VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Laura Hantula

TIETOTURVALLINEN KÄYTTÄJÄN-LUONTI JA KIRJAUTUMINEN WEB-SOVELLUKSEEN

Tekniikka
2019

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Laura Hantula
Opinnäytetyön nimi	Tietoturvallinen käyttäjänluonti ja kirjautuminen Web-sovellukseen
Vuosi	2019
Kieli	suomi
Sivumäärä	42
Ohjaaja	Pirjo Prosi

Opinnäytetyön tarkoituksena oli kehittää sovellus, joka sallii käyttäjien kirjautua Web-sovellukseen siihen luodulla käyttäjätilillä tai sosiaalisen median tunnistautumistiedoilla.

Työ toteutettiin luomalla SPA AngularJS:a, HTML5:ttä ja CSS:ää hyväksi käyttäen. Palvelinpuolen sovellus luottaa toiminnallisuuksissaan ASP.NET Web API 2:een sekä OWIN-väliohjelmistoon. Sovellus hyödyntää tunnistevälineitä käyttäjän todentamiseen.

Opinnäytetyön tuloksena saatiin toimiva Web-sovellus, jolla voidaan demonstroida tietoturvallista käyttäjätietojen hallintaa sekä tunnistevälinepohjaista todentamista.

Avainsanat	AngularJS, Full stack, tunnitautumisväline, SPA, ASP.NET Web API 2

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Laura Hantula
Title	Secure User Registration and Login for a Web Application
Year	2019
Language	Finnish
Pages	42
Name of Supervisor	Pirjo Prosi

The aim of this thesis was to develop an application that allows users to log into a web application by either using a newly created designated account or their social media credentials.

The thesis was executed by creating a SPA that deploys AngularJS, HTML5 and CSS. The back-end of the application heavily relies on ASP.NET Web API 2 and OWIN middleware in its functionalities. The application uses token based approach on user authentication.

The outcome of the thesis was a functioning web application that can be used to demonstrate secure user management and token based authentication.

Keywords AngularJS, Full stack, token, SPA, ASP.NET Web API 2

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

LYHENTEET JA TERMIT

SPA	Single Page Application. Yhden HTML-sivun varaan rakennettu Web-sovellus
Tunnisteväline	Sähköisen tunnisteen käyttämiseen ja suojaamiseen tarkoitettu väline, puhekielessä vakiintunut termi *token*.
API	Application Programming Interface. Ohjelmointirajapinta
Sessio	Istunto, jonka katsotaan alkavan kun käyttäjä kirjautuu tietokoneelle, ohjelmaan tai sivustoon, ja loppuvan, kun käyttäjä kirjautuu ulos tai sulkee tietokoneen.
Natiivialusta	Native platform. Mobiilialusta, jolle tehdyt sovellukset tulee kirjoittaa sen kanssa yhteensopivalla ohjelmointikiellä, esimerkiksi Androidille pääosin Javalla.
SOAP	Simple Object Access Protocol. XML-pohjainen viestintäprotokolla tietokoneiden välistä tiedonvaihtoa varten.
Java	Internetin käyttöön suunniteltu Olio-ohjelmointikieli.
XML	Extensible Markup Language. Merkkikieli tiedonjakoon eri järjestelmien, kuten Internetin kautta.
C++	Yleinen ohjelmointikieli, joka toimii suurimmassa osassa alustoja ja käyttöjärjestelmiä.

HTTP-palvelu	Hypertext Transfer Protocol –palvelu. Mikä tahansa palvelu, joka käyttää HTTP-protokollaa
MVC	Model, View, Object. Ohjelmointiarkkitehtuurimalli, joka jakaa sovelluksen kolmeen loogiseen osaan: käyttöliittymä, data ja sovelluslogiikka.
UI	User interface. Käyttöliittymä.
View Engine	View Engine luo HTML-tiedostoja ohjelmoijan tekemistä näkymistä.
IIS	Internet Information Services. Yleinen Microsoftin kehittämä Web-palvelin, joka toimittaa käyttäjälle pyydettyjä HTML-sivuja tai tiedostoja.
System.Web	Nimiavaruus, joka tarjoaa luokkia sekä käyttölittymiä, jotka mahdollistavat selaimen ja palvelimen välisen kommunikaation. /17/
Assembly	Ohjelmien jakelussa käytettävä pienin yksikkö.
JS	JavaScript, kevyt, tulkattava ohjelmointikieli
UTF-8	Unicode Transformation Format. Merkkien enkoodausstandradi
ng-direktiivi	AngularJS attribuuttidirektiivi, joka muuttaa HTML-elementtien ulkonäköä ja käyttäytymistä.
RESTful API	REprentational State Transfer. Ohjelmointirajapinta, joka hyödyntää HTTP-pyyntöjä datan manipuloimiseen.
RDBMS	Relational Database Management System. Relaatiotietokantojen hallintajärjestelmä

ETL-operaatio	Extract. transform, load. Yhdistelmä kolmesta tietokantafunktiosta, jotka mahdollistavat tiedon vetämisen yhdestä tietokannasta ja sen sijoittamisen toiseen.
IntelliSense	Koodin täydennystyökalu.
Refaktorointi	Lähdekoodin sisäisen rakenteen muuttaminen niin, että sen toiminnallisuus säilyy.
URI	Uniform Resource Identifier. Osoittaa tiedon tai WWW-sivun paikan.
Entiteettiohjelmointikehys	Avoimen lähdekoodin kartoitusviitekehys, joka on osa .NET-ohjelmointikehystä.
Koodivetoinen	Sovelluskehityslähestymistapa, jossa keskitytään sovelluksen toimialueeseen ja sovelluksen kehittäminen aloitetaan luokkien luomisesta eikä tietokantarakenteen suunnittelusta.
XHR-pyyntö	XMLHttpRequest. Olio, jolla voidaan pyytää dataa Web-palvelimelta
HTTP 401	Virhekoodi, joka kertoo, että asiakasohjelmalla ei ole tarvittavia todentamistietoja pyydettyä resurssia varten.
Tuottaja	Tuottaja-luokat operoivat logiikkakerroksen ja tietokantojen välillä.
HTTP 302	Statuskoodi käyttäjän uudelleenohjausta varten.
Allekirjoitettu tunnisteväline	Signed Token. Tunnisteväline, jossa on palvelimen luoma allekirjoitus, jonka avulla palvelin vahvistaa sen omakseen. /21/

Header	Tallennettavan tai siirrettävän datablokin alkuun sijoittuva täydentävä tieto.
Tiiviste	Hajautusfunktiolla luotu merkkijono, joka tiivistää tiedon pienempään tilaan. Dataa ei voi palauttaa alkuperäiseen tilaan, mutta sitä voidaan käyttää digitaalisena allekirjoituksena tai taulukkoindeksinä.
Hajautusfunktio	Algoritmi, joka luo mistä tahansa datasta sekoitetun merkkijonon.

1 JOHDANTO

Tietoturvallinen käyttäjänluonti ja kirjautuminen Web-sovellukseen mahdollistaa käyttäjän antamien tietojen käsittelyn sekä varastoinnin tietoturvallisin keinoin. Lisäksi se estää anonyymejä käyttäjiä katselemasta suojattua dataa ja näkymiä. Käyttäjän tietojen koskemattomuuden varmistaminen on tärkeää, jotta voidaan luoda luotettava palvelusuhde käyttäjän ja palveluntarjoajan välillä. Käyttäjien antamat tiedot on säilytettävä eheinä, eli ne eivät saa muuttua luomisen, käsittelyn ja siirron aikana, sekä kiistämättöminä, eli kaikki tiedonsiirtoon tai –käsittelyyn osallistuneet tekijät täytyy olla tunnistettavissa. Oikeutetuilla käyttäjillä tulee olla myös viiveetön ja helppo saatavuus heille oikeutettuun tarkastettuun tietoon. /2/

Tämän opinnäytetyön aikana rakennetaan SPA-arkkitehtuurin mukainen yhden sivun sovellus, joka hyödyntää AngularJS:a sekä tunnistevälinepohjaista lähestymistä selainpuolella ja ASP.NET Web API 2:a, Owin-väliohjelmistoa ja ASP.NET identitya palvelinpuolella. Sovelluksen tarkoitus on mahdollistaa käyttäjän nopea luominen ja todentaminen, jotta käyttäjälle voidaan esittää kaikki turvalliset näkymät, joihin hänet on oikeutettu Web -sovelluksessa. Sovellus suojelee käyttäjän kirjautumistietoja salaamalla kirjautumisdatan ja kirjaamalla käyttäjän ulos aina 30 minuutin epäaktiivisen käytön jälkeen. Käyttäjälle annetaan myös mahdollisuus kirjautua jo olemassa olevan Facebook- tai Google-tilin kautta uuden käyttäjätilin luomisen sijaan.

Tuloksena on toimiva rekisteröitymis- ja todentamissovellus, jota voidaan soveltaa suurimpaan osaan Web -sovelluksista.

2 TYÖN SUUNNITTELU JA MÄÄRITTELY

Ennen ohjelmoinnin aloittamista käydään läpi minimivaatimukset, jotka sovelluksen tulee täyttää, ja valitaan toteuttamiseen sopivat teknologiat.

2.1 Lähtökohta

Lähtökohtana on rakentaa turvallinen tapa luoda käyttäjiä ja sallia käyttäjien kirjautuminen Web-sovellukseen. Sovellus on yksinkertainen, eikä vaadi käyttäjältä henkilökohtaisten tietojen luovuttamista.

