Leo-Henri Heikkilä

KÄYTTÄJÄN AUTENTIKOINTI JA AUTORISOINTI ASP.NET -SOVELLUKSESSA

KÄYTTÄJÄN AUTENTIKOINTI JA AUTORISOINTI ASP.NET -SOVELLUKSESSA

Leo-Henri Heikkilä

Opinnäytetyö

Kevät 2013

Tietojenkäsittelyn koulutusohjelma

Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä: Leo-Henri Heikkilä
Opinnäytetyön nimi:Käyttäjän autentikointi ja autorisointi ASP.NET -sovelluksessa
Työn ohjaaja: Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Syksy 2013 Sivumäärä:31+1

Työ tehtiin Siperia Systems Oy:lle, joka on oululainen ohjelmistokehitysyritys. Yritys kehittää öljyntorjuntaan ja ympäristön suojeluun liittyvää mobiilisovellusta. Sovelluksen avulla voidaan lähettää palvelimelle esimerkiksi kuvia, tapahtumapaikan koordinaatteja sekä erilaisia mittaustietoja. Sovellusta voidaan käyttää useammalla laitteella samanaikaisesti, jolloin laitteen tai käyttäjän tunnistautuminen palveluun on tärkeää. Lisäksi lähetettävät tiedot voivat sisältää arkaluontoisia tietoja, jonka vuoksi tiedonsiirto täytyy salata.

Työn tavoitteena oli kehittää yrityksen uuden mobiilisovelluksen palvelinpuolen ohjelmistoa. Työn tietoperusta koostuu .NET-ohjelmistokehykseen liittyvistä tuetut kielet-, luokkakirjastot- sekä ASP.NET -osioista. Lisäksi teoriaosuudessa käydään läpi ASP.NET MVC 4 Web API ja siinä tuettuja ominaisuuksia, tarkastellaan MVC-arkkitehtuuria sekä määritellään käsitteet autentikointi ja autorisointi.

Työssä on käytetty suurimmaksi osaksi Microsoftin vuoden 2012–2013 aikana julkaistuja lähteitä, koska käytännön osuudessa käytetty Web API -palvelu on julkaistu vuonna 2012. Opinnäytetyön työosuudessa saatiin onnistuneesti rakennettua palvelu, johon käyttäjä voi tunnistautua. Tunnistautumisen jälkeen käyttäjä voi tehdä palveluun autorisointia vaativia kutsuja. Toimeksiantaja käyttää opinnäytetyön Demossa käytettyjä ratkaisuja yrityksen uuden mobiilisovelluksen HTTP-palvelun kehityksessä.

Palvelun jatkokehitykseen ehdotettiin muun muassa seuraavia ratkaisuja: Autentikointi rakennettaisiin tietoturvan lisäämiseksi filter-luokkaan tai HTTP message handler:n. Käyttäjät tulisi jakaa ryhmiin autentikoinnin yhteydessä. Lisäksi suositeltiin cross-site request forgery -tokenin käyttöönottoa lisäämään tietoturvaa tiedonvälityksessä.

Asiasanat: autentikointi, autorisointi, ASP.NET MVC 4 Web API, tietoturva

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Leo-Henri Heikkilä
Title of thesis: User Authentication and Authorization in the application ASP.NET
Supervisor: Jouni Juntunen
Term and year when the thesis was submitted: Autumn 2013
Number of pages: 31+1

This thesis was done for Siperia Systems Oy, which is a software development company located in Oulu. The company is developing a mobile application for measuring oil destruction and environmental protection. With the application, user can send different kinds of data to the server, such as pictures, site coordinates, as well as a variety of measurement data. Because several devices can be used at the same time, the device or the user's authentication service is essential. In addition, the information sent may contain sensitive data which is why the data must be encrypted during the transmission.

The purpose of this thesis was to develop the server part of software of the company's new mobile application. The theoretical framework consist of sections about supported languages and class libraries in the.NET -framework, as well as a section about ASP.NET in the .NET -framework. Furthermore the theoretical framework covers the ASP.NET MVC 4 Web API and features supported by the Web API as well as MVC architecture and the definitions of authentication and authorization.

The sources used in the thesis are mostly sources released by Microsoft in 2012-2013. The reason for this is that the HTTP service used in the practical part was released by Microsoft in 2012. In the practical part of the thesis, HTTP service was successfully developed so that it enables the clients to authenticate themselves. After the authentication, the client can send the HTTP service requests that require authorization. Siperia Systems Oy uses these techniques to help the development of the new HTTP service.

Several improvements were suggested for the further development of the service. Firstly, the authentication process should be built in filter class or in the HTTP message handler for security reasons. Secondly, users should be devided into groups during the authentication process to the service. Since the basic authentication is vulnerable to cross-site request forgery attacks, the use of an anti-forgery token was recommended.

Keywords: authentication, authorization, ASP.NET MVC 4 Web API, information security

SISÄLLYS

1 JOHDANTO

Suoritin opintoihini kuuluvan työharjoittelun oululaisessa ohjelmistokehitysyrityksessä, Siperia Systems Oy:ssä. Harjoittelun aikana pääsin tutustumaan mobiilisovelluksien kehitykseen. Tietämykseni kasvaessa aloin kiinnostua sovelluksien tietoturvaan liittyvistä asioista. Siperia Systems Oy kehittää muun muassa öljyntorjuntaan ja ympäristön suojeluun liittyvää mobiilisovellusta. Sovelluksen avulla loppukäyttäjä voi lähettää esimerkiksi kuvia, tietoja ja koordinaatteja palvelimelle. Laitteen tai käyttäjän tunnistautuminen on tärkeä osa palvelun toimintaa, koska sovelluksia ja laitteita voi olla useampi toiminnassa samanaikaisesti. Lisäksi lähetettävät tiedot voivat sisältää arkaluontoista materiaalia, jonka vuoksi tiedonsiirtoon täytyy käyttää salattua yhteyttä.

Opinnäytetyön tavoitteena on kehittää yrityksen uuden mobiilisovelluksen palvelinpuolen ohjelmistoa. Opinnäytetyön työosuudessa tehdään demo käyttäjän autentikoinnista ja autorisoinnista ASP.NET MVC 4:ssä. Web API on MVC 4-versiossa uutena ominaisuutena tullut HTTP-palvelu, ja sitä käytetään yleensä mobiilisovelluksien HTTP-palveluiden kehityksessä. (Microsoft 2013e, hakupäivä 4.9.2013.) Yritys hyödyntää demossa käytettyjä ratkaisuja mobiilisovelluksen palvelinpuolen kehityksessä.

Autentikointi on prosessi, jossa varmistetaan, pitääkö jokin väittämä paikkansa. Autentikointi voidaan tehdä esimerkiksi laitteelle, käyttäjälle ja lähetettävälle viestille (Kent & Millett 2003, 33). Työssä selvitettävä autentikointi tarkoittaa käyttäjän tunnistautumista HTTP-palveluun. Käyttäjä tunnistetaan käyttäjätietokantaa vastaan käyttäjätunnuksen ja salasanan avulla.

Autorisoinnissa varmistetaan, että tietyt säännöt täyttyvät, ennen kuin haluttu toiminto suoritetaan. Autorisointi liitetään yleensä autentikoinnin lisäksi rajoittamaan resurssien käyttöä HTTP-palvelussa. (Kent & Millett 2003, 36–41.) Opinnäytetyön teoriaosuudessa käydään läpi, mitä autorisointi tarkoittaa. Työosuudessa tehdään esimerkki resurssien rajoituksesta Web API -palvelussa käyttämällä Web API:ssa tuettua filter-luokkaa (ks. 3.2 Web API).

