/ | Poistaa kalenterin tunnuksen perusteella. |

Taulukossa 2 on sisäänrakennettu CakePHP-sovelluksen URL-reititys. Pyynnön metodin ja URL-verkkopalvelupohjan perusteella REST API lähettää vastauksia takaisin asiakkaan sovellukseen.

### 4.2.1 Yhtenäinen käyttöliittymä

Yhtenäinen käyttöliittymän rajoite on olennainen REST-palvelun suunnittelussa. Yhtenäinen käyttöliittymä yksinkertaistaa ja erottaa arkkitehtuurin, mikä mahdollistaa jokaisen osan kehittymistä itsenäisesti. Neljä rajoitusta yhtenäiseen rajapintaan ovat:

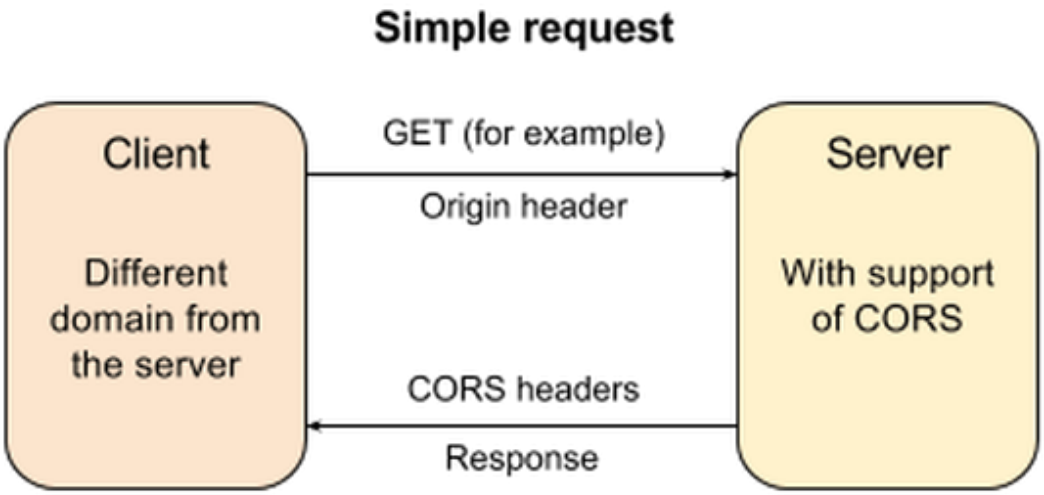
- Yksittäiset resurssit tunnistetaan pyynnöt esimerkiksi käyttämällä URI:a web-pohjaisissa REST-järjestelmissä. Resurssit on itse käsitteellisesti eroteltu edustuksestaan ja palautetaan takaisin käyttöliittymään.
- Kun asiakas pitää edustuksen resurssia mukaan lukien metadata, sillä on riittävästi tietoa muokata tai poistaa datan.
- Kukin viesti sisältää riittävästi tietoa kuvaamaan, miten käsitellään viestiä.
- Hypermedia as the engine of application state (HATEOAS), jolla REST-asiakas pitäisi pystyä käyttämään palvelimen tarjoamia linkkejä dynaamisesti löytää kaikki käytettävissä toimet ja resursseilla. (34.)

### 4.2.2 Tilattomuus

Asiakas-palvelin-viestintää rajoittaa se, että asiakastietoja ei tallenneta palvelimeen pyyntöjen aikana. Käyttöliittymä pitää sisällään tarvittavat tiedot ja istunnon tilan. Istunnon tila voidaan siirtää palvelimelta toiselle kuten tietokanta, jotta pystyisi säilyttämään pysyvän tilan istunnon ajaksi ja mahdollistamaan todennuksen. Asiakas lähettää pyynnöt, kun se on valmis siirtymään toiseen tilaan. Jokainen edustettava tila sisältää linkkejä, jotka voidaan käyttää uuden tilasiirron aloittamisessa. (2.)

### 4.2.3 Palvelimen ja asiakkaan erottelu

Käyttöliittymän ja palvelimenpuolen erottelu on osa client-server-periaatteita. Siirrettävyys parantuu erottamalla näitä kahta. Tämä mahdollistaa käyttöliittymän kehityksen eri alustoille ja parantaa skaalautuvuutta yksinkertaistamalla palvelimen komponentteja. Erottelu mahdollistaa komponenttien itsenäistä kehittämistä. Palvelimen ja käyttöliittymän pitäminen erillisessä domainissa vaatii CORS-tuen. (34.)




Kuva 7. CORS:in toiminta eri pyynnöissä.

Cross Origin Resource Sharing (CORS) sallii web-sovellusten kommunikoinnin eri domainilta. CORS käytetään monissa eri paikoissa ja tapauksissa. Web-kehityksessä se usein tulee tarpeen, kun sovelluksen käyttöliittymä ja palvelin erotetaan. CORS-mekanismi on pääosin toteutettu palvelimeen. Se vaikuttaa myös käyttöliittymän puolelta, jos headerit puuttuvat vastauksista. (35.)

### 4.2.4 Välimuistiin tallentaminen

Vastausviestit palvelimelta käyttöliittymään on merkitty joko välimuistiin tallennettavaksi tai tallentamattomaksi tiedoksi. Näin kuluttaja ja/tai palvelin voi tallentaa ne uudelleenkäytettäväksi. Välimuistirajoitus perustuu Client-Serveriin ja tilattomuuteen, mikä vaatii vastauksien luokittelemisen tallennettavan tai tallentamattomana dataksi. (34.)

Pyynnöt menevät välimuistikomponentin läpi, mikä voi käyttää edellisiä vastauksia osittain tai poistaa kokonaan verkon yli olevaa vuorovaikutusta. Tämä eliminointimuoto voi parantaa tehokkuutta, skaalautuvuutta ja käyttäjän havaittavissa olevia suorituskykyjä vähentämällä keskimääräisen viiveen vuorovaikutuksessa. (34.)

### 4.2.5 Kerrostettu järjestelmä

Asiakas ei voi kertoa sitä, onko yhteyksissä suoraan palvelimeen vai onko välittäjä välissä. Välittäjäpalvelimet saattavat parantaa järjestelmän skaalautuvuutta mahdollistamalla kuormituksen tasapainottamista ja tarjoamalla yhteisiä välimuisteja. Ne voivat myös valvoa turvallisuuskäytäntöjä. (34.)