2.2 Rekisteröitymis- ja kirjautumissovelluksen määrittely

Sovelluksen palvelinpuoli tulee olemaan Owin-väliohjelmiston päälle rakennettu ASP.NET Web API 2 -projekti. Palvelinpuolta ei rakenneta suoraan ASP.NET-viitekehyksen päälle, koska palvelin konfiguroidaan myös antamaan Oauth-tunnistevälineitä käyttämällä Owin-väliohjelmistoa. Kaiken rakentaminen samalle kanavalle on siksi järkevämpää. Lisäksi sovellus käyttää Owin-väliohjelmiston päälle rakennettua ASP.NET Identity-järjestelmää uusien käyttäjien rekisteröimiseen, ja heidän pääsytietojensa todentamiseen ennen tunnistevälineen luomista.

Selainpuolen SPA rakennetaan käyttämällä HTML5:ttä, AgnularJS:ää ja Twitter Bootstrapia.

2.2.1 Sovelluksen vaatimusmäärittely

Sovelluksen vaatimusmäärittelyssä kuvataan siltä vaadittavat tärkeimmät toiminnot ja tavoitteet. Yleensä vaatimusmäärittelyssä tarkastellaan miten Web-sovelluksen tulisi toimia ja miten toiminnallisuudet saadaan taattua ja optimoitua. /3/

Tätä projektia varten vaatimusmäärittely toteutettiin yksinkertaisesti listaamalla välttämättömät ominaisuudet, jotta käyttäjä pystyy turvallisesti tunnistautumaan Web-sovellukseen valitsemallaan keinolla.

Taulukko 1. Käyttäjänluonnilta ja kirjautumiselta vaadittavat toiminnallisuudet.

Viite	**Kuvaus**	**Prioriteetti**
F1	Sovellus on SPA-sovellus.	1
F2	Sovellus käyttää AngularJS API:a, HMTL5:ttä ja CSS-tyyliä selainpuolella.	1
F3	Sovellus käyttää ASP.NET-viitekehystä.	1
F4	Luo uusi käyttäjä.	1
F5	Todenna käyttäjä.	1
F6	Kirjaa käyttäjä ulos.	1
F7	Kirjaudu Facebook-tilillä.	2
F8	Kirjaudu Google-tilillä.	2
F9	Takaa tietojen turvallinen käsittely.	1
1 = pakollinen ominaisuus, 2 = toissijainen ominaisuus		

Toiminnallisten ominaisuuksien lisäksi projektissa on oleellista ottaa huomioon sen käytettävyyteen vaikuttavat tekijät. Yleiset Web-sovellukset on tarkoitettu julkiseen käyttöön, ja siksi niiden tulisi olla tarpeeksi selkeitä ulkoasultaan ja toimin-

noiltaan, että eritasoiset Internetin käyttäjät pystyvät käyttämään niitä ilman ylimääräistä ohjausta. Sovelluksen käytettävyyden kannalta on tärkeää myös, että se vastaa nopeasti käyttäjän tekemiin pyyntöihin. Suurin osa käyttäjistä sulkee web-sovelluksen, jos sen lataaminen kestää kauemmin kuin kolme sekuntia. /4/

Taulukko 2. Muut huomioita vaativat ominaisuudet.

Ominaisuus	**Kuvaus**	**Prioriteetti**
Käytettävyys	Sovellus ohjaa käyttäjän toimintaa, ja on niin selkeä, että jokainen joka pystyy käyttämään normaalia Internetsivua, onnistuu käyttämään sovellusta ensimmäisellä kerralla.	1
Turvallisuus	Sovellus toimii HTTPS-yhteydessä ja tarvittavat varokeinot on otettu huomioon estämään yleisimmät tietoturvapoikkeamat.	1
Vasteaika	<2 s	1
1 = pakollinen ominaisuus		

2.2.2 Projektin kattamat käyttötapaukset

Projekti kattaa seuraavat käyttötapaukset:

- Sallii käyttäjän rekisteröityä antamalla käyttäjätunnuksen ja salasanan.

- Tallentaa yllä mainitut pääsytiedot tietoturvallisin keinoin.
- Estää anonyymeja käyttäjiä katselemasta turvattua dataa tai näkymiä.
- Ei pyydä käyttäjältä pääsytietoja uudelleen seuraavan 30 minuutin aikana kirjautumisesta.
- Antaa käyttäjän kirjautua sovellukseen käyttäen Facebook- tai Google-tunnuksia.

2.3 Ohjelmointiympäristö ja –kielet

Tähän osioon listataan kaikki projektissa käytetyt kehitysalustat, apuvälineet ja ohjelmointikielet, joita projektissa käytettiin, sekä käydään läpi niiden edut, jotka johtivat niiden valintaan.

2.3.1 Visual Studio

Visual Studio on integroitu kehitysympäristö, jolla voidaan kehittää tietokoneohjelmia, Web-sivuja ja sovelluksia sekä mobiilisovelluksia. Visual Studio tukee 36:tta eri ohjelmointikieltä, ja sen sisäänrakennettuihin kieliin lukeutuu C#. Visual Studion koodieditori tukee IntelliSenseä sekä koodin refaktorointia. Visual Studiossa on monia sisäänrakennettuja ominaisuuksia, jotka helpottavat koodin kanssa työskentelyä, kuten mahdollisuus rakentaa käyttöliittymiä graafisesti, tietokantamalleja sekä työkaluja virheiden etsimiseen ja poistamiseen.

2.3.2 SQL Server

SQL Server on Microsoftin relaatiotietokantojen hallintaohjelma, eli RDBMS. SQL Server tukee standardin mukaista ANSI SQL –kieltä, mutta sen lisäksi sillä on oma T-SQL-kieli. SQL Serveriä voi käyttää sekä graafisen käyttöliittymän että komentorivin kautta. Sen ominaisuuksia ovat tietokantojen luominen ja ylläpitäminen, datan analysoiminen ja raportoiminen sisäänrakennettujen analysointi- ja raportointiominaisuuksien avulla ja ETL-operaatioiden toimeenpano SQL Server-integraatiopalveluiden kautta. /16/

2.3.3 Postman

Postman on API-kehitysympäristö, jota voi käyttää HTML-pyyntöjen tekemiseen RESTful APIen testauksessa. Pyynnöt voidaan tehdä graafisen käyttöliittymän kautta. Sen sijaan, että tarvitsisi kirjoittaa koodia pelkästään API-testausta varten.

2.3.4 Git

Git on ilmainen avoimen lähdekoodin versionhallintatyökalu. Se tekee projektista hakemistopuun, josta voidaan luoda paikallisia haaroja eri ominaisuuksien kehittämistä varten. Haarat, tai vain osa niiden sisällöstä, yhdistetään päähakemistoon aina kattavan dokumentoinnin kanssa. Se tekee projektin hallinnasta helppoa, koska virheet on yksinkertaista paikantaa koska kehittäjä voi aina palata aiempaan projektiversioon ja vertailla sitä uusiin muutoksiin. Päähakemistoon ei ikinä tehdä suoria muutoksia, vaan ne suoritetaan aina tekemällä projektista haara ja yhdistämällä se hakemistoon. /15/

2.3.5 Visual C#

Visual C#, eli Microsoft Visual Studion sovellus C#-kielestä, on .NET-ympäristöön tarkoitettu oliopohjainen ohjelmointikieli, joka yhdistää C++:n laskentatehokkuuden ja Visual Basicin helppokäyttöisyyden. C# perustuu C++-kieleen, jonka lisäksi siinä on Java-kielen piirteitä. C# käyttää tiedonsiirtovälineenä XML:ää ja toteuttaa olioiden välisen kommunikoinnin SOAPia käyttäen. /6/

2.3.6 HTML5

HTML5 on viimeisin versio standardista, joka määrittelee Web-sivun rakenteen. HTML, HyperText Markup Language, on elementeistä koostuva merkkikieli, jota käytetään paketoimaan eri osia Web-sivun sisällöstä, jotta se näyttäisi toivotulta tai käyttäytyisi halutulla tavalla. Rakenne kuvataan tageilla, jotka kirjoitetaan < ja > merkkien väliin.

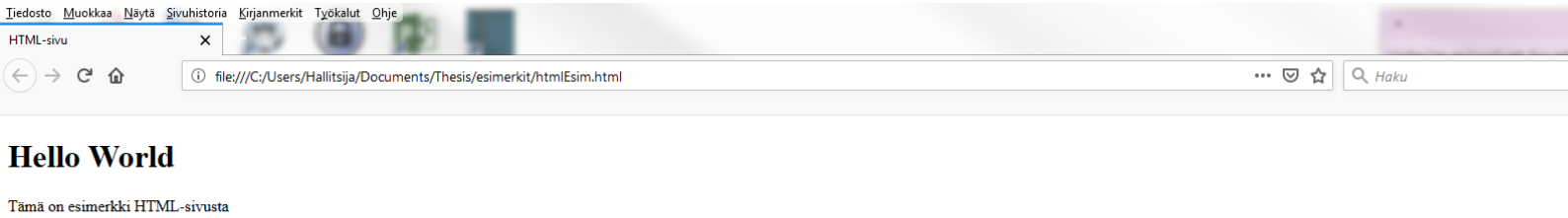
```
<!Doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>HTML-sivu</title>
    </head>
    <body>
        <h1>Hello World</h1>
            <p>Tämä on esimerkki HTML-sivusta</p>
    </body>
</html>
```

Kuvio 1. Esimerkki HTML-kielestä

HTML-dokumentista löytyy aina dokumenttityypin julistus <!Doctype html>. Tämä on peruja ajalta, jolloin HTML-kieli oli vielä uutta ja vaati linkin sääntöihin, joita HTML-dokumentin tuli noudattaa. HTML5 ei enää noudata tätä sääntöä, mutta vaatii silti dokumenttityypin julistuksen toimiakseen. Muut esimerkissä esiintyvät tagit ovat:

- <html>-juurielementti, jolla koko tiedoston sisältö ympäröidään, kuvaa WWW-sivun.
- <head>-elementtiin pakataan dokumentin kaikki sivulla oleva sisältö jota ei näytetä käyttäjälle. Se voi sisältää esimerkiksi avainsanoja hakukoneita varten, tai CSS-dokumentin jossa määritellään sivun ulkonäkö.
- <meta charset=”utf-8”>-elementti määrittelee dokumentin käyttävän UTF-8-standardin mukaista merkistöä, johon lukeutuu suurin osa kirjoitettujen kielien merkistöistä.
- <title>-elementtiin kirjoitetaan selaimen otsikkoriville tuleva otsikko
- <body>-elementti sisältää kaiken sisällön, joka halutaan näyttää sivun käyttäjälle
- <h1>-elementti määrittelee tekstin koon ja tyylin suureksi otsikoksi
- <p>-elementti sisältää tekstikappaleet.