Toisessa luvussa käsitellään .NET ohjelmistokehitykseen liittyviä asioita. .NET on Windows-kehittäjille suunniteltu ohjelmistokehys (Chappel 2006, 1–3). Valitsin teoriaosuuteen .NET ympäristöstä ASP.NET:n, tuetut luokkakirjastot ja kielet sekä common language runtime:n (CLR). CLR

voidaan kuvata koodin prosessoijana, joka huolehtii koodin ajamisesta sekä yhteensopivuudesta käyttöjärjestelmän kanssa (Chappel 2006, 1–3). CLR:n toimintaan perehtyminen on tärkeää, kun halutaan ymmärtää, miten .NET ohjelmistokehys toimii. Teorian toisessa luvussa tarkastellaan myös C#-, C++- sekä VB.NET-kieliä. Työosuudessa käytetään C#-kieltä, jolloin muut kielet on otettu vertauskuvaksi valitulle kielelle. Standardisoidut luokkakirjastot ovat ohjelmistokehityksen tärkeimpiä ominaisuuksia. Kirjastot tarjoavat mahdollisuuden käyttää valmiiksi rakennettuja toiminnallisuuksia, mikä puolestaan mahdollistaa nopean ohjelmistokehityksen. Lisäksi luvussa tutustutaan ASP.NET-ohjelmistokehykseen. ASP.NET- osiossa käydään läpi ohjelmistokehyksen hyviä puolia sekä tarkastellaan, mihin tarkoitukseen ohjelmistokehystä käytetään. Valitsin ASP.NET-ohjelmistokehyksen, jotta ymmärtäisin tarkemmin, mihin tarkoitukseen ja miten sitä käytetään.

Kolmannessa luvussa tutustutaan Web API -palveluun sekä palvelussa tuettuihin toimintoihin. Teoriassa käydään läpi Web APIn tuetuista toiminnoista työn kannalta olennaisimmat osat, joita ovat filters, model binding and validation, full support for routes ja content negotiation (ks. 3.2 Web API). Filter-luokka käydään läpi tarkemmin, koska sitä käytetään opinnäytetyön työosassa käyttäjän autorisointiin. Luvussa käydään läpi myös MVC-arkkitehtuuri ja sen hyvät puolet. MVC-mallinnuksen ymmärtäminen ja käyttäminen ovat tärkeitä palvelun jatkokehityksen kannalta. Lisäksi luvussa tutustutaan ASP.NET MVC 4:n tuomiin uusiin ominaisuuksiin, joita ovat uusi mobiiliprojektimalli, näkymän palauttaminen loppukäyttäjälle, uudenlainen renderointi ja asynkronisten metodien tuki. Valitsin nämä uudet ominaisuudet, koska ne ovat yrityksen mobiilikehityksen kannalta olennaisia ja tehokkaita tekniikoita.

Neljännessä luvussa tarkastellaan autentikointia ja autorisointia. Lisäksi luvussa käydään läpi Basic-autentikointi. Basic-autentikointi on standardisoitu tapa autentikoida käyttäjä HTTP-palveluun. Valitsin basic-autentikoinnin, koska se on tietoturvallinen, yksinkertainen ja koska sitä on helppo kehittää. Yritys voi myös halutessaan soveltaa basic-autentikointia laitteen autentikointiin.

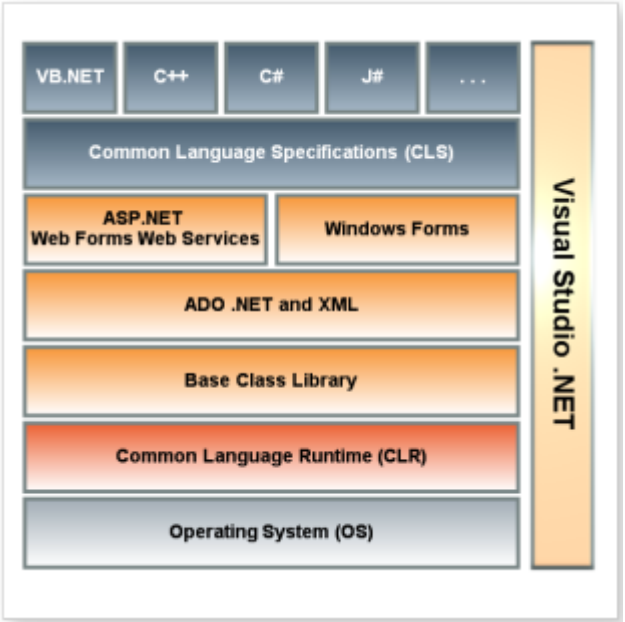
Viidennessä luvussa käydään läpi, millä työkaluilla HTTP-palvelu kehitettiin. Lisäksi luvussa havainnollistetaan, miten autentikointi- ja autorisointi-viestit rakennettiin. Luvussa pohditaan myös, miksi näihin ratkaisuihin päädyttiin ja miten ne toteutettiin.

Opinnäytetyön käytännön osuuteen tehtiin seuraavanlaisia rajauksia: työssä ei käytetty tietokantaa, koska se ei olisi tuonut työlle paljoa lisäarvoa. Sen sijaan työssä käytettävät testikäyttäjät ovat kovakoodattuna listaan (array). Lista sisältää lisäksi tiedot aikaleimoista ja tokeneista, joita käytetään käyttäjän autentikoinnissa ja autorisoinnissa. Työ kehitettiin ja testattiin localhost-ympäristössä. Lisäksi työhön ei tehty mobiilikäyttöliittymää, vaan toiminnallisuus testattiin web-käyttöliittymän avulla.

2 .NET

Hyvän ohjelmiston kehittämiseen voidaan nykyään käyttää melkein mitä tahansa ohjelmointikieltä. Kehitystä voidaan helpottaa huomattavasti, mikäli käytössä on oikeanlainen alusta sekä työkalut. Microsoft esitteli .NET:n ensimmäisen beta-version kesäkuussa vuonna 2000. .NET on Windows-kehittäjille suunniteltu ohjelmistokehys, joka viittaa pääasiassa kahteen asiaan: Common Language Runtime:n (CLR) sekä .NET Framework luokkakirjastoon. (Chappel 2006, 1–3.)

Ohjelmistot, jotka on kehitetty käyttämällä .NET ohjelmistokehystä, vaativat aina CLR:n toimiakseen. CLR tarjoaa muun muassa standardoidun perusympäristön ohjelmistokehitykseen. CLR voidaan kuvata koodin prosessoijana, joka huolehtii koodin ajamisesta sekä yhteensopivuudesta käyttöjärjestelmän kanssa. .NET Framework luokkakirjasto sisältää puolestaan ison kokoelman standardoituja valmiita luokkia. Sisäänrakennettuja luokkia voidaan käyttää millä tahansa tuetulla ohjelmointikielellä .NET ympäristössä (kuvio 1). (Chappel 2006, 1–3.)



KUVIO 1. .NET Framework. Lähde: Singanamala 2012.

2.1 .NET versiot ja ohjelmointikielet

.NET Framework:in ensimmäinen versio 1.0 julkaistiin vuonna 2002. 1.0 sisälsi peruselementit, joita ovat CLR, .NET Framework sekä ASP.NET. .NET:n kehitys on jatkunut vuosien varrella ja viimeisin julkaistu versio on 4.5. Versiossa 4.5 on kehitetty esimerkiksi tietoturvaan, suorituskykyyn sekä prosessointiin liittyviä osa-alueita (kuvio 2) (Microsoft 2013i, hakupäivä 12.8.2013).

Overview of .NET Framework release history

Generation	Version number	Release date	Development tool	Distributed with
1.0	1.0.3705.0	2002-02-13	Visual Studio .NET	N/A
1.1	1.1.4322.573	2003-04-24	Visual Studio .NET 2003	Windows Server 2003
2.0	2.0.50727.42	2005-11-07	Visual Studio 2005	Windows Server 2003 R2
3.0	3.0.4506.30	2006-11-06	Expression Blend	Windows Vista, Windows Server 2008
3.5	3.5.21022.8	2007-11-19	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	2010-04-12	Visual Studio 2010	N/A
4.5	4.5.50709.17929	2012-08-15	Visual Studio 2012	Windows 8, Windows Server 2012

KUVIO 2. .NET release history. Lähde: Wikipedia 2013.

CLR suunniteltiin tukemaan useaa eri ohjelmointikieltä. Ohjelmointikieliä, jotka ovat tuettu .NET ympäristössä, ovat muun muassa C#, Visual Basic ja C++. C# kuuluu nimensä mukaan C-kieliperheessä oleviin ohjelmointikieliin. C# on olio-pohjainen kieli ja se suunniteltiin helposti lähestyttäväksi. (Chappel 2006, 77–79.) Kielen kehitys alkoi vuonna 1999, kun Microsoft päätti kehittää uuden kielen, joka tukisi täysin CLR:ä eikä sisältäisi muiden kielien tuomia rajoituksia. (Wikipedia 2013a, hakupäivä 6.9.2013.) C# suunniteltiin niin, että kuka tahansa, jolla on kokemusta C++- tai Java-kielistä, voi helposti omaksua kielen. Yleisin ohjelmointiympäristö C#-ohjelmistokehitykseen on Microsoftin Visual Studio. (Chappel 2006, 77–79.)