## 4.3 JSON

JavaScript Object Notation eli JSON on kevyt datanvaihtomuoto. Se on helppo ihmisille lukea ja kirjoittaa. Myös koneille se on helppo jäsentää ja generoida. JSON perustuu JavaScript-ohjelmointikieleen ja Standard ECMA-262 3rd Editioniin – December 1999 osajoukoista. JSON:in tekstimuoto on täysin kieliriippumaton. Sopivuussäännöt ovat tuttuja C-kielten ohjelmoijille, mihin sisältyvät C, C++, Java, JavaScript, Python, Perl jne. Nämä asetukset tekevät JSON:ista ideaalisen datanvaihto kielen.

JSON on rakennettu kahteen struktuuriin:

- Kokoelma nimi/avain pareja. Monissa ohjelmointikielissä tämä pidetään objektina, kirjauksena, structina, sanakirjana, hash tablina, keyed lista tai taulukkona.
- Lista järjestettyjä arvoja. Useimmissa kielissä tämä pidetään joukkona, vektorina, listana tai sekvenssinä.

Nämä ovat yleisiä datamuotoja. Kaikki modernit ohjelmointikielet virtuaalisesti tukevat näitä tavalla tai toisella. On järkevää, että datan muoto, mikä on vaihdettavissa ohjelmointikielten kanssa, myös perustuu näihin rakenteisiin.

```
{ ”nimi” : arvo , ”nimi2” : arvo2 , ”nimi3” : arvo3 }
```

Esimerkkikoodi 5. JSON-rivistä, joka on kokoelma järjestettyjä nimi/arvopareja.

Rivi alkaa ”{” vasemmalla aaltosululla ja loppuu ”}” oikealla aaltosululla. Jokaisen nimen jälkeen tulee ”:” kaksoispiste ja nimi/arvo parit erotetaan ”,” pilkuilla. (36.)

```
{
    ”nimi_lista”: [
```

```
            { ”etunimi” : arvo , ”sukunimi” : arvo },
            { ”etunimi” : arvo , ”sukunimi” : arvo },
            { ”etunimi” : arvo , ”sukunimi” : arvo }
        ]
    }
```

Esimerkkikoodi 6. JSON-olio, jonka sisällä on useita rivejä. Tämä kokoelma sisältää järjestämättömiä nimi/arvo pareja.

Taulukko 3. JSON-formaatin tukemat datatyypit.

| **Tyyppi** | **Kuvaus** |
|---|---|
| Numero | tupla (double) tarkkuuden liukuluku JavaScript muodossa |
| Merkkijono (String) | kaksois lainatun Unicode takakenoviiva merkinnön kanssa |
| Boolean | tosi (true) tai epätosi (false) |
| Rivi (Array) | järjestetty sarja arvoja |
| Objekti | kokoelma järjestämättömiä avain ja arvo pareja |
| Välilyönti (White space) | voidaan käyttää minkä tahansa merkin välissä |
| nolla (null) | tyhjä |

(36.)

# 5 Tietoturva

## 5.1 Istunnon hallitseminen ja todennus

Istunto on jakso verkon HTTP-pyynnön (request) ja vastauksen (response) transaktioille, jotka liittyvät samaan käyttäjään. Modernit ja monimutkaiset verkkosovellukset edellyttävät tietojen säilyttämisen tai tietoa jokaisen käyttäjän tilasta pyyntöjen ajaksi. Siksi istunnot tarjoavat mahdollisuuden luoda muuttujia kuten käyttöoikeudet ja lokalisointiasetukset. Se sovelletaan jokaiseen vuorovaikutukseen, mitä käyttäjällä on verkkosovelluksen kanssa istunnon ajaksi.

Todennetun istunnon tunnus tilapäisesti vastaa vahvinta autentikoinnin menetelmää, jota sovellus käyttää. Niitä ovat käyttäjätunnus ja salasana, kertasalasanat, asiakaspohjainen digitaali sertifikaatti, älykortit tai biometriset tunnisteet (kuten sormenjäljen tai silmän verkkokalvo). (37.)

REST-rajapinnoissa ei ole istunnon hallintaa, koska se on tilaton. Tilattomuudella meinataan sitä, että palvelin ei tallenna käyttäjän tilaa eli istuntoa palvelimen puolelta. Asiakkaan sovellus hoitaa istunnon hallinnan. Todennus tapahtuu aina pyyntöjen yhteydessä esimerkiksi silloin, kun Asiakas lähettää HTTPS-yhteydellä käyttäjä- ja salasana pyynnön yhteydessä, mikä on tallessa heidän sovelluksessaan (Salattuna) heti ensimmäisen kerran kirjautuessa. Tällä tavoin vältytään istunnon fiksaatiohyökkäyksestä. (38)

## 5.2 Valtuutus

Valtuutus on toiminto, jossa määritellään käyttöoikeuksia tiedonlähteille liittyen tietoturvaan, tietokoneen turvallisuuteen ja kulunvalvontaan. REST-käyttöliittymät yleensä käyttävät GET- (lukemiseen/tiedon hakemiseen), POST- (luomiseen), PUT- (päivittämiseen) ja DELETE- (tiedon poistamiseen) metodeja. Nämä kaikki eivät kuitenkaan ole käytettävissä jokaiselle resurssin kokoelmalle, käyttäjille tai toiminnoille. Kannattaa varmistaa, että saapuvalla HTTP-metodilla on voimassa oleva istuntotunnus/-API avain ja siihen liittyvää resurssi kokoelmaa, toiminto ja rekisteri. (38.)

## 5.3 Syötteiden tarkistus ja XSS-injektion ehkäiseminen

Tietojen tarkistuksen ideana on vähentää epämuodostuneiden datojen päästäminen järjestelmään. Syötteiden tarkistus ei ole ensisijainen tapa ennaltaehkäistä XSS- tai SQL-injektiota. JavaScript-syötteiden tarkistus pystyy helposti ohittamaan poistamalla käytöstä JavaScriptin selaimesta tai Web Proxylla. Kannattaa kuitenkin suorittaa syötteiden validointi asiakkaan ja myös palvelimen puolelta. (39.)