Esimerkin HTML-dokumentti tuottaa seuraavanlaisen sisällön WWW-sivulle:



Kuvio 2. Esimerkki HTML-sivun sisällöstä

Otsikkoelementissä oleva teksti on suurempi ja voimakkaampi suhteessa alempaan kappaleeseen. <head>-elementin sisällä oleva nimi näkyy välilehden yläreunassa.

2.3.7 AngularJS

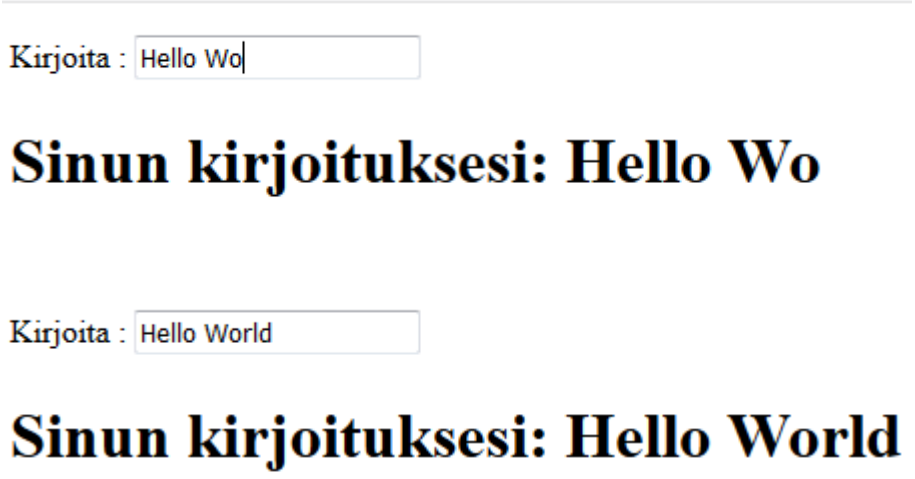
AngularJS on erittäin kevyt, rakenteellinen JavaScript-ohjelmointikehys, joka on yhteensopiva pöytäkone- ja mobiiliselainten kanssa. Se on tarkoitettu dynaamisten Web-sovellusten kehittämiseen ja mahdollistaa HTML-syntaksin laajentamisen ilmaisemaan sovelluksen komponentteja selkeästi ja ytimekkäästi. AngularJS sopii erinomaisesti SPA-sovellusten kehittämiseen, koska se poistaa perinteisen HTML-sivun staattisuuden ja mahdollistaa Web-sivun toimimisen sovelluksena. /14/

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Kirjoita : <input type="text" ng-model="text"></p>
  <h1>Sinun kirjoituksesi: {{text}}</h1>
</div>