Visual Basic ja C#-kielet ovat lähellä toisia ja mahdollistavat lähes yhtä tehokkaat menetelmät ohjelmointiin. Tämän vuoksi kielen valinta näiden kahden väliltä perustuu usein omiin tottumuksiin. C# ja .NET versio Visual Basic:sta kehitettiin alun perin .NET Framework varten, toisin kuin C++-kieli, joka oli olemassa ennen .NET tuloa. (Chappel 2006, 110.)

Microsoft kehitti C++-kielestä oman version, joka on CLR:n perustuva. CLR-C++- sekä C++ -kielen tuomat haasteet johtivat Microsoftin lopulta kehittämään laajennuksen C++-kielelle. Laajennus tunnetaan nimellä Managed Extensions for C++. Kielen jo ennestään tunnettu haasteelli-

suus kasvoi edelleen Managed Extensions -laajennuksen tuomien ominaisuuksien myötä. Ohjelmistokehittäjille, joille C++ on ennestään tuttu, kieli on hyvä .NET ohjelmien kehitykseen. Uusien .NET ohjelmistokehittäjien tulisi luultavasti jättää C++ taakse ja aloittaa opiskeleminen C#-kielellä, joka on huomattavasti helpompi omaksua. (Chappel 2006, 130–131.)

2.2 ASP.NET

ASP.NET on ilmainen web-kehitykseen suunniteltu ohjelmistokehys, jota käytetään web-sivustojen ja ohjelmistojen kehityksessä. ASP.NET on osa .NET Framework:a. ASP.NET mahdollistaa ohjelmistokehityksen kielillä, jotka ovat tuettuna Common Language Runtime:ssa. Esimerkiksi ohjelmointi C#:lla sekä Visual Basic:llä mahdollistaa valmiiden luokkien periyttämisen sekä syntaksin tarkastuksen. (Microsoft 2013b, hakupäivä 14.8.2013.)

Olio-ohjelmointi on mahdollistanut laajojen ohjelmistojen rakentamisen tavalla, joka pitää koodin siistinä sekä helposti tulkittavana. ASP.NET mahdollistaa saman tavan webissä erottamalla ASP.NET koodin täysin web-sivuilta. Lisäämällä viittauksia web-sivuilla oleviin HTML-koodeihin, voidaan kutsua ASP.NET:ä rakentamaan esimerkiksi painikkeita ja tekstiä haluttuihin elementteihin. Tämän jälkeen ASP.NET:n luomien objektien ulkonäköä sekä sitä, mitä ne näyttävät, voidaan muokata. Objekteihin voidaan myös lisätä tapahtumia (event), jonka seurauksena suoritetaan haluttu toiminto käyttäjän klikatessa objektia. (Harper 2013a, hakupäivä 14.8.2013.)

ASP.NET sisältää Microsoftin kehittämän laajan luokkakirjaston**.** Kirjasto tarjoaa paljon hyödyllisiä valmiita luokkia, jotka nopeuttavat ja helpottavat ohjelmistokehitysprosessia. Esimerkiksi hakemalla tietoja tietokannasta ja näyttämällä tiedot näkymässä yksinkertaisessa grid-kontrollerissa. Perinteisellä ASP-menetelmällä tiedon hakemiseen ja näyttämiseen joutuisi kirjoittamaan paljon koodia. ASP.NET ohjelmoinnissa käyttäjän ei tarvitse kirjoittaa yhtään koodia tiedon näyttämiseen. Koodi liitetään (bind) DataGrid-objektiin, ja sivustolle koodataan viittaus, johon objektin tulisi asettua. DataGrid sisältää haetut tiedot tietokannasta ja tiedot näkyvät taulukkona viitatussa paikassa. (Harper 2013b, hakupäivä 14.8.2013.)

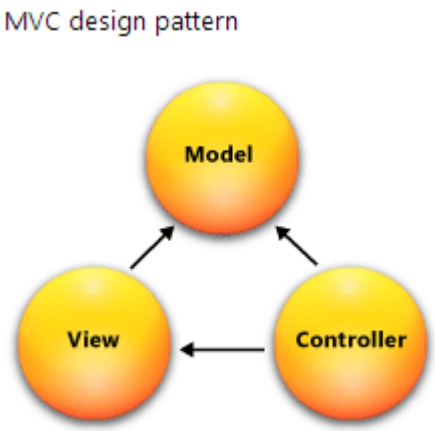
Yksi .NET:n suurimmista tavoitteista on ollut ASP.NET-sovellusten kehitys useammalla kielellä samanaikaisesti. Esimerkiksi puolet ryhmän jäsenistä voisi erikoistua VB.NET-kieleen ja toinen puoli kehittäisi ohjelmaa C#-kielellä. (Harper 2013c, hakupäivä 14.8.2013.)

3 ASP.NET MVC 4

ASP.NET MVC 4 on standardisoitu ohjelmistokehys web-kehitykseen. Versiosta julkaistiin Beta, RC sekä täysi versio vuoden 2012 aikana. Viimeisin versio on 4.0, joka julkaistiin toukokuussa vuonna 2013. ASP.NET MVC 4:ssä on kehitetty muun muassa mobiililaitteille tulevia palveluja ja toimintoja. (Microsoft 2013e, hakupäivä 4.9.2013.)

3.1 MVC-arkkitehtuuri

MVC-arkkitehtuuri (Model-View-Controller) jakaa sovelluksen kolmeen pääkomponenttiin, joita ovat model, view ja controller (Microsoft 2013a, hakupäivä 14.8.2013). Suomessa osista käytetään nimitystä malli, ohjain ja näkymä (kuvio 3).



KUVIO 3. MVC design pattern. Lähde: Microsoft 2013.

Malliosioon kuvataan tapa, jota käytetään tehdessä hakuja tietovarastoihin. Malli voi olla yksinkertaisen näkymän malli, jota käytetään kuvaamaan siirrettäviä tietoja eri näkymiltä ohjaimille. Malli voi olla myös domain-malli, joka sisältää business-domainin tiedot, operaatiot, muunnokset sekä säännöt tietojen manipulointiin. Malliin kuvatut objektit muodostavat ohjelman tietojenhaussa käytettävän hakutavan. Mallin objektit tallentavat mallin tilan sekä suorittavat haut tietokantaan. (Freeman, Sanderson 2011, 61.)

Näkymä on osa, jossa kuvataan käyttäjän näkyvillä olevat tiedot. Yleensä näkymä rakentuu mallissa olevista tiedoista, mutta näkymä voidaan luoda myös täysin mallista riippumatta. Esimerkiksi

edit-näkymässä olevat pudotuslistat sekä tekstikentät voisivat rakentua riippuen mallissa olevien objektien tiloista. (Microsoft 2013a, hakupäivä 14.8.2013.)

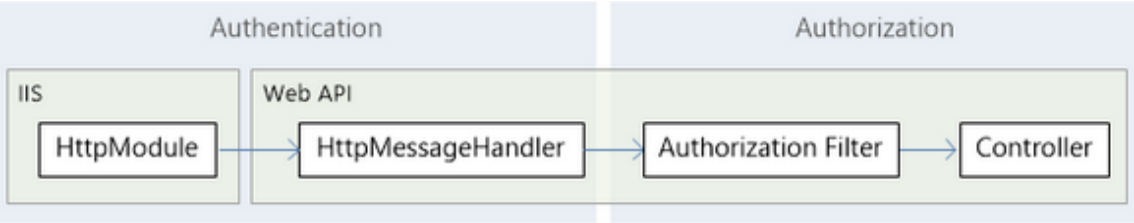
Ohjain on osa, joka huolehtii käyttäjän antamista syötteistä, ja joka suorittaa halutut operaatiot malliin. MVC-arkkitehtuurissa näkymässä kuvataan ainoastaan tiedot käyttäjälle. Ohjain vastaa ja vastaanottaa tiedot käyttäjältä ja suorittaa halutut operaatiot. Esimerkiksi ohjain käsittelee hakuehtojen parametrit ja välittää tiedot malliin. Tämän jälkeen mallissa voidaan suorittaa haku tietokantaan käyttäen välitettyjä parametreja. (Microsoft 2013a, hakupäivä 14.8.2013.)