Cross-site Scripting (XSS) viittaa asiakaspuolen injektiokoodihyökkäykseen, jossa hyökkääjä voi suorittaa haitalliset skriptit kotisivuun tai web-sovellukseen. XSS on yksi yleisimmistä verkkosovelluksen haavoittuvuuksista ja tapahtuu silloin, kun verkkosovellus tulostaa vahvistamattomat tai koodaamattomat käyttäjäsyötteet.

Kaikki hallitut käyttäjätiedot pitää muuntaa silloin, kun palautetaan HTML-sivulle suoritettavaksi haitallisen datan estämiseksi. Esimerkiksi <script> olisi palautettava <script> . Näin pystymme välttämään XSS-hyökkäyksen, koska syötteen muoto on vaihdettu ja sitä ei pysty ajamaan skriptinä. (40.)

```
<?php
      $input = 'Merkki jono jossa on html tägi <h1> Moi </h1>';
      $encoded = htmlspecialchars($input);
      echo $encoded;
      // Tulostaa "Merkki jono jossa on html tägi &lt;h1&gt;
      Moi &lt;/h1&gt;"
?>
```

Esimerkkikoodi 7. Merkkien kääntäminen HTML-elementiksi PHP-kielellä.

## 5.4 Suojautuminen SQL-injektiosta

SQL-injektio on tekniikka, jossa hyökkääjä pystyy sujauttamaan SQL-komennon SQL-kyselyyn nettisivun kautta. SQL-injektion haavoittuvuus saattaa mahdollisesti vaikuttaa minkä tahansa sivustoon tai verkkosovellukseen, joka on SQL-pohjainen. Haavoittuvuus on yksi vanhimmista, yleisimmistä ja vaarallisimmista verkkosovelluksen haavoittuvuuksista. Hyödyntämällä SQL-injektion haavoittuvuutta oikeissa olosuhteissa hyökkääjä voi käyttää sitä ohittamaan verkkosovelluksen todennus- ja valtuutusmekanismia. Hyökkääjä pystyy hakea kaikki tiedot tietokannasta. Sen avulla hyökkääjä pystyy lisäämään, muokkaamaan ja poistamaan tietoa tietokannasta vaikuttaen myös tietojen eheyteen. (41.)

Valmistetulla määrittelyllä (Prepared Statements) parametrisoidut kyselyt pystyvät suojautumaan SQL-injektiosta. Parametrisoidut kyselyt pakottavat kehittäjän määrittelemään kaikki SQL-koodit ja sitten siirtämään kunkin parametrin kyselyyn myöhemmin. Tämä varmistaa sen, että hyökkääjä ei voi muuttaa kyselyn tarkoitusta, vaikka hyökkääjä on lisännyt SQL-komennon. Kuitenkin tallennettavat datat on tarkistettava. (42.)

```
<?php
    // Luodaan PDO-olio, jolla saamme yhteyden tietokantaan.
    // Tietokantaa käytetään luodun PDO-olion kautta.
```

```
    $db = new PDO('mysql:host=localhost;dbname=test',
    'kayttajatunnus','salasana');

    // Valmistellaan kyselyä
    $stmt = $db->prepare("UPDATE kayttaja SET nimi=? WHERE
id=?");
    // Suoritetaan kysely parametreineen
    $stmt->execute(array($name, $id));
?>
```

Esimerkkikoodi 8. Parametrisoitujen kyselyiden tekeminen PHP-ohjelmointikielellä.

Tallennetut proseduurit (Stored procedure) on myös hyvä vaihtoehto. Tietyissä talletetuissa proseduureissa on samanlainen vaikutus kuin parametrisoiduissa kyselyissä turvallisesti toteutettaessa. Ne vaativat kehittäjän kirjoittamaan SQL-määrittelyn parametrilla, jotka ovat automaattisesti parametroituja ellei kehittäjä tee jotain, mikä rikkoo normin. Ero valmistetuilla määrittelyillä (Prepared statements) ja tallennettujen proseduurien (Stored procedures) on, että SQL-koodi tallennetuille proseduureille on määritelty ja tallennettu tietokantaan. Tallennettu proseduuri kutsutaan sovelluksesta. (42.)

```
DELIMITER $$
CREATE PROCEDURE UpdateUserName(
 IN nimi VARCHAR(25),
 IN id INT)
BEGIN
 UPDATE kayttaja SET nimi = nimi WHERE id = id;
END$$
DELIMITER ;
```

Esimerkkikoodi 9. Proseduurien tallentaminen MySQL-tietokannassa.

Näiden lisäksi voi laatia lista hyväksyttyjä sanoja (whitelist validation) ja escaping, jolla voi muuntaa erikoismerkkejä turvallisiksi. (43.)