</body>
</html>
```

Kuvio 3 Esimerkki AngularJS:ä Web-sivulla

AngularJS-ohjelmointi kehystä levitetään JavaScript tiedostona, joka lisätään Web-sivulle käyttämällä skriptitagia <script src=”https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js”></script>. Se laajentaa HTML-kieltä lisäämällä siihen ng-direktiivejä:

- ng-app-direktiivi määrittää sivun AngularJS-sovellukseksi.
- ng-model-direktiivi sitoo HTML-kenttien syötteen sovellusdataan.



Kuvio 4. Esimerkki AngularJS-sovelluksesta

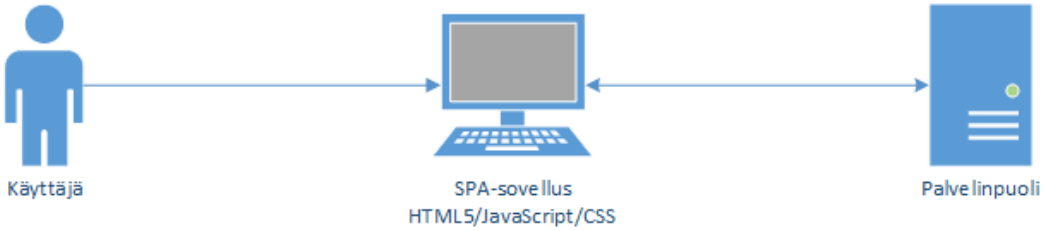
Lopputuloksena on Web-sovellus, joka vastaa suoraan käyttäjän syötteeseen.

2.4 Sovelluksessa käytettävät teknologiat

Tässä osiossa tarkastellaan projektiin käytettyjä teknologioita.

2.4.1 SPA-arkkitehtuuri

SPA-sovellus (Single-Page Application) tarkoittaa JavaScript-ohjelmointikielellä toteutettua yhden sivun sovellusta, jonka kaikki tiedot sekä toiminnallisuudet ladataan kerralla Internet-sivulle, mutta niitä paljastetaan käyttäjälle vain osa kerrallaan. Se on yksisivuinen dokumentti, joka muuttaa näkymiä käyttäjän toimien mukaisesti. /5/



Kuvio 5. SPA-sovelluksen yksinkertaistettu rakenne.

SPA-arkkitehtuuri vähentää käytännössä uusien sivulatausten määrää perinteiseen web-sovellukseen verrattuna. Uusi sisältö haetaan palvelimelta ja käyttäjälle näytetään vain käytön kannalta oleellisia näkymiä.

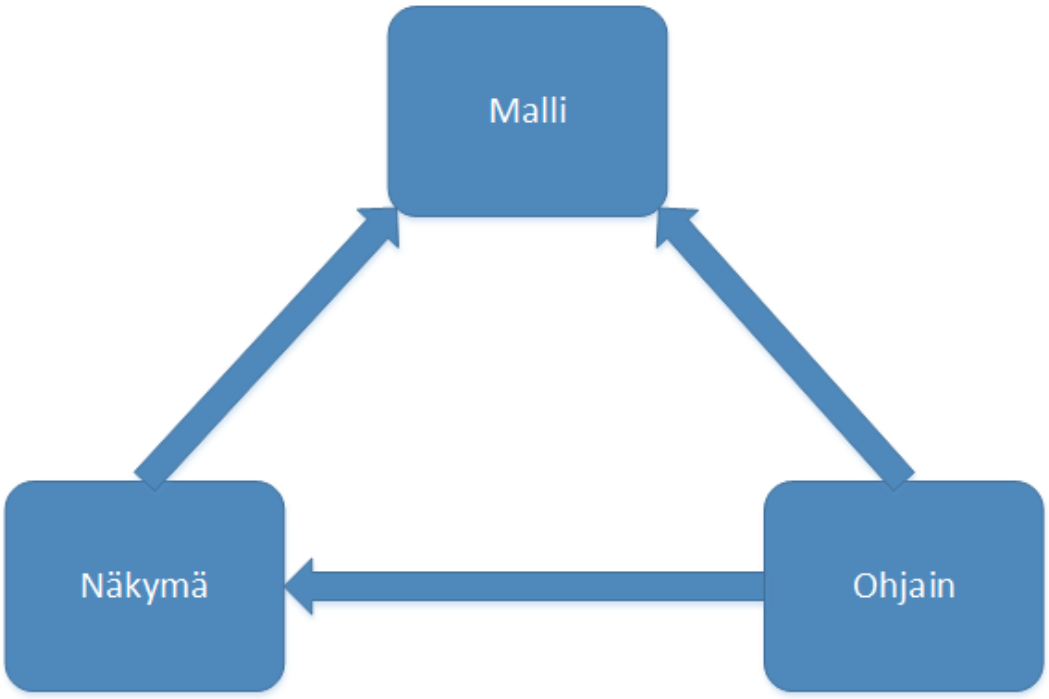
2.4.2 ASP.NET Web API 2

ASP.NET Web API 2 on ilmainen Microsoftin tarjoama ohjelmistokehys HTTP-palvelujen rakentamista varten. ASP.NET tarjoaa kolme eri sovelluskehystä web-sovellusten kehitystä varten: Web Forms, ASP.NET MVC ja ASP.NET Web Pages. Kaikilla kolmella voi rakentaa dynaamisia Web-sovelluksia selaimille ja mobiiliselaimille sekä mobiilisovelluksia natiivialustoille. /7/

2.4.3 ASP.NET Identity

Identity on käyttäjänhallintaan tarkoitettu kirjasto. Toisin kuin edeltäjänsä, Identity sallii eteenpäin ohjautuvat sisäänkirjautumiset sosiaalisen median todennuksen tarjoajilta, kuten Facebook, Twitter ja Google. /8/

2.4.4 MVC 5



Kuvio 6. MVC-komponentit.

MVC 5 on Microsoftin kehittämä sovelluskehys MVC-arkkitehtuurin käyttöönottoa varten. MVC-arkkitehtuuri tarkoittaa sovelluksen erottelua kolmeen erilliseen loogiseen komponenttiin: malli (model), näkymä (view) ja ohjain (controller). MVC-arkkitehtuuri on erittäin suosittu standardi webkehityksessä skaalautuvien ja laajennettavien projektien kehitykseen.

Mallikomponentti vastaa kaikesta dataan liittyvästä logiikasta, jonka kanssa käyttäjä reagoi. Tämä voi tarkoittaa joko näkymän ja ohjaimen välillä siirrettävää tietoa tai mitä tahansa muuta liiketoimitalogiikkaan liittyvää dataa. Näkymäkomponenttia käytetään kaikissa sovelluksen UI-komponentteihin liittyvissä logiikoissa. Se sisältää siis kaikki komponentit, kuten tekstikentät, alasvetovalikot ja napit, joiden kanssa käyttäjä on vuorovaikutuksessa. Ohjaimet toimivat rajapintana mallin ja näkymän välillä. Ne prosessoivat kaikki saapuvat pyynnöt, muokkaavat dataa käyttäen mallikomponenttia ja ovat kanssakäymisessä näkymien kanssa tuottaakseen lopullisen tuloksen. Ohjain sisältää siis suurimman osan sovelluksen logiikasta. /9-10/

ASP.NET tukee kolmea isoa kehitysmallia: Web Pagesia, Web Formsia ja MVC:tä. ASP.NET MVC –ohjelmistokehys on kevyt, helposti testattava ohjelmistokehys, joka on integroitu jo olemassa oleviin ASP.NET ominaisuuksiin, esimerkiksi todentamiseen. ASP.NET MVC tärkeimpiä ominaisuuksia on helposti laajennettava sovelluskehys, johon voi lisätä valinnaisia komponentteja. Esimerkiksi sisäänrakennettu ASPX View Engine voidaan korvata kolmannen osapuolen tuotteella tai muokata kehittäjän toiveiden mukaan.

ASP.NET MVC hyötykäyttää sovelluksen komponenttipohjaista mallia jakamalla sen loogisesti malliin, näkymään ja ohjaimeen, joka helpottaa suurien projektien hallitsemista ja yksittäisten komponenttien työstämistä. MVC-rakenne edistää testivetoista kehitysmallia ja helpottaa sovelluksen testausta, sillä kaikki komponentit voidaan suunnitella käyttöliittymäpohjaisesti ja testata jäljitelmäobjekteilla. ASP.NET-ohjelmistokehys on näin ollen erittäin sopiva suurille projekteille, joissa työskentelee usea kehittäjä samanaikaisesti. ASP.NET MVC ei käytä View State –konseptia, joka on läsnä ASP.NETissa, mikä helpottaa kevyiden web-sovellusten rakentamista ja antaa täyden vallan kehittäjälle. /11/

2.4.5 OWIN

OWIN on standardirajapinta web-sovellusten ja .NET-web-palvelimien välillä. Sen tarkoitus on kytkeä palvelin erilleen sovelluksesta ja samalla tukea yksinkertaisten moduulien kehittämistä .NET-web-kehityksessä sekä edistää avoimen lähdekoodin käyttöä .NET-webkehitystyökaluissa. ASP.NET-ohjelmistokehys on erittäin riippuvainen IIS-palvelimesta ja sitä voidaan ylläpitää vain IIS-palvelimella. Tämä tekee ASP.NET-sovellusten siirtämisestä toiselle alustalle mahdotonta. System.Web Assembly vaatii useita HTTP-pyynnön käsittelytoimenpiteitä toimiakseen, vaikkei niitä käytettäisikään sovelluksessa. Tämän takia kanavalla suoritetaan aina joitakin ei-haluttuja ominaisuuksia, mikä alentaa sovelluksen suorituskykyä.

OWIN-standardia käytetään näiden luottamussuhteiden poistamiseen. Se tekee sovelluksesta modulaarisemman ja kytkee sen komponentit löyhemmin yhteen. Vaikka System.Web-riippuvuus poistetaankin sovelluksesta, sen kehitys on silti

hyvin samanlaista kuin perinteisen ASP.NET-sovelluksen kehittäminen, muutamia arkkitehtuurimuutoksia lukuunottamatta. /12/

2.4.6 Bootstrap

Bootstrap on suosittu avoimen lähdekoodin komponenttikirjasto reagoivien ja mobiiliystävällisten selainpuolen sovellusten rakentamista varten. Sitä voidaan käyttää HTML:n, CSS:in ja JS:in kanssa ohjelmoimiseen. Bootrsapin avulla kehitetyt sivustot ovat yhteensopivia kaikkien yleisten Internetselainten, tablettien ja älypuhelinten kanssa. /13/

3 TYÖN TOTEUTUS

Tässä luvussa käydään läpi kaikki sovelluksen onnistumisen kannalta oleelliset toiminnallisuudet ja niiden toteutus. Toiminnallisuudet on jaettu otsikoiden alle, jotka kuvaavat niissä käytettyjä teknologioita.

3.1 Tunnistevälinepohjainen todentaminen ASP.NET Web API 2, Owin ja Identity avulla

Sovellus hyödyntää tunnistevälinepohjaista todentamista selainpuolen sovelluksen ja palvelinpuolen APIn välillä vanhemman evästepohjaisen todentamisen sijaan. Evästepohjaisessa todentamisessa eväste lähetetään jokaisen pyynnön yhteydessä asiakasohjelmalta palvelimelle, jossa sillä tunnistetaan todennettu käyttäjä.

Selainpuolen sovelluskehysten kehittyminen ja muutos tyylissä, jolla Web-sovelluksia kehitetään, suosii käyttäjien tunnistamista allekirjoitetuilla tunnistevälineillä jotka lähetetään palvelimelle jokaisella pyynnöllä. Tämän lähestymistavan etuja ovat mm. palvelimien skaalautuminen, mobiililaiteyhteensopivuus sekä selain- ja palvelinpuolen sulava yhteenliittäminen.

Palvelimen skaalautuminen tarkoittaa tässä projektissa sitä, että palvelimelle lähetettävä tunnisteväline sisältää kaiken todentamiseen vaadittavan tiedon. Palvelimien lisääminen ympäristöön on helppoa, koska riippuvuuksia jaettuihin sessiovarastoihin ei ole. Evästeiden tallentaminen natiivialustalle on monimutkainen prosessi, joten jos sovelluksesta haluaa tehdä mobiiliystävällisen, on järkevämpää käyttää standardoitua käyttäjän todentamistapaa siltä varalta, että selainpuolen APIa halutaan käyttää natiivisovelluksesta. Sulavan yhteenliittämisen puolesta puhuu se, että selainpuolen sovellusta ei ole liitetty tarkoin määrättyyn todentamismekanismiin, vaan tunnisteväline luodaan palvelinpuolella ja sovelluksen API on rakennettu ymmärtämään tunnistevälinettä ja hoitamaan todentamisen.

3.1.1 Palvelinpuolen APIn rakentaminen

Palvelinpuolen APIn kulmakivinä ovat Microsoft ASP.NET Identity ja –Owin ohjelmistokehykset. Sovellukselle luodaan aluksi Startup-niminen luokka, joka käynnistyy aina ensimmäisenä palvelimen käynnistyessä. Tämä määritellään assembly-attribuutilla.

```
[assembly: OwinStartup(typeof(AngularJSAuthentication.API.Startup))]
//Assembly attribuutti määrittää mikä luokka ajetaan ensin ohjelman käynnistyessä.
namespace AngularJSAuthentication.API
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        //Metodi hyväksyy IAppBuilder-parametrin jonka hosti antaa sille kun ohjelma ajetaan - app-parametri on käyttöliittymä jota käytetään Owin-palvelimen sovelluksen muodostamiseen.
        {
            ConfigureOAuth(app);
            HttpConfiguration config = new HttpConfiguration();
            //Objektia käytetään konfiguroimaan API-reittejä, joten periytytetään Register-metodille WebApiConfig-luokassa
            WebApiConfig.Register(config);
            app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
            app.UseWebApi(config); //config-objekti annetaan apumetodille UseWebApi joka kytkee ASP.NET Web APIn OWIN-kanavalle
        }

        public void ConfigureOAuth(IAppBuilder app)
        {
            OAuthAuthorizationServerOptions OAuthServerOptions = new OAuthAuthorizationServerOptions()
            {
                AllowInsecureHttp = true,
                TokenEndpointPath = new PathString("/token"),
                //tunnistevälineiden luomista varten on polku http://localhost:port/token, jonne tehdään HTTP POST -pyyntöjä tunnistevälineen luomista varten
                AccessTokenExpireTimeSpan = TimeSpan.FromMinutes(30),
                //tunnistevälineen elinaika on 30 minuuttia, jos käyttäjä yrittää kirjautua samalla tokenilla tämän ajan jälkeen pyyntö hylätään ja palautetaan statuskoodi 401
                Provider = new SimpleAuthorizationServerProvider()
                //tunnistevälineitä pyytävät käyttäjät validoidaan luokassa SimpleAuthorizationServerProvider
            };

            app.UseOAuthAuthorizationServer(OAuthServerOptions);
            app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());
        }
    }
}
```

Kuvio 7. Startup-luokka

Startup-luokassa määritellään uusi instanssi luokasta OAuthAutorizationServerOptions, joka on konfiguroitu luomaan tunnistevälineitä päätepisteessä http://localhost:port:/token. Tunnistevälineet vanhenevat 30 minuutin kuluessa kirjautumisesta, eli jos käyttäjä yrittää käyttää samaa tunnistevälinettä uudelleen myöhemmin, pyyntö hylätään ja käyttäjälle palautetaan HTTP-statuskoodi 401. Uuden käyttäjän todentamistiedot todennetaan luokassa SimpleAuthotizationServerProvider.