MVC-arkkitehtuurin etuna on, että ohjelmiston koodit pysyvät erillään toisistaan. Ylläpidon ja päivittämisen kannalta koodin lisääminen ja muokkaaminen jälkeenpäin on yksinkertaisempaa. Jos tarvitaan muutoksia esimerkiksi näyttöihin, koodin lisääminen tapahtuu näkymä-osiossa. Mikäli tarvitaan muutoksia tiedon käsittelyssä, koodin muutos tehdään ohjaimeen. Jos muutos koskee tuettuja formaatteja, muutos tehdään malliin.

3.2 Web API

Web API on ASP.NET MVC 4:n tuoma uusi ohjelmistokehys HTTP-palveluiden rakentamiseen. Uuden Web API:n ominaisuuksia voidaan käyttää eri selaimilla sekä mobiililaitteilla. Web API on erinomainen alusta RESTFUL-palveluiden rakentamiseen. Uudessa Web API:ssa tuettuja toimintoja ovat muun muassa *filters*, *model binding and validation*, *full support for routes* ja *content negotiation*. (Microsoft 2013c, hakupäivä 16.8.2013.)

Filters sisältää esimerkiksi laajalti tunnetun Authorize-attribuutin. Tämän ominaisuuden avulla voidaan rakentaa autorisointi-moduuleja ja lisätä näitä haluttujen metodien autorisointiin sekä virheiden hallintaan. (Microsoft 2013c, hakupäivä 16.8.2013.) Autorisointi tapahtuu ohjaimen (controller) lähellä, joten sitä käyttämällä voidaan tehokkaasti rajoittaa palvelun tarjoamia resursseja (kuvio 4).



KUVIO 4. Authorization. Lähde Microsoft 2013.

Authorize-attribuuttia käyttämällä voidaan rajoittaa resursseja yksittäisiltä käyttäjiltä tai rooliperustaisessa autorisoinnissa kokonaisia ryhmiä. Ominaisuutta käytetään lisäämällä hakasuluissa oleva attribuutti metodin yläpuolelle. Lisäksi attribuuttiin lisätään parametrina rajoitukset, jotka halutaan asettaa metodille (kuvio 5). (Microsoft 2013h, hakupäivä 6.9.2013.)

```
// Restrict by user:
[Authorize(Users="Alice,Bob")]
public class ValuesController : ApiController
{
}

// Restrict by role:
[Authorize(Roles="Administrators")]
public class ValuesController : ApiController
{
}
```

KUVIO 5. Authentication and Authorization in ASP.NET Web API. Lähde: Microsoft 2013.

Kustomoituja omia filtereitä voidaan rakentaa esimerkiksi periyttämällä AuthorizationFilterAttribute-luokka. Luokan sisälle rakennetaan haluttu toiminnallisuus, ja attribuutti lisätään rajoitetun metodin yläpuolelle. Tämä tarkoittaa, että palvelimella voisi olla esimerkiksi HTTP- ja HTTPS-yhteyden vaativia metodeja. Seuraavassa esimerkissä havainnollistetaan, miten HTTPS-yhteyden vaativa luokka rakennetaan (kuvio 6).

```
public class RequireHttpsAttribute : AuthorizationFilterAttribute
{
    public override void OnAuthorization(HttpActionContext actionContext)
    {
        if (actionContext.Request.RequestUri.Scheme != Uri.UriSchemeHttps)
        {
            actionContext.Response = new HttpResponseMessage(System.Net.HttpStatusCod
            {
                ReasonPhrase = "HTTPS Required"
            };
        }
        else
        {
            base.OnAuthorization(actionContext);
        }
    }
}
```

KUVIO 6. Enforcing SSL in a Web API Controller. Lähde: Microsoft 2013.

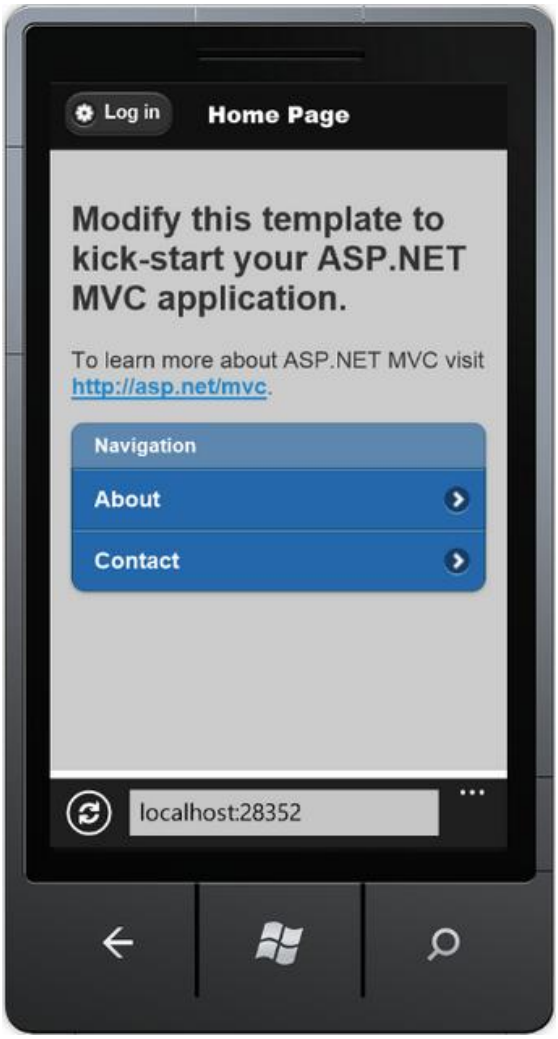
Model binding and validation:n avulla voidaan erotella HTTP-kutsuista osia ja kääntää näiden kutsujen osat .NET-objekteiksi. Tämän jälkeen objektit ovat Web API -metodien käytössä. Validointi suoritetaan metodien parametreille riippuen data annotation:sta. (Microsoft 2013c, hakupäivä 16.8.2013.)

Full support for routes tarkoittaa, että Web API -reititys on täysin tuettu ominaisuuksilla, jotka ovat aina olleet osa ASP.NET:n reititystä. Lisäksi viittaus (mapping) metodeihin on tuettu, mikä tarkoittaa, että enää ei tarvitse käyttää luokkien sekä metodien alkupäässä esimerkiksi [HttpPost] attribuuttia. (Microsoft 2013c, hakupäivä 16.8.2013.)

Formaatin valinta (Content Negotiation) tarkoittaa, että palvelin ja client voivat yhdessä määritellä sopivan formaatin Web API:sta palautettavalle tiedolle. Web API on oletuksena tuettu muun muassa XML-, JSON- sekä Form URL-encoded-formaateille. Lisäksi on mahdollista käyttää omia formaatteja tai korvata kokonaan oletuksena oleva formaatin neuvottelustrategia. (Microsoft 2013c, hakupäivä 16.8.2013.)

3.3 Uusi mobiiliprojektimalli, näkymä ja renderointi

Uusi mobiililaitteille suunniteltu malli mahdollistaa web-sivujen luonnin nopeasti ja helposti. Uusi malli perustuu suosittuun jQuery Mobile -kirjastoon. jQuery Mobile on avoimen lähdekoodin kirjasto, joka on suunniteltu kosketusnäyttölaitteiden ohjelmistokehitykseen. Seuraavassa esimerkissä on käytetty samaa rakennetta, jota tavallinen internetsivuston oletusmalli käyttää. Ohjaimen koodi on samanlainen kuin tavallisessa oletusmallissa. UI:n tyyli on muokattu jQuery Mobilea käyttämällä ja toiminnallisuus on samanlainen kuin kosketusnäyttölaitteissa (kuvio 7). (Microsoft 2013e, hakupäivä 20.8.2013.)