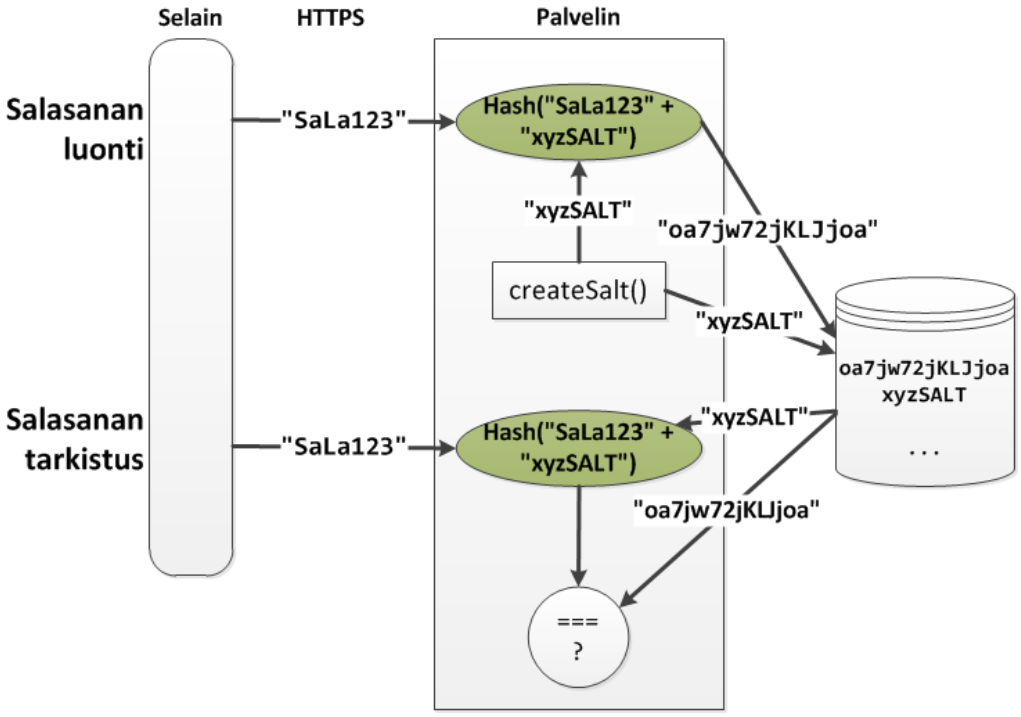
## 5.5 Salasanojen salaus ja talletus

Tietovuotojen vahingot voidaan vähentää salaamalla tiivisteellä (hashing) luottamuksellista dataa. Täten edes sovelluksen holhooja ei voi saada selville käyttäjän salasanaa. Salasanan tarkastuksessa generoidaan tiiviste annetusta salasanasta, jota verrataan tietokannassa olevaan tiiviisteeseen. Suolaa kannattaa laittaa jokaiseen salasanaan ennen kuin tiivistää, koska tiivistealgoritmi tuottaa samaan salasanaan aina saman tiivisteen. (44.)

Salasanan suolaus

- hidastaa sanakirjahyökkäyksen, jossa hyökkääjä käyttää tuhansia sanoja sisältävä lista ja lisää niihin numeroita
- hidastaa brute-force hyökkäyksiä, jossa hyökkääjä yrittää arvata monella eri salasanoilla toivoen onnea
- ennaltaehkäisee tiivistelmä törmäykset, joissa eri käyttäjillä on samat salasanat ja tiivisteet. Jos hyökkääjä selvittää yhden tiivisteen, hän automaattisesti tietää myös toisen käyttäjän salasanan. (45.)

Salasanat suolataan satunnaisilla arvoilla ja lisätään salasanan kanssa. Suolaukset tekevät yhdistelmästä pidemmän, jotta se pidentäisi brute-force-hyökkäyksen laskenta-aikaa.



Kuva 8. Salasana tallennetaan suolan kanssa tiivisteenä tietokantaan. Tarkistuksessa syötetään salasana suolan kanssa tiivistettynä ja sitä verrataan tietokannassa olevaan tiivisteeseen. (46.)

Kannattaa kuitenkin muistaa, että tämä ei estä salasanojen selvittämistä. Suolan eli satunnaismerkkien laittaminen salasanan kanssa pitkittää salasanan murtamisen. Myös oikean tiiviste algoritmin valitsemiseen on tärkeää, koska tietyt algoritmit on suunniteltu generoimaan tiivisteitä erittäin nopeasti ja tehokkaasti.

Taulukko 4. Tunnettujen tiivisteiden algoritmien Brute-force-nopeus laskemia seitsemänmerkkiselle salasanalle.

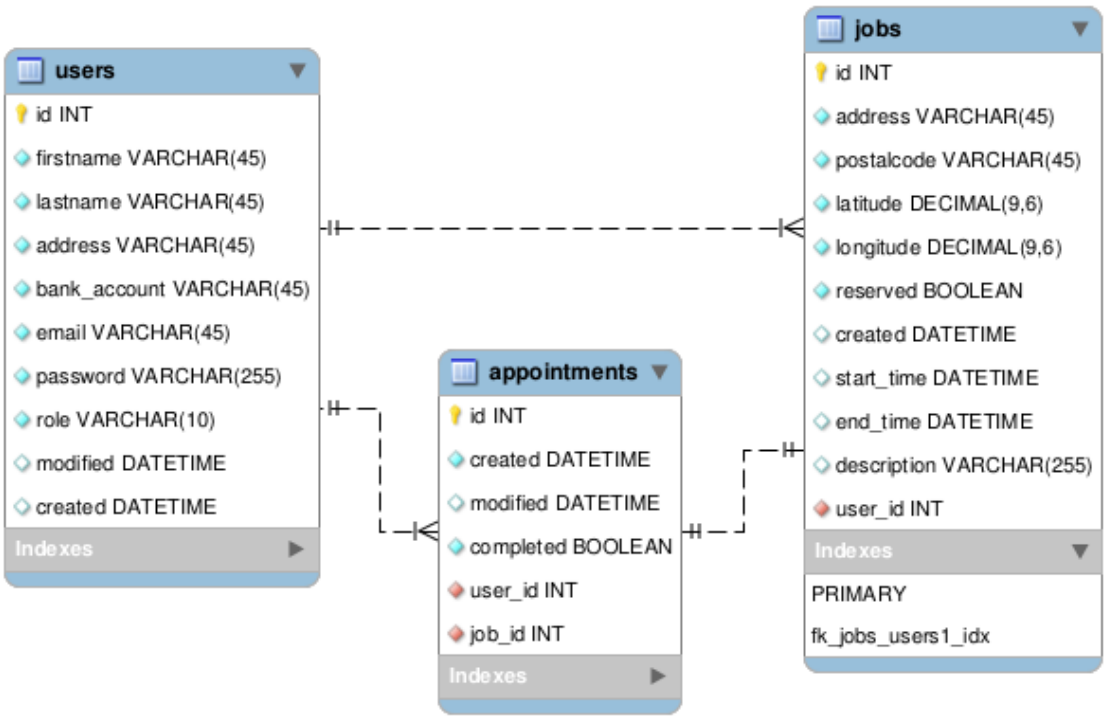
| **Algoritmi** | **Nopeus** | **Kesto** |
|---|---|---|
| MD4 | 9177K key/s | 18 päivää, 17 h, 12 min, 16 sekuntia |
| MD5 | 8745K key/s | 19 päivää, 14 h, 54 min, 38 sekuntia |
| SHA1 | 7107K key/s | 24 päivää, 4 h, 2 min, 27 sekuntia |
| Perinteinen DES | 4670K key/s | 36 päivää, 18 h, 43 min, 49 sekuntia |
| SHA256 | 2600K key/s | 66 päivää, 1 h, 31 min, 10 sekuntia |
| SHA512 | 2054K key/s | 83 päivää, 14 h, 59 min, 12 sekuntia |

Kunnollisen salasanan keksiminen on myös tärkeää. Sitä enemmän erikoismerkkejä salasana sisältää, sitä haastavampi se on saada selville. Taulukossa näkyy, kuinka monta päivää kestää, että hyökkääjä selvittää kaikki, mitä seitsemänmerkkiset salasanayhdistelmät brute-force-menetelmällä. Merkit (mixalpha-numeric-symbol14) ”abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$ %^&*()-_+=” ovat käytettävissä ja salasanayhdistelmiä on yhteensä 14840463832732 kappaletta. Tulosten saamiseen käytettiin 3.4GHz Intel Core i7-2600k. Vertailukriteerinä käytettiin john the ripper -nimistä ohjelmaa. (47.)