```
public class SimpleAuthorizationServerProvider : OAuthAuthorizationServerProvider
//periytyy luokasta OAuthAuthorizationServerProvider
{
    public override async Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
    //todentaa asiakasohjelman, tässä tapauksessa asiakasohjelmia on vain yksi, joten metodi on ohitettu ja se palauttaa aina onnistuneen validoinnin
    {
        context.Validated();
    }
```

Kuvio 8. Todentamispalvelimen periytyminen

SimpleAuthotizationServerProvider on OAuthAuthorizationServerProvider-luokasta periytetty luokka, joka vastaa asiakasohjelmien sekä kirjautumistietojen todentamisesta. Jos käyttäjän todentamistiedot menevät läpi todentamisesta, luodaan luokka nimeltä ClaimsIdentity, jolle lisätään kaksi vaatimusta nimeltä sub ja role. Nämä lisätään loppuliseen todentamisvälineeseen.

```
var identity = new ClaimsIdentity(context.Options.AuthenticationType); //ClaimsIdentity-luokka luodaan ja sille annetaan autentikaatiotyyppi jos todentamistiedot ovat valideja
identity.AddClaim(new Claim("sub", context.UserName)); //Lisätään kaksi claimiä, sub ja role, jotka lisätään todentamisvälineeseen
identity.AddClaim(new Claim("role", "user"));

context.Validated(identity); // Todentamisvälineen luominen tapahtuu taustalla kun context.Validated(identity) kutsutaan
```

Kuvio 9. Todentamisvälineen luominen

Kommunikoinnista tietokantojen kanssa huolehtii tietokantakontekstiluokka. Tässä tapauksessa tietokantakontekstiluokka periytetään IdentityDbContext-luokasta. Se luo tietokantakontekstiluokalle entiteettiohjelmointikehyksen koodivetoiselle kartoitukselle, sekä DbSet-ominaisuudet, joita tarvitaan identiteettitaulujen hallitsemiseen SQL-palvelimella. /18/

Tietokantakontekstiluokka toimii yhteistyössä UserModel-luokan kanssa. UserModel-luokka sisältää mallin kaikista ominaisuuksista, jotka tulee lähettää tietokantaan aina kun uusi käyttäjä rekisteröidään. UserModel-luokka hyödyntää data-annotaatioita todentamaan rekisteröinnissä annetut tiedot.

Sovellus tarvitsee metodit käyttäjien todentamiseen ja rekisteröimiseen. Sitä varten luodaan uusi luokka, joka luottaa ASP.NET UserManager-luokkaan. UserManager-luokka tarjoaa kaiken domainlogiikan, jota tarvitaan käyttäjätietojen kanssa työskentelyyn. Se osaa sotkea (hash) salasanan aina kun se on tarpeellista, tietää milloin ja miten todentaa käyttäjä sekä miten hallita vaatimuksia. /19/

Jotta sovellus pystyy rekisteröimään uudet käyttäjät, se tarvitsee Web API-ohjaimen. Ohjaimeen lisätään Register-niminen metodi, jonka päätepisteeseen voidaan lähettää HTTP POST-pyyntöjä.

```
[Route("Register")]
public async Task<IHttpActionResult> Register(UserModel userModel)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    IdentityResult result = await _repo.RegisterUser(userModel);
    IHttpActionResult errorResult = GetErrorResult(result);
    if(errorResult != null)
    {
        return errorResult;
    }

    return Ok();
}
```

Kuvio 10. Rekisteröintimetodi

Pyynnön sisällön tulee olla JSON-objektimuodossa:

{

”userName” : ”Käyttäjänimen sisältö”,

”password” : ”Salasanan sisältö”,

”confirmPassword” : ”Salasanan sisältö”

}

Tämä palauttaa HTTP-tilakoodin 200, luo automaattisesti uuden lisäyksen tietokantatauluun, joka on määritelty yhteysmerkkijonossa, ja käyttäjän tiedot lisätään tietokantatauluun nimeltä dbo.AspNetUsers. Yhteysmerkkijono määritellään automaattisesti luodussa Web.Config-luokassa, ja se tulee nimetä samoin kuin tietokantakontekstiluokka. Sovellukseen lisätään myös toinen ohjain, joka sisältää suojatun käyttäjälle näytettävän näkymän. Tässä tapauksessa sivulla on staattista dataa tekaistuista asiakastilauksista.

3.1.2 Palvelinpuolen APIn testaus

Palvelinpuolen API testataan lähettämällä HTTP POST –pyyntö sovelluksen päätepisteeseen. Pyynnön sisältötyyppi on x-www-form-urlencoded, jotta pyyntö saadaan esitettyä lomakemuodossa. Palvelin palauttaa tunnistevälineen vastausviestissä.

3.2 AngularJS tunnistevälinetodentaminen ASP.NET Web API 2, Owin ja Identity avulla

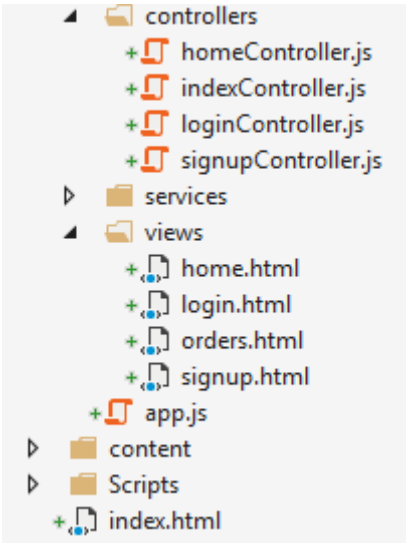
Tässä kappaleessa käydään läpi miten sovellukseen rakennetaan yksinkertainen SPA AngularJS:in avulla. SPA sallii käyttäjien rekisteröityä sovellukseen antamalla käyttäjänimen ja salasanan, estää anonyymejä käyttäjiä katsomasta heiltä kiellettyjä näkymiä sekä sallii rekisteröityjen käyttäjien kirjautua sisään ja pysyä sisäänkirjautuneina 30 minuutin ajan tai kunnes he kirjautuvat ulos sovelluksesta.

3.2.1 HTML-näkymät

Projektiin lisätään SPA-arkkitehtuurin mukainen yksi etusivu, index.html, joka toimii säiliönä sovellukselle. Sivu sisältää navigaatiopalkin sekä AngularJS-direktiivin sovelluksen muiden näkymien esittämiseen. Sivulla viitataan myös kaikkiin kolmannen osapuolen JavaScript-kirjastoihin sekä CSS-tiedostoihin, joita sovellus käyttää. Etusivun lisäksi projektiin lisätään HTML-näkymät kotisivua, kirjautumista, rekisteröitymistä sekä tilausten katselemista varten.

3.2.2 Ohjaimet

Jokainen HTML-näkymä tarvitsee vastaavan ohjaimen. Ohjaimet sisältävät kaiken liiketoimintalogiikan, mitä vastaava näkymä esittää käyttäjälle graafisessa muodossa. Sovelluksella on ohjaimet käyttäjien rekisteröimistä, todennettujen käyttäjien uudelleenohjausta, näkymien muuttamista sekä kotinäkymän näyttämistä varten.



Kuvio 11. Ohjaimet ja vastaavat näkymät kansiorakenteessa

Ohjaimiin voidaan kuitenkin lisätä muitakin toiminnallisuuksia, esimerkiksi käyttäjien uudelleenohjauksesta vastuussa oleva loginController-ohjain uudelleenohjaa kaikki todentamattomat käyttäjät, jotka yrittävät katsella tilauksia takaisin sisäänkirjautumissivulle.