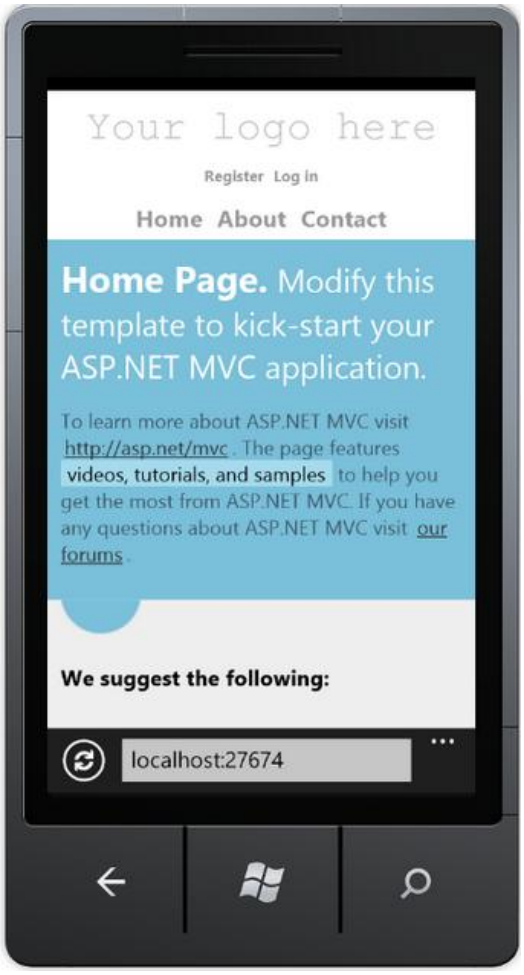


KUVIO 7. Mobile Project Template. Lähde: Microsoft 2013.

Uudella *display mode* -ominaisuudella voidaan hyödyntää jo olemassaolevia työpöytäkoneiden selaimille suunniteltuja palveluja. Pyynnön tehneestä selaimesta riippuen display modella voidaan määrittää tulostettava näkymä. Esimerkiksi näkymä voidaan palauttaa työpöytäkoneiden selaimil-

le sijainnista *Views\Home\Index.cshtml*. Mikäli mobiiliselain on tehnyt kutsun, palautetaan selaimeen näkymä sijainnista *Views\Home\Index.mobile.cshtml*. (Microsoft 2013e, hakupäivä 20.8.2013.)

Kosmeettisten ominaisuuksien lisäksi uusissa malleissa on uusi toiminnallisuus. Uudesta toiminnallisuudesta käytetään termiä *mukautuva renderointi* (*adaptive rendering*). Mukautuvan renderoinnin tavoitteena on, että sivusto näyttää sekä työpöytäkoneen että mobiililaitteen selaimessa yhtä hyvältä ilman ylimääräisiä muokkauksia. Uuden renderoinnin avulla näkymä muuttuu helpommin luettavaksi riippuen selaimen koosta (kuvio 8). (Microsoft 2013d, hakupäivä 20.8.2013.)



KUVIO 8. New rendering technique. Lähde: Microsoft 2013.

3.4 Paranneltu asynkronisten metodien tuki

Asynkronisten metodien käyttö esiteltiin ensimmäisen kerran .NET 4 Framework:in versiossa. Asynkronista menetelmää kutsutaan *Task*-menetelmäksi. ASP.NET MVC 4:n ohjainluokka yhdistettynä .NET 4.5 version kanssa mahdollistaa asynkronisten metodien käytön. .NET 4.5 Framework tukee asynkronista menetelmää, johon kuuluvat sekä await- että async-attribuutit. Await- sekä async-attribuuttien käyttö helpottaa Task-objektien kanssa työskentelyä. Await-attribuuttilla viitataan koodia odottamaan asynkronisesti jotain toista osaa koodista. Async-attribuutilla merkitään metodi viittaamaan asynkroniseen metodiin. (Microsoft 2013f, hakupäivä 21.8.2013.)

Metodien valinnassa tulee huomioida, tarvitseeko kehitettävä ohjelmisto synkronia tai asynkronisia metodeja. Synkronisia metodeja suositellaan käytettäväksi, kun suoritettavat operaatiot ovat lyhyitä ja yksinkertaisia, tai yksinkertaisuus on tärkeämpää kuin tehokkuus. Synkronisia metodeja tulisi käyttää myös silloin, kun operaatiot ovat CPU-operaatioita, eivätkä operaatiot rasita verkkoa. Asynkronisia metodeja puolestaan suositellaan käytettäväksi, kun kutsutaan palvelua, joka voi käyttää hyväksi asynkronisia metodeja, ja kun käytössä on .NET 4.5 tai uudempi versio. Lisäksi asynkronisten metodien käyttäminen on kannattavaa, kun operaatiot kuormittavat verkkoa enemmän kuin CPU:ta, ja kun yhdenaikaisuus on tärkeämpää kuin yksinkertaisuus. Asynkronisten metodien käyttö tarjoaa myös mekanismin, jonka avulla loppukäyttäjä voi perua pitkään kestävän pyynnön (CancellationToken). (Microsoft 2013g, hakupäivä 21.8.2013.)

4 AUTENTIKOINTI JA AUTORISOINTI

Tietoturvallisen palvelun kehityksessä on hyvä pohtia, miten voidaan erotella oikeat ja väärät käyttäjät. Tällaista erottelua voidaan kutsua autentikoinniksi. Autentikoinnin tarkoitus on tunnistaa käyttäjä tai palvelu. Autentikointia seuraa yleensä autorisointi. Autorisointi verkossa tarkoittaa tunnistetun käyttäjän rajoittamista palvelussa. (Apple 2012, hakupäivä 9.10.2013.)

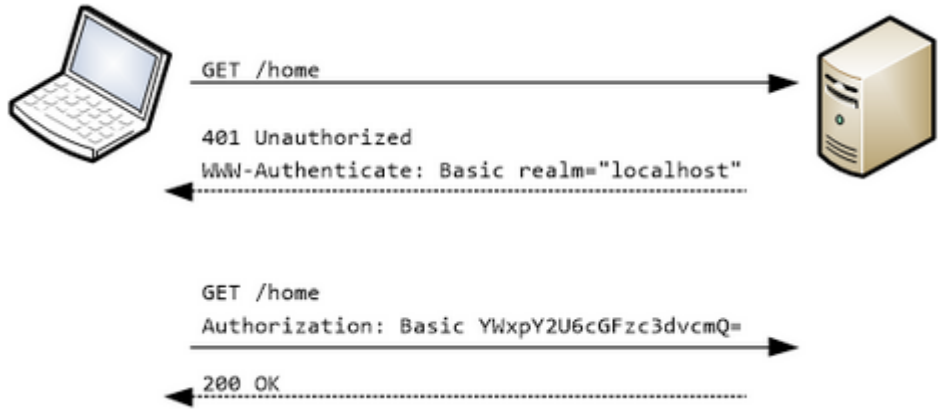
4.1 Autentikointi

Autentikointi on prosessi, jossa varmistetaan, pitääkö joku väittämä paikkansa. Sekä laite että laitteen haltija voidaan autentikoida. Laitteen autentikointi voidaan havainnollistaa seuraavan esimerkin avulla: Henkilö A haluaa mennä huvipuistossa laitteeseen, jossa on pituusrajoitus. Laitteeseen pääsy voidaan varmistaa henkilön A:n pituudesta, ja tässä tapauksessa henkilö A:n nimellä ei ole merkitystä. (Kent & Millett 2003, 33.)

Autentikoinnin tarkoitus ei ole ainoastaan varmistaa tietty väittämä, vaan tietojärjestelmät autentikoivat käyttäjän myös tietoturvan tai jonkun muun vaatimuksen vuoksi. Autentikoinnin avulla voidaan muun muassa seurata käyttäjän tekemiä toimia sekä varmistaa, että käyttäjä on vastuussa tekemistään teoista. (Kent ym. 2003, 34.)

Basic-autentikointi on yksinkertainen tapa autentikoida loppukäyttäjä HTTP-palveluun. Käyttäjä autentikoidaan käyttäjätunnuksen ja salasanan avulla jokaiselle palvelimen alueelle erikseen. Palvelimen eri alueiden resurssit ovat käytössä ainoastaan, mikäli autentikointi alueelle onnistuu. (W3 Consortium, hakupäivä 3.10.2013.)

Basic-autentikoinnissa palvelin palauttaa käyttäjälle *unauthorized request* -viestin, jos pyynnön autentikointi ei onnistunut**.** Palvelimen palauttaessa unauthorized request -viestin siihen lisätään haaste (challenge), joka voisi olla esimerkiksi *WWW-Authenticate: Basic realm="localhost"*. Tässä tapauksessa käyttäjä tietää, että autentikointi suoritetaan basic-menetelmällä ja "localhost"-alueelle (kuvio 9). (W3 Consortium, hakupäivä 3.10.2013.)