# 6 Toteutus

## 6.1 Tietokanta

Tietokantaa suunnitellaan CakePHP:n nimeämissääntöjen mukaan, jotta vältymme ylimääräisten asetusten tekemisestä. Viiteavaimien nimet on nimetty yksilölliseksi.




Kuva 9. Suunniteltu relaatiokaavio työvuorosovellusta varten.

Tauluja on yhteensä 3, niiden suhteet menevät seuraavanlaisesti:

- Users-taululla on yksi suhde moneen jobs-tauluun, koska työnantaja voi lisätä monta työvuoroa.

- Users-taululla on myös yksi suhde moneen appointments-tauluun, jotta työntekijät voisivat ottaa enemmän työvuoroja.

- Appointments-taululla on yksi suhde yhteen jobs-tauluun. Appointments-taululla pystymme katsomaan, kuinka monta työvuoroa työntekijällä on.

CakePHP on kätevä myös siitä, että se pystyy luomaan aikaleiman lisäämällä ’created’- ja ’modified’-nimiset muuttujat.

## 6.2 REST-rajapinta

Sama komento ajetaan kaikille tauluille. Kun lisää all argumentin komentoon, sillä saa kaikki mallit, näkymät ja ohjaimien koodit generoitua.

REST- tai RESTful-rajapinnan toteuttaminen CakePHP:lla on harvinaisen helppoa ja nopeaa. Nopein tapa saada REST pystyyn on lisätä muutama rivi koodia routes.php -nimiseen tiedostoon, joka sijaitsee app/Config -polulla. Reitittimen olio sisältää metodi-nimeltään mapResources(). Sillä asennetaan lukuisat oletusarvoiset reitit, jotta REST saa pääsyn ohjaimiin.

```
//app/Config/routes.php...
Router::parseExtensions(['json']);

Router::mapResources('users');
Router::mapResources('appointments');
Router::mapResources('jobs');
```

Esimerkkikoodi 10. Resurssien konfigurointi CakePHP-sovelluskehyksessä.

Kun reititin on asennettu, REST-rajanpinnan pitäisi toimia suoraan ilman muita asetuksia.

### 6.2.1 Alkuvalmistelut CakePHP:lle

CakePHP:n projektin luominen tapahtuu Composer-paketinhallintaohjelman avulla. Se generoi projektiin automaattisesti salasanojen suolan kirjoittamalla komentoriville komennon ” php composer.phar create-project --prefer-dist cakephp/app”.

Tietokannan yhdistäminen on melko suoraviivaista. Pitää vain korvata arvot Datasources.default-merkkijonosta, joka sijaitsee config/app.php-tiedostossa omilla asetuksilla. Esimerkki valmiista asetuksista näyttäisi tällaiselta:

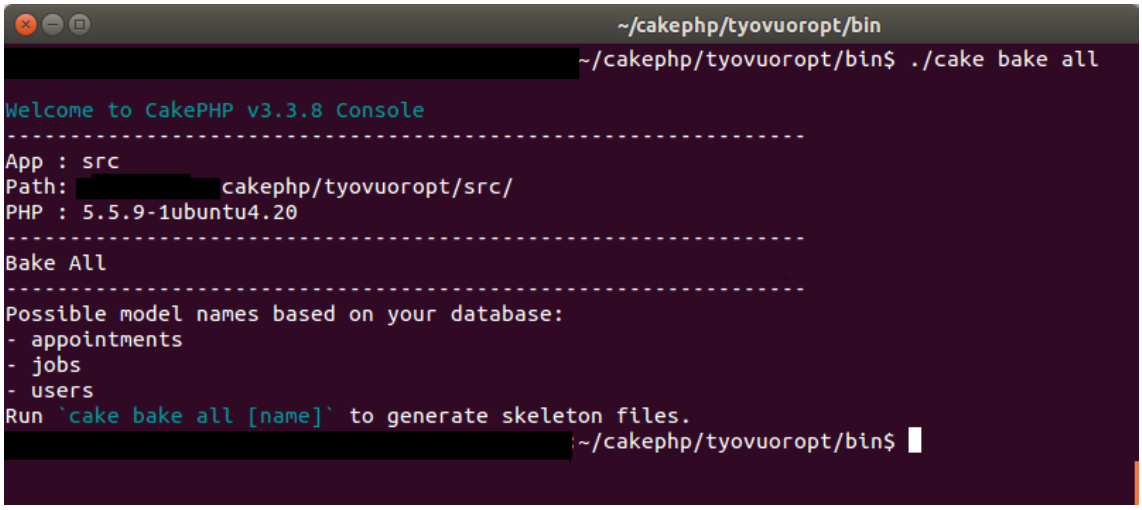
```
return [
   // lisää asetuksia ylhäällä.
   'Datasources' => [
```

```
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'käyttäjätunnus',
            'password' => 'salasana',
            'database' => 'tietokannan nimi',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Enemmän asetuksia.
];
```

Esimerkkikoodi 11. Tietokannan konfigurointi CakePHP:n kanssa.

Kun on tallentanut config/app.php tiedoston, pitäisi nähdä, että "CakePHP pystyy yhdistää tietokantaan" osiossa on valintamerkki.

CakePHP:lla on kätevä komentoriviasennusvelho nimeltään Bake, jolla pystyy generoimaan sivut. Bake-konsoli voi luoda minkä tahansa CakePHP-perusosan kuten: malleja, näkymiä ja ohjaimia. Bake kykenee luomaan täysin toimivan sovelluksen vain muutamassa minuutissa. Komentojen käyttö on tavallinen askel, kun sovellus on luotu. ”Cake bake all [taulun nimi]” komennolla näkee, mitä tauluja voi asentaa projektiin.