```
app.controller('signupController', ['$scope', '$location', '$timeout', 'authService', function ($scope, $location, $timeout, authService) {
    //sisältää liiketoimintalogiikan uusien käyttäjien tallentamiseen ja kutsuu authServicen saveRegistration-metodia

    $scope.savedSuccessfully = false;
    $scope.message = "";

    $scope.registration = {
        userName: "",
        password: "",
        confirmPassword: ""
    };

    $scope.signUp = function () {

        authService.saveRegistration($scope.registration).then(function (response) {

            $scope.savedSuccessfully = true;
            $scope.message = "Rekisteröityminen onnistui, ohjautuminen kotisivulle tapahtuu 2 sekunnin kuluessa.";
            startTimer();

        },
            function (response) {
                var errors = [];
                for (var key in response.data.modelState) {
                    for (var i = 0; i < response.data.modelState[key].length; i++) {
                        errors.push(response.data.modelState[key][i]);
                    }
                }
                $scope.message = "Käyttäjää ei voitu rekisteröidä " + errors.join(' ') + " virheen takia.";
            });
    };

    var startTimer = function () {
        var timer = $timeout(function () {
            $timeout.cancel(timer);
            $location.path('/login');
        }, 2000);
    }
```

Kuvio 12. Esimerkki ohjaimesta

Kuvion 12 ohjain on vastuussa uusien käyttäjien rekisteröimisestä kutsumalla autentikaatiopalvelun rekisteröimisentallennusfunktiota.

3.2.3 Palvelut

Palvelu on funktio tai objekti joka on pelkästään AngularJS-sovelluksen saatavilla. Tässä projektissa palveluiksi on eritelty kaikki HTTP-pyyntöjä vaativat funktiot. Sovelluksella on kolme palvelua, joista autentikaatiopalvelu huolehtii uusien käyttäjien rekisteröimisestä, todennettujen käyttäjien sisään- ja uloskirjaamisesta sekä käyttäjille luotujen todennusvälineiden varastoimisesta paikalliseen varastoon, jotta todennusväline voidaan lähettää jokaisen pyynnön mukana, mikä vaatii turvattuja resursseja palvelinpuolelta.

Toinen merkittävä palvelu on AngularJS Interceptor, jonka ansiosta jokainen XHR-pyyntö voidaan kaapata ja sen dataa manipuloida ennen kuin se lähetetään palvelinpuolen APIlle tai sen jälkeen kun vastaus on saatu APIltä. Tässä tapauksessa pyynnöt halutaan kaapata ennen lähettämistä, jotta niihin voidaan asettaa tunnisteväline, jonka lisäksi palvelinpuolen APIn vastaus halutaan tarkistaa virhekoodien varalta. Jos vastauksessa on virhekoodi, se tulee tarkastaa ja virheen ollessa HTTP 401, käyttäjä ohjataan takaisin sisäänkirjautumissivulle.

3.2.4 Web-sovelluksen käynnistäminen

SPA-sovelluksen rakenteelle on oleellista rakentaa tiedosto pelkästään sovelluksen käynnistämistä ja reitittämistä varten. Tämän tiedoston nimi on aina app.js, ja se on vastuussa moduulien luomisesta sovelluksessa. Tässä projektissa on käytössä vain yksi moduuli, jota voidaan pitää kokoelmana palveluita, direktiivejä ja suodattimia, joita käytetään sovelluksessa. Jokaisella moduulilla on konfiguraatiolohko, jossa se pannaan täytäntöön sovelluksen esilatausohjelman aikana.