KUVIO 9. Basic authentication flow. Lähde Microsoft 2013.

Käyttäjä voi palautuksen jälkeen lähettää palveluun käyttäjätunnuksen ja salasanan. Käyttäjätunnus ja salasana Base64-koodataan ja asetetaan Authorization header:n. Base64-koodaus tehdään tiedon eheyden vuoksi. Tämän jälkeen käyttäjä voi tehdä autorisointia vaativia kutsuja palveluun. (Microsoft 2013d, hakupäivä 3.10.2013.)

Basic-autentikoinnissa käyttäjätunnus ja salasana eivät ole kryptattu millään tavalla, jonka vuoksi tiedonsiirto ei ole tietoturvallinen. Tämän tekniikan kanssa suositellaan käytettäväksi salattua HTTPS-yhteyttä. (Microsoft 2013d, hakupäivä 3.10.2013.)

Basic-autentikoinnin vahvuuksia ovat internet standardi, yksinkertaisuus sekä se, että basic-autentikointi on tuettu lähes kaikissa selaimissa. Tekniikan huono puoli on, että käyttäjätunnus ja salasana lähetetään jokaisen pyynnön yhteydessä salaamattomana. (Microsoft 2013d, hakupäivä 3.10.2013.)

4.2 Autorisointi

Autorisointi on prosessi, jonka avulla varmistetaan vaadittujen sääntöjen täyttyminen, ennen kuin haluttu toiminto suoritetaan. Autorisointia voidaan havainnollistaa seuraavan esimerkin avulla: Ravintolassa on kaksi vessaa. Ensimmäisen vessan ovessa on kyltti *henkilökunta*. Toisen vessan ovessa on kyltti *asiakkaat*. Henkilökunnan vessassa on lisäksi lukko, joka estää pääsyn vessaan. Autorisointi tässä tapauksessa tarkoittaa, että henkilökunta on autorisoitu käyttämään henkilökunnan vessaa. (Kent ym. 2003, 36–41.)

Autorisointia ei aina liitetä yksittäisiin käyttäjiin, vaan käyttäjät pyritään ryhmittelemään autorisointitasojen mukaan. Eri tasoja voisivat olla esimerkiksi member-, admin- ja anonymous-tasot. Jakamalla käyttäjät eri tasoihin ja rajoittamalla tasojen käyttöoikeuksia, voidaan tehokkaasti rajoittaa resurssien käyttöä tietojärjestelmässä. (Kent ym. 2003, 36–41.)

ASP.NET -sovelluksessa käyttäjän autorisointi voitaisiin toteuttaa käyttämällä esimerkiksi authorize-attribuuttia (ks. 3.2 Web API). Authorize-attribuutin avulla käyttäjät voitaisiin autorisoida eri käyttäjäryhmien perusteella tai tehdä käyttäjäkohtainen autorisointi (Microsoft 2013j, hakupäivä 10.11.2013).

5 AUTENTIKOINTI JA AUTORISOINTI KÄYTÄNNÖSSÄ

Työ on toteutettu käyttämällä Visual Web Developer 2010 Express -ohjelmointiympäristöä. Palvelinpuolen ohjelmointi on toteutettu C#-kielellä ja käyttöliittymä on rakennettu HTML-kielellä. Selain, jota on käytetty demon kehityksessä, on Google Chromen versio 29.0.1547.66 m.

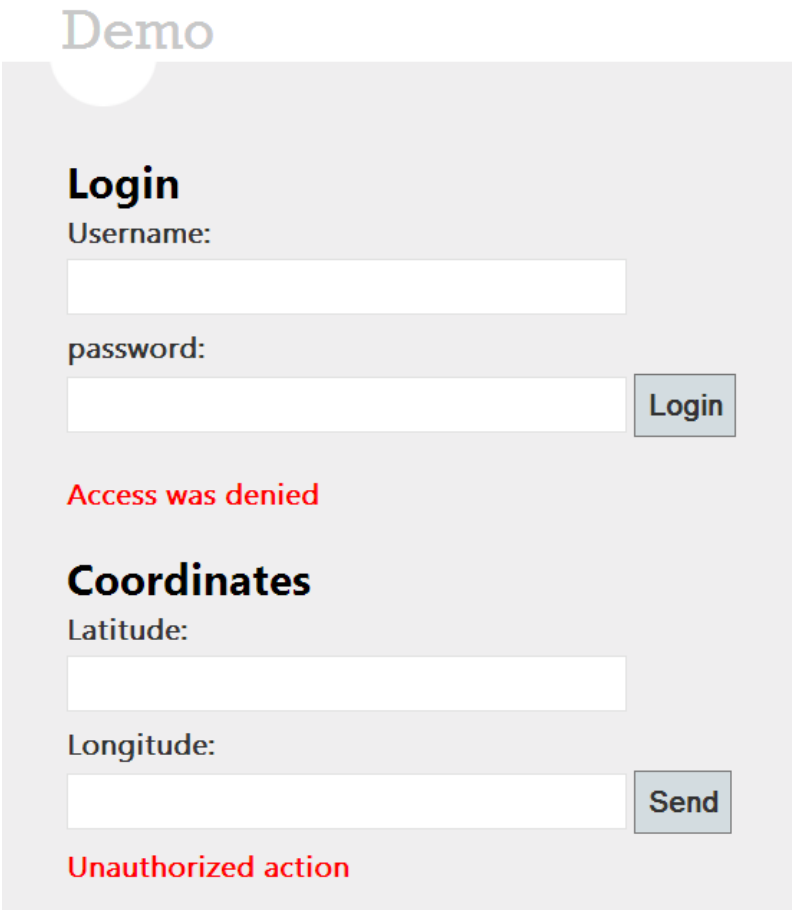
Työn toiminnallisuusvaatimuksia ovat web-sivu, jossa on JavaScript (jQuery) -toiminnallisuus. Lisäksi ohjelmisto vaatii HTTP-palveluun login-metodin, joka vastaanottaa käyttäjätunnuksen ja salasanan sekä saveCoordinate-metodin, joka vastaanottaa JSON-objektin. Työn toiminnallisuusvaatimuksiin kuuluu myös käyttäjän autentikointi ja autorisointi.

Työn autentikointi perustuu HTTP basic -autentikointiin. Valitsin basic-autentikoinnin, koska se on yksinkertainen toteuttaa, muokata ja jatkokehittää. Lisäksi basic-autentikointi voidaan toteuttaa täysin ilman käyttöliittymää. Tämä mahdollistaa esimerkiksi laitteen autentikoinnin yksinomaan.

Autentikoinnissa Base64-koodataan käyttäjän syöttämä käyttäjätunnus sekä salasana. Tiedot lähetetään kustomoidussa header-elementissä palvelimelle. Base64-koodausta ei pidä sekoittaa tietoturvaan. Koodaus tehdään ainoastaan tiedon eheyden vuoksi. Tiedot lähetetään suojaamattomana, jonka vuoksi lähetys täytyy suojata käyttämällä HTTPS-yhteyttä. Käyttäjätunnuksen ja salasanan tunnistamisen jälkeen palvelimella luodaan token autorisointia varten. Token palautetaan tunnistuksen jälkeen käyttäjälle. Käyttäjä voi tämän jälkeen liittää tokenin autorisointia vaativiin kutsuihin. Tämä tekniikka poikkeaa perinteisestä basic-autentikoinnista siten, että autentikoinnin jälkeen käyttäjä tunnistetaan tokenista eikä käyttäjätunnuksesta ja salasanasta. Jos yhteys jossain vaiheessa kaapataan ja tokenin tiedot menetetään ulkopuoliselle toimijalle, token on voimassa ainoastaan tietyn ajan ja väärinkäyttäjä voi käyttää palvelua vain ajan, jonka token on voimassa. Tällä tavalla voidaan minimoida menetykset sekä estää väärinkäyttäjää käyttämästä käyttäjätunnusta ja salasanaa jatkossa.

5.1 Työssä käytetty autentikointi

Käyttöliittymän avulla voidaan lähettää käyttäjätunnus ja salasana sekä leveys- ja korkeusaste. Leveys- ja korkeusasteen lähettäminen onnistuu vain autentikoidulta käyttäjältä. Virheilmoitus näytetään, kun käyttäjä lähettää autorisoimattoman kutsun tai virheellisen käyttäjätunnuksen ja salasanan (kuvio 10).