Kuva 10. CakePHP-komentorivivelho, jossa leivotaan tauluille sovellukseen vaadittavat tiedostot.

### 6.2.2 Todennus ja valtuutus

CakePHP:ssa on vaivatonta toteuttaa todennuksen ja valtuutuksen. AppController on yliluokka muille ohjaajille, jota aliluokat perivät. Se mahdollistaa komponenttien jakaamisen, siksi latasin yläluokkaan Auth-komponentin.

```
//src/Controller/AppController.php

public function initialize(){
    //lisää komponentteja

    $this->loadComponent('Auth', [
        'authorize' => ['Controller'],
        'authenticate' => [
            'Form' => [
                'fields' => ['username' => 'email',
                'password' => 'password']
            ]
        ],
        'loginAction' => [
            'controller' => 'users',
            'action' => 'login',
            '_ext'=>'json'
        ]
    ]);
}
```

Esimerkkikoodi 12. Todennus-komponentin lataus.

AppController.php-tiedostoon lisäsin Auth-komponentin, joka vastaa todennuksesta. Todennuksen ohjaimeksi valitsin UsersControllerin, koska se pitää sisällään käyttäjän sähköposti ja salasanan. Merkkijonossa oleva ‘authorize' => ['Controller'] antaa AppControllerille eli pääohjaimelle oikeuden määritellä oikeudet kaikille ohjaimille.

```
public function login(){
    if ($this->request->is('post')){
        $user = $this->Auth->identify();

        if ($user) {
            $this->Auth->setUser($user);
            $this->set('user', $user);
            $this->set('_serialize', ['user']);

        } else {
            throw new UnauthorizedException;
        }
   }else{
       throw new UnauthorizedException;
```

```
    }

}
```

Esimerkkikoodi 13. Todennuksen toteutus.

UsersController pitää sisällään login-metodin, joka vastaa sisäänkirjautumisesta.

```
public function isAuthorized($user){
    return true;
}
```

Esimerkkikoodi 14. Valtuutuksen toteutus ylemmässä luokassa.

Metodi isAuthorized() tarkistaa, onko todennettu käyttäjä valtuutettu. Pääohjaimen kaikki käyttäjät on valtuutettu suorittamaan kaikkien ohjaimien metodeja. Alempaan luokkaan määritellään tarkemmin roolien mukaan, mikä rooli on valtuutettu tekemään mitäkin.

```
public function beforeFilter(Event $event){
    $this->Auth->allow(['login']);
}
```

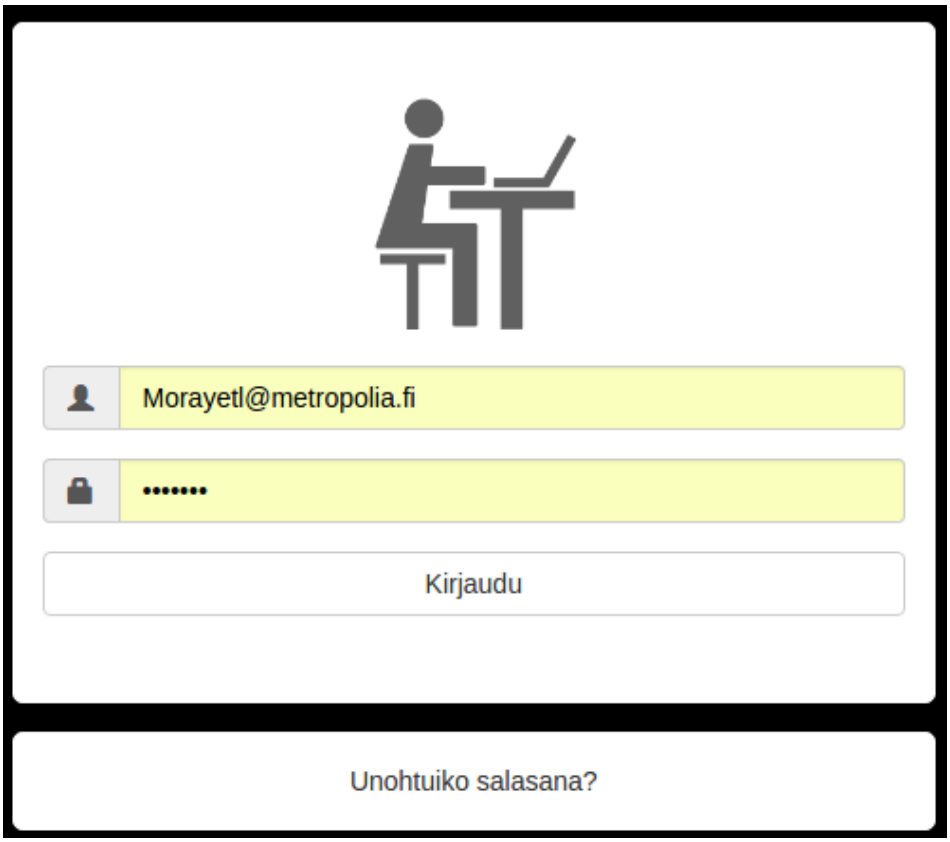
Esimerkkikoodi 15. Todennuksen konfigurointi.

Esimerkkikoodilla 15 kerron palvelimelle, mitkä sivut/metodit eivät vaadi sisäänkirjautumista. Kaikki muut toiminallisuudet vaativat sisäänkirjautumisen paitsi login.

## 6.3 Käyttöliittymä

### 6.3.1 Ulkoasu

Käyttöliittymän sivun tyylipohjana oli Bootstrap, joka helpotti sivujen räätälöimisen. Tein sivusta responsiivisen, jotta se skaalautuisi käyttäjän näytön koon mukaan.




Kuva 11. Kirjautumissivun ulkoasu, joka käyttää Bootstrapin tyylipohjaa.