```
$routeProvider.when("/signup", {//Rekisteröitymisnäkymä, jonne myös anonyymeillä käyttäjillä on pääsy
    controller: "signupController",
    templateUrl: "/app/views/signup.html"
});
```

Kuvio 13. Esimerkki konfiguraatiolohkosta.

3.3 OAuth Refresh –tunnistevälineiden aktivoiminen AngularJS sovelluksessa käyttäen ASP.NET Web API 2 ja Owin teknologiaa

Tuen lisääminen Oauth Refresh –tunnistevälinesiin tekee todentamispalvelimesta huomattavasti monimutkaisemman. Tässä osiossa käydään läpi miten palvelimelle lisätään asiakasohjelmia, pysyviä uudelleenlatautuvia tunnistevälineitä (englanniksi refresh tokens), konfiguroidaan dynaamisesti uudelleenlatautuvien tunnistevälineiden vanhenemisajankohdat ja kumotaan uudelleen latautuvat tunnistevälineet.

3.3.1 Uudelleenlatautuvien tunnistevälineiden käyttäminen

Uudelleenlatautuvien tunnistevälineiden käyttö perustuu siihen, että käyttäjä todentaa itsensä käyttäjänimen, salasanan ja asiakasohjelman tietojen avulla. Jos tiedot ovat oikein, käyttäjälle lähetetyssä vastauksessa on lyhytikäinen yhteystunnisteväline sekä pitkäikäinen uudelleenlatautuva tunnisteväline, joka toimii tunnisteena yhteystunnistevälineelle. Kun yhteystunnisteväline vanhenee, uudelleenlatautuvaa tunnistevälinettä käytetään uuden yhteystunnistevälineen saamiseen. /22/

3.3.2 Uudelleenlatautuvien tunnistevälineiden hyödyt

Sovellus hyödyntää uudelleenlatautuvia tunnistevälineitä pitkäikäisten yhteystunnistevälineiden sijaan kolmen pääsyyn vuoksi:

1. Yhteystunnistevälineen sisällön päivittäminen
2. Pääsyn kumoaminen todennetuilta käyttäjiltä
3. Todentamistietojen varastoinnille ja pyytämiselle ei ole enää tarvetta.

Yhteystunnistevälineet pitävät sisällään kaiken informaation todennetusta käyttäjästä luomisensa jälkeen. Jos käyttäjälle luodaan pitkäikäinen tunnisteväline, jossa hänelle on annettu käyttäjän rooli, tämä tieto pysyy tunnistevälineessä sen eliniän

ajan. Käyttäjän roolia ei voi muuttaa, esimerkiksi järjestelmänvalvojan rooliin pyytämättä häntä todentamaan itseään uudestaan, jotta todentamispalvelin voi päivittää uuden roolin käyttäjän tietoihin. Yhteydenotto käyttäjään aina kun tietoja halutaan päivittää, ei ole realistista, joten heille määrätään lyhytikäisiä yhteystunnistamisvälineitä, jotka vanhenevat 30 minuutin jälkeen. Kun käyttäjä saa uuden yhteystunnistamisvälineen uudelleenlatautuvan tunnistevälineen avulla, todentamispalvelin voi lisätä tunnistevälineeseen käyttäjän uudet tiedot, esimerkiksi lisätä hänet järjestelmänvalvojan ryhmään, uuden yhteystunnistevälineen luonnin yhteydessä.

Jos käyttäjälle annetaan pitkäikäinen tunnisteväline, hänellä on pääsy palvelimen resursseihin kunnes tunnisteväline on vanhentunut. Pääsyn kumoamiseen ei ole standardimenetelmää ja se on mahdollista vain, jos todentamispalvelimella on erikseen määriteltynä logiikka, joka pakottaa luotujen tunnistevälineiden tallentamisen tietokantaan ja tarkistaa ne tietokannasta jokaisen pyynnön yhteydessä. Uudelleenlatautuvilla tunnistevälineillä järjestelmänvalvoja voi kumota pääsyn poistamalla käyttäjän uudelleenlatautuvan tunnistevälineen tunnisteen tietokannasta, jolloin yhteystunnistevälineen vanhetessa todennuspalvelin estää käyttäjän pääsyn resursseihin, koska uudelleenladattava tunnisteväline ei ole enää saatavilla.

Käyttäjältä pyydetään tunnistautumistietoja vain kun tunnistautuvat ensimmäisen kerran, koska todentamispalvelin voi antaa hyvin pitkäikäisiä uudelleenlatautuvia tunnistevälineitä. Tunnistevälineen elinikä voi olla esimerkiksi 365 päivää, ja käyttäjä pysyy sisäänkirjautuneena tämän ajan, ellei järjestelmänvaloja kumoa uudelleenlatautuvaa tunnistevälinettä. Tästä on erityisesti hyötyä sovelluksessa, jossa tunnistautumistietojen kysyminen ei useinkaan ole toteuttamiskelpoista.

3.3.3 Uudelleenlatautuvat tunnistevälineet ja asiakasohjelmat

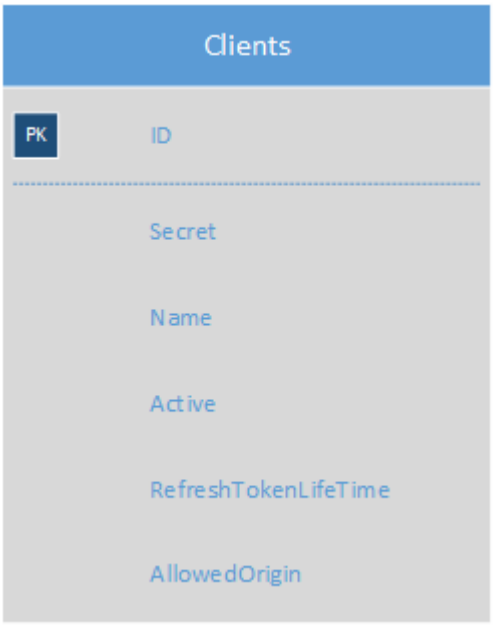
Uudelleenlatautuvat tunnistevälineet täytyy sitoa asiakasohjelmaan. Asiakasohjelma on sovellus, joka kommunikoi palvelinpuolen APIn kanssa saadakseen tunnistevälineen. Jokaisella asiakasohjelmalla tulee olla ID ja Secret, jotka saadaan yleensä kun ohjelma rekisteröidään palvelinpuolen APIn kanssa.

Asiakasohjelman ID on ainutlaatuinen julkinen merkkijono, jota käytetään tunnistamaan sovellus muiden samaa palvelinpuolen APIa käyttävien sovellusten joukossa. Asiakasohjelman ID voidaan sisällyttää sovelluksen lähdekoodiin, mutta asiakasohjelman Secret pitää aina pitää luottamuksellisena. Tämän vuoksi Secret ei voi olla osallisena lähdekoodia JavaScript-sovelluksissa, koska sen pitäminen salaisena ei ole mahdollista avoimen lähdekoodin käytännön vuoksi.

Uudelleenlatautuvien tunnistevälineiden sitominen asiakasohjelmaan on tärkeää, jotta muut asiakasohjelmat eivät saisi mahdollisuutta käyttää tämän sovelluksen todentamispalvelimen luomia tunnistautumisvälineitä.

3.3.4 Muutokset tietokantarakenteeseen

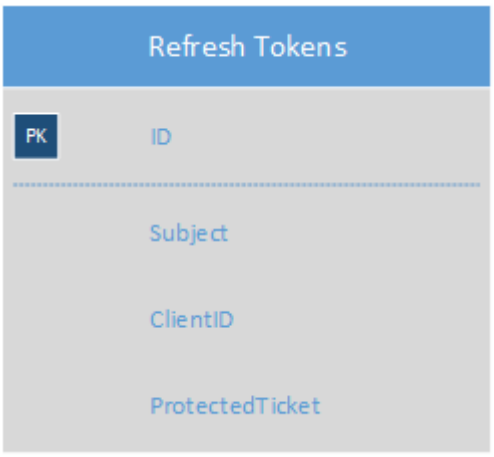
Asiakasohjelmistot, jotka kommunikoivat palvelinpuolen APIn kanssa, täytyy tallentaa pysyvästi.



Kuvio 14. Asiakasohjelmien tietokantataulu

Secret-kolumnin sisältö suojataan tekemällä siitä tiiviste hajautusfunktion avulla, jotta se pysyy suojattuna kaikilta, joilla on pääsy tietokantaan. Active-kolumnin sisältö mahdollistaa asiakasohjelmien helpon poistamisen. Järjestelmänvalvoja voi

asettaa asiakasohjelman epäaktiiviseksi, jolloin kaikki sen kautta pyydetyt tunnistevälineet hylätään. RefreshTokenLifeTime-kolumnissa määritellään uudelleenlatautuvan tunnistamisvälineen elinikä. Jos sovelluksella olisi useampi asiakasohjelma, ne voisivat jaella tunnistautumisvälineitä joilla on eri elinikä. AllowedOrigin-kolumnia käytetään CORS-resurssienjaon konfiguroimiseen palvelinpuolen APIlle.



Kuvio 15. Uudelleenlatautuvien tunnistevälineiden tietokantataulu

ID-kolumnissa on uudelleenlatautuvan tunnistevälineen tunniste hajautettuna. Subject-kolumni viittaa mille käyttäjälle tunnisteväline kuuluu, ja ClientID mille asiakasohjelmalle se kuuluu. ProtectedTicket-kolumni on OWIN-väliohjelmistoa varten, joka pystyy rakentamaan yhteystunnistevälineen automaattisesti tätä merkkijonoa apuna käyttäen.

3.3.5 Asiakasohjelman todentaminen

Asiakasohjelma pitää todentaa, kun se pyytää yhteystunnistevälinettä tai käyttää uudelleenlatautuvaa tunnistevälinettä uuden yhteystunnistevälineen saamiseen. Asiakasohjelma todennetaan ottamalla selville ClientID- ja Secret-arvo todennusheaderistä. Header tarkistetaan myös siltä varalta, että käyttäjä ei lähettänyt mitään asiakasohjelmatietoja.

Asiaskasohjelman ClientID-arvon vastaanottamisen jälkeen tarkistetaan onko se rekisteröity palvelinpuolen APIn kanssa. Jos asiakasohjelmaa ei ole rekisteröity,

pyyntö hylätään ja tunnistautmistiedot mitätöidään. Pyyntö hylätään myös, jos asiakasohjelma ei ole aktiivinen tietokannan mukaan.

Lopuksi asiakasohjelman sallittu alkuperä ja uudelleenlatautuva tunnistevälineen elinikä varastoidaan OWIN-kontekstiin, jotta se on saatavilla kun uudelleenlatautuva tunnisteväline luodaan ja sen vanhenemisaika määritellään. Jos kaikki on todenpitävää, tunnistautumiskonteksti merkitään pitäväksi, mikä tarkoittaa, että asiakasohjelma on läpäissyt tarkistuksen ja sovellus voi siirtyä suorittamaan seuraavaa koodia.

3.3.6 Uudelleenlatautuvan tunnistevälineen luominen

Uudelleenlatautuvien tunnisteiden luomista varten lisätään tuottajaluokka, nimeltä SimpleRefreshTokenProvider. Tämä luokka sisältää logiikan tunnistevälineiden luomisesta ja niiden säilömisestä tietokantaan. Jokaiselle uudelleenladattavalle tunnistevälineelle luodaan yksilöllinen GUID-tunnusnumeron. GUID on 128-bittinen automaattisesti generoitu satunnaisluku, jota käytetään resurssien tunnistamiseen.

Uudelleenladattavan tunnistevälineen elinikä luetaan OWIN-kontekstista, johon se varastoitiin asiakasohjelmaa todennettaessa. Tätä lukua käytetään määrittämään kuinka kauan uudelleenladattava tunnisteväline on voimassa minuutteina. Nämä tiedot sarjallistetaan yhteen merkkijonoon, joka säilytetään tietokannassa.

Tunnistevälineistä luodaan dokumentointi, joka tallennetaan RefreshTokens-tieotkantatauluun. Tunniste tiivistetään ennen tietokantatauluun tallentamista siltä varalta, ettei kukaan, jolla on tietokantapääsy, pysty lukemaan sitä. Lopuksi uudelleenlatautuvan tunnistevälineen tiivistämätön ID-arvo lähetetään takaisin vastausviestin rungossa.

3.3.7 Uudelleenlatautuvan tunnistevälineen käyttö yhteystunnistevälineen luomisessa

Yhteystunnistevälineen luomiseen tarvittava logiikka toteutetaan keräämällä ensin kaikki tunnistautumiseen tarvittava informaatio yhdellä metodilla ja luomalla toi-

nen metodi joka huolehtii uusien vaatimusten julkaisemisesta sekä vanhojen päivittämisestä ja niiden sisällyttämisestä uusiin yhteystunnistevälineisiin ennen tunnistevälineen lähettämistä käyttäjälle.