KUVIO 10. Fault messages.

Käyttäjätunnuksen ja salasanan lähetys palvelimelle on rakennettu käyttäen jQuery Ajax:a. Kuvissa 11 on esimerkki rakennetusta autentikointikutsusta.

```
function login() {
    var login = $('#logID').val();
    var password = $('#passID').val();
    var cred = login + ":" + password;
    var credentials = btoa(cred);

    $.ajax({
        beforeSend: function (xhr) {
            xhr.setRequestHeader("authenticationToken", credentials);
        },
        url: "https://localhost:44301/api/logins/login",
        type: "POST",
        success: function (res, status, xhr) {
            $("#token").html("token: " + xhr.getResponseHeader("authenticationToken"));
            sessionToken = xhr.getResponseHeader("authenticationToken");
        },
        statusCode: {
            200: function (result) {
                $("#autheSuccess").html(result);
                $("#autheFault").html("");
            },
            403: function (result) {
                $("#autheFault").html("Access was denied");
                $("#autheSuccess").html("");
            }
        }
    });
}
```

KUVIO 11. Login post message.

Käyttäjätunnus ja salasana Base64-koodataan ja asetetaan kustomoituun header-elementtiin. Kutsu lähetetään palvelimelle url:n osoittamaan metodiin. Login-metodissa tarkastetaan lähetetty käyttäjätunnus ja salasana. Käyttäjätunnuksen tai salasanan ollessa virheellinen, palvelimelta lähetetään vastauskoodi *403 Forbidden status*. Onnistuneen autentikoinnin jälkeen käyttäjälle palautetaan palvelimella luotu token sekä vastauskoodi *200 Ok status*. Tokenia voidaan tämän jälkeen käyttää autorisointia vaativien kutsujen suorittamiseen.

Palvelimen login-metodi vastaanottaa käyttäjän lähettämän autentikointikutsun. Metodissa tarkastetaan, onko header-elementtiin asetettu autentikointi token. Mikäli on, käyttäjätunnus ja salasana puretaan ja niitä verrataan tietokannassa oleviin käyttäjätunnuksiin. Jos tietokannasta löytyy vastaava käyttäjätunnus ja salasana, luodaan autorisointiin käytettävä token. Kuviossa 12 on havainnollistettu, miten token luodaan käyttäen Globally Unique Identifier -luokkaa.

```
Guid guid = Guid.NewGuid();
string str = guid.ToString();
var byt = Encoding.UTF8.GetBytes(str);
var to64 = Convert.ToBase64String(byt);
```

KUVIO 12. Guid token.

Tokenin luonnin jälkeen tietokantaan asetetaan autentikoidun käyttäjän tietoihin luotu token ja aikaleima, jolloin käyttäjä on autentikoitu. Lopuksi token asetetaan header-elementtiin ja se lähetetään takaisin käyttäjälle.

5.2 Työssä käytetty autorisointi

Koordinaattien lähetyksessä viestin header-elementtiin asetetaan palvelimelta saatu token. Viestin kuormaksi asetetaan JSON-objekti, joka sisältää lähetettävät koordinaatit. Lähetyksen jälkeen käyttöliittymässä näytetään, onko viestin lähetys onnistunut. Kuviossa 13 on havainnollistettu koordinaattien lähetyksessä käytetty Ajax-kutsu.

```
$.ajax({
    beforeSend: function (xhr) {
        xhr.setRequestHeader("authenticationToken", sessionToken);
    },
    url: "https://localhost:44301/api/logins/saveCoordinate",
    type: "POST",
    contentType: "application/json",
    data: JSON.stringify(coords),
    success: function (res, status, xhr) {
        $("#token").html("token: " + xhr.getResponseHeader("authenticationToken"));
    },
    statusCode: {
        200: function (result) {
            $("#authoSuccess").html("Authorized action completed");
            $("#authoFault").html("");
        },
        403: function (result) {
            $("#authoFault").html("Unauthorized action");
            $("#authoSuccess").html("");
        }
    }

});
```

KUVIO 13. SaveCoordinates post message.

Koordinaatit vastaanotetaan palvelimen saveCoordinate-metodissa (Kuvio 14). Metodin yläpuolelle on asetettu ValidateUser-attribuutti. ValidateUser on filter-attribuutti, joka suoritetaan ennen kuin viesti prosessoidaan ohjaimen saveCoordinate-metodissa.

```
[ValidateUser]
public HttpResponseMessage saveCoordinate(Person item) {

    if (item != null)
    {
        double longitude = item.longitude;
        double latitude = item.latitude;
        var response = Request.CreateResponse(HttpStatusCode.OK);
        return response;
    }
    var faultResponse = Request.CreateResponse(HttpStatusCode.Forbidden);
    return faultResponse;
}
```

KUVIO 14. SaveCoordinate method.

Liitteessä 1 on kuva rakennetusta ValidateUser-luokasta. ValidateUser-luokka on rakennettu periyttämällä AuthorizeAttribute-luokka. ValidateUser-luokassa vastaanotetun viestin header-elementti tutkitaan. Mikäli autentikointi-token löytyy, token puretaan ja sitä verrataan tietokannassa oleviin tokeneihin. Tokenin löydyttyä tietokannasta verrataan nykyhetkeä aikaleimaan, jolloin käyttäjä on autentikoitu. Vastaanotettu viesti hyväksytään, mikäli tietokannassa oleva aika on suurempi kuin nykyhetki. Hyväksynnän jälkeen tieto siirtyy ohjaimen saveCoordinate-metodiin ja toiminto suoritetaan. Mikäli aika on pienempi, kuin nykyhetki tai autentikointi-tokenia ei löydy tietokannasta, käyttäjälle palautetaan 403 forbidden status -koodi.

6 POHDINTA

Aloitin opinnäytetyöni kesäkuussa 2013. Teoriaosuutta kirjoitin kesä- ja heinäkuun ajan. Heinäkuun lopussa pidetyssä palaverissa ilmeni, että yrityksellä on käynnissä projekti, johon voisin soveltaa tietoturvaan liittyvää työosuuttani. Työ poikkesi kuitenkin sen verran kirjoittamastani teoriasta, että sitä ei voinut käyttää enää työn teoriaosuutena. Kirjoitin uutta teoriaosuutta samalla, kun tutustuin työosuudessa käytettäviin menetelmiin. Opinnäytetyö valmistui marraskuussa 2013.

Käyttäjän autentikointi on yksi tärkeimmistä asioista, joita tietoturvallisen palvelun tulisi sisältää. Käytännön osuuden aikana huomasin, että vaikka palvelu sisältäisi miten hyvän autentikoinnin tahansa, se on vain yksi osa tietoturvallisen palvelun kokonaisuudesta. Tietoturvallisessa palvelussa tulisi ottaa huomioon tasapuolisesti eri tietoturvan osa-alueet. Opinnäytetyön aikana opin ymmärtämään, mistä osista .NET koostuu ja mihin tarkoitukseen osia käytetään. ASP.NET MVC 4 soveltui hyvin autenikointipalvelun rakentamiseen, koska se tarjoaa useita erilaisia valmiiksi rakennettuja toiminnallisuuksia. Näitä toiminnallisuuksia voidaan muokata palvelua vastaavaksi. Opinnäytetyön käytännön osuuden myötä opin ymmärtämään MVC-arkkitehtuurin merkityksen ohjelmistokehityksessä.

Työosuudessa saatiin onnistuneesti toteutettua palvelu, johon käyttäjä voi tunnistautua ja tunnistautumisen jälkeen suorittaa autorisointia vaativia kutsuja. Tietoturvallisen autentikoinnin lisäämiseksi autentikointiprosessiin lisättiin token. Tokenin avulla käyttäjä voidaan tunnistaa ilman käyttäjätunnusta ja salasanaa. Tämä estää tilanteen, jossa käyttäjätunnus ja salasana ovat joutuneet ulkopuolisen käsiin. Tokenin eliniän hallinnan avulla väärinkäyttäjä voi käyttää palvelua vain tietyn ajan. Työn toimeksiantaja käyttää työssä käytettyjä menetelmiä apuna mobiilipalveluiden kehityksessä.