### 6.3.2 Pyyntöjen tekeminen ja kirjautuminen

AngularJS tukee JSON median lähettämisen, mutta palvelinpuoli ei oletusarvoisesti vastaanota JSON dataa. Palvelinpuoli tukee application/x-www-form-urlencoded sisältötyyppiä. Käyttöliittymän puolella määrittelin $httpProvider asetuksille oletusarvoisen sisältötyypin-headeriksi application/x-www-form-urlencoded, jotta palvelin vastaanottaisi pyynnöt.

```
app.config(function($httpProvider) {

   $httpProvider.defaults.headers.post['Content-Type'] = 'appli-
cation/x-www-form-urlencoded; charset=UTF-8';

   $httpProvider.defaults.headers.put['Content-Type'] = 'appli-
cation/x-www-form-urlencoded; charset=UTF-8';

});
```

Esimerkkikoodi 16. $httpProviderin POST- ja PUT-pyyntöjen header-konfigurointi.

Ennen pyyntöjen lähetystä olio muunnetaan $.param-metodilla, jotta palvelin voisi hyväksyä datat. Pyynnön onnistuessa metodi tallentaa evästeeseen kirjautumistiedot käyttäjästä.

```
$scope.login = function () {
       $http({
           method : "POST",
           url : "/php/users/login.json",
           data: $.param($scope.credentials)
       }).then(function mySucces(response) {
           $rootScope.user = response.data.user;
           $location.path("/frontpage");
           var exp = new Date();
           exp.setMinutes(exp.getMinutes()+20);
           $cookies.put('logged', 'loggedIn',{expires: exp});

       }, function myError(response) {
           alert("Kirjautuminen ei onnistunut.");
       });
   };
```

Esimerkkikoodi 17. Login-metodi, jolla tehdään kirjautumispyyntö ja tallennetaan eväste.

### 6.3.3 Navigointi

Navigoinnin toteutuksessa käytettiin Bower-paketinhallinnasta ladattua UI-routeria. Ui-router-lisäosalla pystyimme toteuttamaan päivittymättömän verkkosovelluksen. Se on siitä kätevä, että sillä pystyy yhdistämään muita näkymiä ja ohjaimia yhteen sivuun. HTML5-tila mahdollistaa edellisen sivuun navigoimisen ja requireBase ollessaan false poistaa ristikkomerkin URL osoitteesta.

```
app.config(function($stateProvider, $locationProvider) {
   $locationProvider.html5Mode({enabled: true, requireBase:
false});

   $stateProvider
       .state("login",{
           url: "/login",
           templateUrl: "app/html/login.html",
           controller: "loginCtrl"
       });
});
```

Esimerkkikoodi 18. Reitityksien ja ohjaimien määrittely Ui-routerilla.

### 6.3.4 Istunnon hallinta

Istunnon hallinta ja uudelleenohjaus tapahtuvat run-lohkon sisällä. Run-lohko pitää huolen, että luvattomalla asiakkaalla ei ole oikeuksia mennä järjestelmän sivuihin kirjautumatta. Run-lohko myös uudelleenohjaa käyttäjän kirjautumissivulle, jos on istunto vanhentunut. Järjestelmä ohjaa kirjautumissivulta etusivulle, jos istunto on vielä voimassa.

## 7 Yhteenveto

Työssä perehdyttiin MVC-malliin ja verkkopalveluihin. Verkkopalvelussa totesin, että REST-rajapinnan toimintaperiaatteiden ymmärtäminen oli helpompaa verrattuna SOAP:iin. Kävimme myös läpi toteutustekniikat, jotka perustuivat MVC-arkkitehtimalliin. Sen jälkeen kävimme web-sovelluksen tietoturvallisuusasiat läpi, mikä oli todella mielenkiintoista. Lopuksi kävimme läpi sovelluksen toteutusvaiheet.

Insinöörityön tavoitteena oli tehdä työvuorosovellus. Oppimistavoitteena oli perehtyä siihen, miten web-sovellus/web palvelin maailma toimii ja miten sellainen toteutetaan. Käyttöliittymä toteutettiin AngularJS-sovelluskehyksellä hyödyntäen Bootstrappia. Palvelinpuolen toiminnallisuudet toteutettiin CakePHP-sovelluskehyksellä. Tehty työ toteutettiin suunnitelman mukaisesti. Käyttöliittymässä oli pyynnön tekemisen ja päivämääräformaattien kanssa vaikeuksia, mutta ne sain selvitettyä pienellä tutkimuksella.

Työtä tehdessäni opin, miten tieto liikkuu internetissä käyttöliittymän ja palvelimen välillä. Käyttöliittymä tekee pyyntöjä palvelimelle, palvelin antaa vastauksen. REST-rajapinnassa istunto tietoja ei tallenneta palvelimen puolelle, vaan tiedot tallennetaan käyttöliittymiin. Vertaillessani SOAP:pia REST-arkkitehtiin totesin, että REST sopii paremmin nykyaikaiseen palvelinpuolen tekniikkaan uusien laitteistojen takia (Puhelimet, tabletit ja yms.). Perehdyin aika paljon tietoturvallisuuteen ja siihen, miten kannattaa suojata verkkosovellus hyökkääjiltä.

Sovellusta voisi jatkokehittää työntekijän arvioinnilla, jotta työnantaja saa lisää tietoa työntekijästä. Sen lisäksi siihen voisi lisätä uutissivu, johon työnantajat voivat lisätä tiedotteita. Puhelin sovellus olisi myös hyvä tällaiselle projektille, koska käyttäjän ei tarvitse käyttää selainta.

Kiitän Metropolian oppilaitoista opetuksista ja näistä vuosista. Lisäksi kiitän vanhempiani, jotka ohjasivat minut tälle linjalle.

## Lähteet


1 Model–view–controller. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>. Luettu. 16.8.2016.

2 MVC Architecture. Verkkodokumentti. Chrome. <https://developer.chrome.com/apps/app_frameworks>. Luettu. 26.1.2017.

3 Six Benefits of Using MVC Model for Effective Web Application Development. Verkkodokumentti. Chrome. <https://developer.chrome.com/apps/app_frameworks>. Luettu 26.1.2017.

4 Verkkodokumentti. Tutorialsteacher. <http://www.tutorialsteacher.com/Content/images/mvc/mvc-architecture.png>. Luettu 26.1.2017.

5 What is PHP. Verkkodokumentti. Wikipedia. <http://fi2.php.net/manual/en/intro-whatis.php>. Luettu 7.7.2016.

6 Lifecycle of a PHP web request (5.1.1). Verkkodokumentti. Webstepbook. <http://www.webstepbook.com/supplements/slides/ch05-php.shtml>. Luettu 11.7.2016.