Tunnistautumistietojen keräämisestä vastuussa oleva metodi vastaanottaa uudelleenlatautuvan tunnistevälineen ID-arvon, tekee siitä tiivisteen ja etsii tunnistevälineen tiivistetyn ID-arvon tietokantataulusta RefreshTokens. Jos ID-arvo löytyy, hyödynnetään OWIN-väliohjelmiston luomaa ProtectedString-arvon sisältämää sarjallistettua merkkijonoa luomaan esitys käyttäjätiedoista, jotka on reititetty kyseiseen uudelleenladattavaan tunnistevälineeseen. Lopuksi taulukosta poistetaan mahdollinen olemassa oleva uudelleenlatautuva tunnisteväline, koska tämä sovellus sallii vain yhden uudelleenlatautuvan tunnistevälineen käyttäjälle.

Uusien vaatimusten julkaisusta ja säilömisestä huolehditaan lukemalla alkuperäinen ClientID-arvo, jota verrataan HTTP-pyynnön kanssa lähetettyyn arvoon. Jos arvot eivät täsmää, pyyntö hylätään, koska tunnistevälinettä ei ole sidottu tässä projektissa luotuun asiakasohjelmaan.

Tarkistuksen jälkeen viestiin voidaan lisätä uusia vaatimuksia tai poistaa vanhoja, mikä ei olisi mahdollista ilman uudelleenlatautuvien tunnistevälineiden käyttöä. Viestin päivittämisen jälkeen luodaan uusi yhteystunnisteväline, joka palautetaan vastausviestin rungossa.

Kun nämä metodit on suoritettu onnistuneesti, luodaan uusi uudelleenlatautuva tunnisteväline, joka palautetaan paluuviestissä yhteystunnistevälineen kanssa.

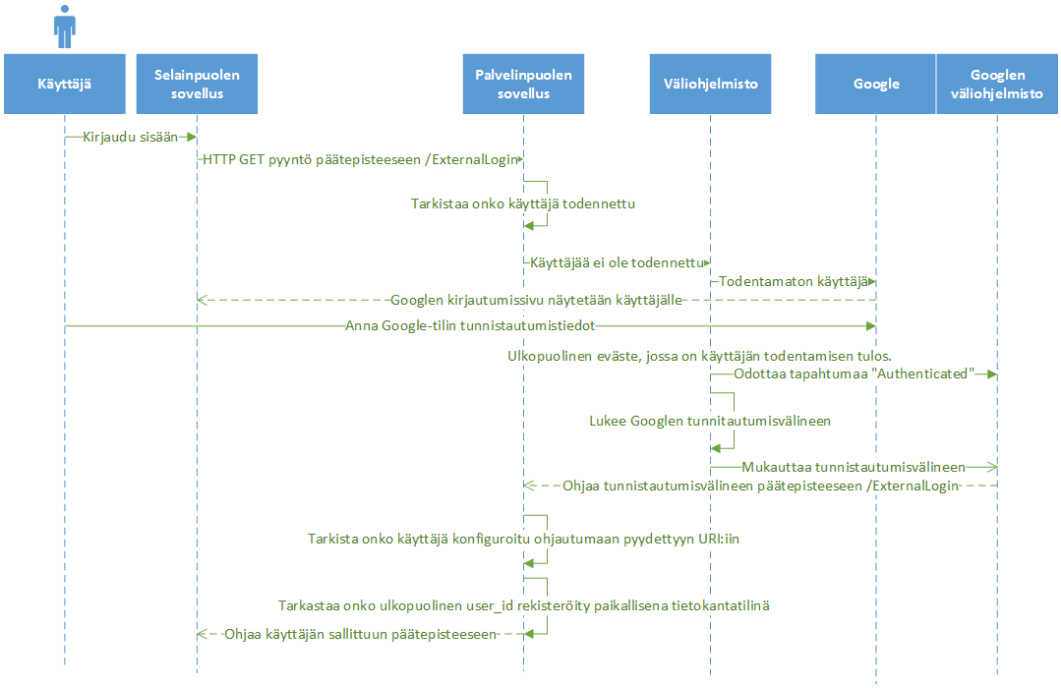
3.3.8 Uudelleenlatautuvan tunnistevälineen kumoaminen

Tunnistevälineiden kumoaminen on erittäin yksinkertaista. Sovellukseen tarvitsee vain lisätä uusi ohjain, jossa on metodi tunnistevälineen tietojen poistamiseen RefreshToken-tietokantataulusta. Oikeus tunnistevälineen poistamiseen annetaan järjestelmänvalvojalle, joka pystyy näin rajoittamaan sisäänkirjautuneiden käyttäjien pääsyä sovellukseen. Jos käyttäjä, jonka tunnistevälinetiedot on poistettu, yrittää

pyytää uutta yhteystunnistevälinettä käyttäen vanhaa uudelleenlatautuvaa tunnistevälinettä, pyyntö hylätään ja käyttäjä joutuu todentamaan itsensä uudelleen käyttäjänimen ja salasanan avulla.

3.4 ASP.NET Web API 2 ulkopuoliset kirjautumiset Facebookin ja Googlen kautta AngularJS-sovelluksessa

Sovellus mahdollistaa käyttäjän kirjautumisen Google- ja Facebook-tileillä erillisen käyttäjätilin luomisen sijaan. Sitä varten sovellukseen pitää lisätä tuki näiden palveluntarjoajien ulkopuolisia tuottajia varten. Lisäksi ulkopuoliset todentamattomat käyttäjätilit liitetään paikallisiin käyttäjätileihin.



Kuvio 16. Sekvenssidiagrammi Googlen todentamistietojen avulla kirjautumisesta

Tapahtumaketju, joka saa alkunsa kun käyttäjä kirjautuu kolmannen osapuolen todentamistietojen avulla, on kuvattu kuvioon 16. Aluksi AngularJS-sovellus lähettää HTTP GET-pyynnön tuntemattomaan päätepisteeseen, joka on määritelty palvelinpuolen APIssä. Kun päätepiste saa GET-pyynnön, se tarkistaa onko käyttäjä todennettu. Sekvenssidiagrammissa oletetaan, että käyttäjää ei ole vielä todennettu, jolloin se lähettää huomautuksen ulkopuoliselle väliohjelmalle, joka on vastuussa uusien kirjautumispyyntöjen käsittelystä. Tässä tapauksessa väliohjelma kuuluu

Googlelle, jolloin käyttäjälle näytetään Google+-hyväksymisnäkymä ja pyydetään antamaan käyttäjätunnus ja salasana. Google lähettää paluuviestinä palvelinpuolen APIlle ulkopuolisen evästeen, johon sisältyy todentamisyrityksen tulos.

Googlen väliohjelma tulostaa tapahtuman nimeltä Authenticated, josta tunnistautumissovellus pystyy lukemaan kaikki Googlen tekemät ulkopuoliset vaatimukset. Tunnistamisen kannalta hyödyllinen vaatimus on nimeltä AccessToken, joka edustaa Google yhteystunnistevälinettä. Tämän vaatimuksen antajalla ei kuitenkaan ole paikallista tunnistusoikeutta, joten AccessToken-arvoa ei voi käyttää suoraan turvallisten pyyntöjen tekemiseen palvelinpuolen APIlle. Ulkopuolisen palveluntarjoajan yhteystunnisteväline nimetään uudelleen, jotta se tunnistetaan ulkopuolisena tunnistevälineenä ja Googlen väliohjelmisto ohjaa takaisin anonyymiin päätepisteeseen.

Kun käyttäjä on todennettu kolmannen osapuolen evästeen avulla tarkastetaan, että alkuperäisessä pyynnössä olevat asiakasohjelman ID ja uudelleenohjausosoite ovat oikein ja, että kyseinen asiakasohjelma on konfiguroitu ohjautumaan määriteltyyn URIin. Sovellus lähettää HTTP 302-uudelleenohjauksen määriteltyyn URIin, joka sisältää tarvittavat kirjautumistiedot tiivistemerkkijonona URL-osoitteessa.

Kun AngularJS-sovellus vastaanottaa vastauksen, se ohjaa käyttäjän päätepisteeseen, joka hyväksyy ulkopuolisen yhteystunnistevälineen ja käyttää sitä käyttäjäntunnistamiseen, jonka jälkeen käyttäjälle annetaan paikallinen yhteystunnisteväline. Paikallisella yhteystunnistevälineellä saadaan pääsy palvelinpuolen APIn suojattuihin päätepisteisiin.

4 YHTEENVETO

Projektin lopputuloksena on toimiva SPA, joka sallii käyttäjän kirjautumisen kolmella eri tavalla. Yksinkertaisen tunnistautumisohjelman tekeminen on melko suoraviivaista, mutta tietoturvallisuuden lisääminen hankaloittaa monesti ohjelman rakenteen suunnittelua ja monimutkaistaa koodia. Kehittäjän täytyy ottaa huomioon kaikki mahdolliset keinot, joilla sovellukseen tai vain osiin sitä pystyy pääsemään käsiksi.

Sovelluksen tietoturvallisuutta voisi parantaa vielä kytkemällä OWIN-autentikaatiopalvelimen pois resurssipalvelimelta, varsinkin jos sovellukseen halutaan lisätä enemmän päätepisteitä. Tämän projektin laajuutta ajatellen, sen sisällyttäminen tähän opinnäytetyöhön ei olisi ollut järkevää, koska se lisää taas huomattavasti sovelluksen monimutkaisuutta.

LÄHTEET

/1/ ASP.NET Dokumentaatio. ASP.NET Core Overview. Viitattu 13.5.2019. https://docs.microsoft.com/en-us/aspnet/

/2/ Suomen Internetopas. Tietoturva – Mitä on tietoturva?. Viitattu 7.5.2019. http://www.internetopas.com/yleistietoa/tietoturva/

/3/ Tekninen vaatimusmäärittely. Viitattu 13.5.2019. https://www.sofokus.com/fi/tekninen-vaatimusmaarittely/

/4/ An D. 2017. Päivitetty helmikuussa 2018. Find out how you stack up to new industry benchmarks for mobile page speed. Viitattu 13.5.2019. https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/

/5/ Halme A. 10.4.2018. Mikä on Single Page App ja mihin sitä käytetään?. Viitattu 20.5.2019. https://citydevlabs.fi/single-page-app/

/6/ Moghadampour G. 2012. C# - Windows- ja tietokantaohjelmointi. Jyväskylä. Docendo.

/7/ Dykstra T., Anderson R., Latham L., Addie S., Pasic A., Patel S., Schonnig N., Warren G. 3.12.2010. ASP.NET Overview. Viitattu 14.5.2019. https://docs.microsoft.com/en-us/aspnet/overview

/8/ Anderson R., Schonnig N., Latham L., Addie S., Pickett W., Goldman P., Pasic A., Dykstra T., Chidi E. 1.22.2019. Introduction to ASP.NET Identity. Viitattu 14.5.2019. https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

/9/ Informaatioteknologia – Jyväskylän yliopston informaatioteknologian tiedekunta. Web-sovellukset (TIEA218). Luennot. Model-view-controller (MVC) –arkkitehtuuri. Viitattu 14.5.2019. http://appro.mit.jyu.fi/tiea2080/luennot/mvc/

/10/ Tampereen teknillinen yliopisto. Web-sovellusten arkkitehtuuri. Lainattu 14.5.2019. http://www.cs.tut.fi/~seitti/2015/kalvot/webarchitectures/all.html

/11/ Scott Addie, Isaac Levin, Andy Pasic, Tom Dysktra, Sean G. Wright, Rick Anderson, Nick Schonnig, Timmy Kokke, Genevieve Warren, Luke Latham. 10.11.2018. ASP.NET MVC 5. Viitattu 14.5.2019. https://docs.microsoft.com/en-us/aspnet/mvc/mvc5

/12/ OWIN working group. 10.10.2012. OWIN: Open Web Server Interface for .NET. Viitattu 14.5.2019. http://owin.org/html/spec/owin-1.0.html

/13/ Bootstrap, from Twitter. Viitattu 14.5.2019. https://bootstrapdocs.com/v2.0.2/docs/

/14/ Developer Guide/Introduction. What is AngularJS. Viitattu 14.5.2019. https://docs.angularjs.org/guide/introduction

/15/ Getting Started – What is Git?. Viitattu 15.5.2019. https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

/16/ SQL Server 2017 Features. Viitattu 15.5.2019. https://www.microsoft.com/en-us/sql-server/sql-server-2017-features#CP_StickyNav_1

/17/ System.Web Namespace. Viitattu 15.5.2019. https://docs.microsoft.com/en-us/dotnet/api/system.web?view=netframework-4.8

/18/ Scott Allen. 3.1.2014. ASP.NET Identity with the Entity Framework. Viitattu 17.5.2019. https://odetocode.com/blogs/scott/archive/2014/01/03/asp-net-identity-with-the-entity-framework.aspx

/19/ Scott Allen. 20.1.2014. Implementing ASP.NET Identity. Viitattu 17.5.2019. https://odetocode.com/blogs/scott/archive/2014/01/20/implementing-asp-net-identity.aspx

/20/ Satheesh Babu. 14.2.2017. What is OWIN? A Beginners Guide. Viitattu 20.5.2019. http://www.codedigest.com/posts/1/what-is-owin-a-beginners-guide.

/21/ Intorduction to JSON Web Tokens. Viitattu 20.5.2019. https://jwt.io/introduction/.

/22/ Understanding Refresh Tokens. Viitattu 20.5.2019. https://auth0.com/learn/refresh-tokens/