7 CakePHP at a Glance. CakePHP. <https://book.cakephp.org/3.0/en/intro.html>. Luettu 17.3.2017.

8 CakePHP Conventions. Verkkodokumentti. CakePHP. <https://book.cakephp.org/2.0/en/getting-started/cakephp-conventions.html#file-and-class-name-conventions>. Luettu 17.3.2017.

9 Components. Verkkodokumentti. CakePHP. <https://book.cakephp.org/3.0/en/controllers/components.html>. Luettu 17.3.2017.

10 AngularJS. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/AngularJS>. Luettu 23.1.2017.

11 Two-way data binding. Verkkodokumentti. Dotnettricks. <http://www.dotnettricks.com/img/angularjs/two-waybinding.png>. Luettu 23.1.2017

12 What Is AngularJS? 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/introduction>. Luettu 23.1.2017.

13 Brad, Green. 2015. Verkkodokumentti. Wikipedia. <http://angularjs.blogspot.fi/2015/08/angular-1-and-angular-2-coexistence.html>. Luettu 25.01.2017.

14 Nyambati, Thomas. 2016. AngularJS 1.x Fundamentals. Verkkodokumentti. <https://scotch.io/tutorials/angularjs-1-x-fundamentals-part-1>. Luettu 26.1.2017.

15 Angular 1 vs Angular 2 – a high-level comparison. 2015. Verkkodokumentti. Codingpedia. <http://www.codingpedia.org/jhadesdev/angular-1-vs-angular-2-a-high-level-comparison/>. Luettu 7.2.2017.

16 Verkkodokumentti. Quora. <https://qph.ec.quoracdn.net/main-qimg-811de07402f863f2e3a9427131f36bb9>. Luettu 7.2.2017.

17 Green, Brad. 2015. Verkkodokumentti. <http://angularjs.blogspot.fi/2015/12/building-mobile-apps-with-angular-2-and.html>. Luettu 7.2.2017.

18 AngularJS Scope. Verkkodokumentti. W3schools. <https://www.w3schools.com/angular/angular_scopes.asp>. Luettu 16.2.2017.

19 Dependency Injection. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/di>. Luettu 8.2.2017.

20 What is a Module? Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/module>. Luettu 8.2.2017.

21 Understanding Controllers. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/controller>. Luettu 8.2.2017.

22 Mindtree. Verkkodokumentti. Jackalstack. <http://jackalstack.com/training/mindtree/Angularjs/#/18>. Luettu 8.2.2017.

23 Services. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/services>. Luettu 8.2.2017.

24 What does it do? Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/$location>. Luettu 8.2.2017.

25 Service components in ng. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ng/service>. Luettu 8.2.2017.

26 Creating Custom Directives. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/directive>. Luettu 8.2.2017.

27 What is npm? Verkkodokumentti. NPM. <https://docs.npmjs.com/getting-started/what-is-npm>. Luettu 16.2.2017.

28 Benesch, Christian. 2014. Verkkodokumentti. <https://www.quora.com/What-was-the-web-like-before-REST-principles-were-in-wide-use>. Luettu 7.3.2017.

29 SOAP. Verkkodokumentti. Wikipedia. < https://en.wikipedia.org/wiki/SOAP>. Luettu 7.3.2017.

30 Cox, Jack. 2015. Verkkodokumentti. https://www.captechconsulting.com/blogs/soap-vs-rest-for-mobile-services>. Luettu 7.3.2017.

31 Restful Api Structure. 2016. Verkkodokumentti. Gatech. <http://gtjourney.gatech.edu/live/gt-devhub/documentation/restful-api-structure>. Luettu 11.7.2016.

32 Verkkodokumentti. Gurufocus. <http://static.gurufocus.com/images/rest.png>. Luettu 10.1.2016.

33 Creating RESTful Routes. 2016. Verkkodokumentti. CakePHP. <https://book.cakephp.org/3.0/en/development/routing.html#resource-routes>. Luettu 18.7.2016.

34 Representational state transfer. Verkkodokumentti. Wikipedia. <https://en.wikipedia.org/wiki/Representational_state_transfer#Client-Server>. Luettu 12.3.2017.

35 Templier, Thierry. 2015. Verkkodokumentti. <http://restlet.com/company/blog/2015/12/15/understanding-and-using-cors/>. Luettu 22.3.2017.

36 Introducing JSON. Verkkodokumentti. JSON. <http://www.json.org/>. Luettu 20.7.2016.

37 Siles, Raul. Verkkodokumentti. 2016. <https://www.owasp.org/index.php/Session_Management_Cheat_Sheet>. Luettu 22.07.2016.

38 Wilander, John. 2011. Verkkodokumentti. <http://appsandsecurity.blogspot.fi/2011/04/rest-and-stateless-session-ids.html>. 22.07.2016.

39 Whichers, Dave. 2016. Verkkodokumentti. <https://www.owasp.org/index.php/REST_Security_Cheat_Sheet>. Luettu 1.8.2016.

40 Cross-site Scripting (XSS) Attack. 2016. Verkkodokumentti. Acunetix <http://www.acunetix.com/websitesecurity/cross-site-scripting/>. Luettu 1.8.2016.

41 SQL Injection (SQLi). 2016. Verkkodokumentti. <http://www.acunetix.com/websitesecurity/sql-injection/>. Luettu 1.8.2016.

42 Whichers, Dave. 2016. Verkkodokumentti. <https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet>. Luettu 1.8.2016.

43 Whichers, Dave. Verkkodokumentti. <https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet>. Luettu 1.8.2016.

44 Tietoturva. 2015. Verkkodokumentti. CS. <http://www.cs.tut.fi/~seitti/2015/kalvot/tietoturva/all.html>. Luettu 3.8.2016.

45 2.3 Murtotekniikat. 2014. Verkkodokumentti. Okol. < http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kehittaminen/tietoturvajarjestelmat/internetin_tietoturva/murtotekniikat.htm>. Luettu 3.8.2016.

46 Verkkodokumentti. CS. <http://www.cs.tut.fi/~seitti/2015/kalvot/img/hashpass-salt.png>. Luettu 3.8.2016.

47 Bruteforce calculator. 2016. Verkkodokumentti. Open Security Research. <http://calc.opensecurityresearch.com/>. Luettu 3.8.2016.