Web API julkaistiin vuonna 2012 ASP.NET MVC 4 -versiossa, jonka vuoksi autentikointiin ja autorisointiin liittyviä esimerkkejä oli niukasti saatavilla. Esimerkkejä kuitenkin oli jonkin verran, ja niiden avulla sain tietoja erilaisista tavoista rakentaa autentikointi ja autorisointi. Vaihtoehtoisia autentikointitapoja on useita, kuten basic-, forms-, Integrated Windows- ja OAuth 2.0 -autentikointi. Valitsin työhöni basic-autentikoinnin, koska sen toiminta on helppo ymmärtää ja sitä voidaan so-

veltaa erilaisiin tilanteisiin. Lisäksi se on internetstandardi, joka takaa sen toimivuuden lähes kaikissa selaimissa.

Autentikoinnin jatkokehityksen kannalta siihen voitaisiin lisätä parannuksia. Esimerkiksi tunnistetut käyttäjät voisi jakaa ryhmiin, jolloin pystyttäisiin tekemään ryhmäkohtainen autorisointi. Autentikointilogiikka voitaisiin siirtää omaan filter-luokkaan tai käsitellä tiedot HTTP message handler:ssä. Kaikki palvelussa vastaanotetut viestit tulisi tutkia ennen kuin ne pääsevät ohjaimessa oleviin metodeihin. Jatkokehityksen kannalta olisi myös hyvä huomioida basic-autentikoinnin haavoittuvuus Cross-Site Request Forgery (CSRF) -hyökkäyksille. Hyökkäyksien ehkäisyyn voitaisiin käyttää anti-forgery-tokenia.

LÄHTEET

Apple 2012, Authentication & Authorization. Hakupäivä 9.10.2012, https://developer.apple.com/library/ios/documentation/Security/Conceptual/Security_Overview/AuthenticationAndAuthorization/AuthenticationAndAuthorization.html

Chappel, D. 2006. Understanding .NET Second Edition. United States, Crawfordsville, Indiana.

Singanamala, C. B. 2012. .Net Framework. Hakupäivä 13.8.2013, http://dotnetoffice.wordpress.com/2012/01/17/net-framework/

Freeman, A. & Sanderson, S. 2011. Pro ASP.NET MVC 3 Framework, Third edition. Apress. Hakupäivä 14.8.2013, http://it-ebooks.info/book/476/

Harper, M. 2013a, OOP on the Internet. Hakupäivä 14.8.2013, http://www.javascriptkit.com/howto/aspnet.shtml

Harper, M. 2013b, Class Library. Hakupäivä 14.8.2013, http://www.javascriptkit.com/howto/aspnet.shtml

Harper, M. 2013c, Complete Compatibility. Hakupäivä 14.8.2013, http://www.javascriptkit.com/howto/aspnet.shtml

Kent, S. & Millett, I. 2003. Who Goes There? Authentication Through the Lens of Privacy. United States, Washington, DC. National Academies Press

Microsoft 2013a. ASP.NET MVC Overview. Hakupäivä 14.8.2013, http://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx

Microsoft 2013b, ASP.NET Overview. Hakupäivä 14.8.2013, http://msdn.microsoft.com/library/4w3ex9c2.aspx

Microsoft 2013c, New Features in ASP.NET MVC 4. Hakupäivä 16.8.2013, http://www.asp.net/whitepapers/mvc4-release-notes#_Toc317096197

Microsoft 2013d, Basic authentication. Hakupäivä 3.10.2013,
http://www.asp.net/web-api/overview/security/basic-authentication

Microsoft 2013e, Mobile Project Template. Hakupäivä 20.8.2013,
http://www.asp.net/whitepapers/mvc4-release-notes#_Toc317096197

Microsoft 2013f, Using Asynchronous Methods in ASP.NET MVC 4. Hakupäivä 21.8.2013,
http://www.asp.net/mvc/tutorials/mvc-4/using-asynchronous-methods-in-aspnet-mvc-4#ChoosingSyncVasync

Microsoft 2013g, Choosing Synchronous or Asynchronous Action Methods. Hakupäivä 21.8.2013, http://www.asp.net/mvc/tutorials/mvc-4/using-asynchronous-methods-in-aspnet-mvc-4#ChoosingSyncVasync

Microsoft 2013h, AuthorizeAttribute Class. Hakupäivä 6.9.2013,
http://msdn.microsoft.com/en-us/library/system.web.mvc.authorizeattribute(v=vs.108).aspx

Microsoft 2013i, What's New in the .NET Framework 4.5. Hakupäivä 12.8.2013,
http://msdn.microsoft.com/en-us/library/ms171868.aspx

Microsoft 2013j. Authentication and Authorization in ASP.NET Web API. Hakupäivä 10.11.2013,
http://www.asp.net/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api

Microsoft 2013. Authentication and Authorization in ASP.NET Web API. Hakupäivä 6.9.2013,
http://www.asp.net/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api

Microsoft 2013. Authorization. Hakupäivä 6.9.2013,
http://www.asp.net/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api

Microsoft 2013. Basic authentication flow. Hakupäivä 3.10.2013,
http://www.asp.net/web-api/overview/security/basic-authentication

Microsoft 2013. Enforcing SSL in a Web API Controller. Hakupäivä 8.9.2013, http://www.asp.net/web-api/overview/security/working-with-ssl-in-web-api

Microsoft 2013. MVC design pattern. Hakupäivä 14.8.2013, http://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx

Microsoft 2013. New rendering technique. Hakupäivä 20.8.2013, http://www.asp.net/whitepapers/mvc4-release-notes#_Toc317096197

Wikipedia 2013. .NET release history. Hakupäivä 13.8.2013, http://en.wikipedia.org/wiki/.NET_Framework

Wikipedia 2013a. History. Hakupäivä 6.9.2013, http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

W3 Consortium 1996. Basic Authentication Scheme. Hakupäivä 3.10.2013, http://www.w3.org/Protocols/HTTP/1.0/spec.html#AA

LIITTEET

ValidateUser-luokka. LIITE 1

```
public class ValidateUser : System.Web.Http.AuthorizeAttribute
{
    public override void OnAuthorization(System.Web.Http.Controllers.HttpActionContext actionContext)
    {


            if (actionContext.Request.Headers.GetValues("authenticationToken") != null)
            {
                // Haetaan tiedot headeristä
                string authenticationToken = Convert.ToString(actionContext.Request.Headers.GetValues("authenticationToken").FirstOrDefault());
                // Muutetaan base64 koodattu merkkijono stringiksi.
                string decodedToken = Encoding.UTF8.GetString(Convert.FromBase64String(authenticationToken));


                // Esimerkin "käyttäjätietokanta"
                Person[] persons = new Person[]{
                new Person{logName="admin",password="admin123",lastLogin="22.9.2013",tToken="1234-1234-1234-1234"},
                new Person{logName="user1",password="user1",lastLogin="7.8.2013",tToken="1234-1234-1234-1235"},
                new Person{logName="user2",password="user2",lastLogin="5.8.2013",tToken="1234-1234-1234-1236"}
            };

                // Etsitään löytyykö token tietokannasta.
                var person = persons.FirstOrDefault((p) => p.tToken == decodedToken);
                if (person == null) {
                    HttpContext.Current.Response.AddHeader("authenticationToken", authenticationToken);
                    HttpContext.Current.Response.AddHeader("AuthenticationStatus", "NotAuthorized");
                    actionContext.Response = actionContext.Request.CreateResponse(HttpStatusCode.Forbidden);
                    return;
                  }
                   // Verrataan session tokenia tietokannassa olevaan tokeniin.
                    if (decodedToken != person.tToken)
                    {
                        // Asetetaan viestit header-elementtiin ja lähetetään forbidden status takaisin.
                        HttpContext.Current.Response.AddHeader("authenticationToken", authenticationToken);
                        HttpContext.Current.Response.AddHeader("AuthenticationStatus", "NotAuthorized");
                        actionContext.Response = actionContext.Request.CreateResponse(HttpStatusCode.Forbidden);
                        return;
                    }

                    // Tarkistetaan onko tokenin aika mennyt umpeen.
                    string tokenTime = person.lastLogin;
                    string[] aTokenTime = tokenTime.Split('.');
                    int year = Convert.ToInt32(aTokenTime[2]);
                    int month = Convert.ToInt32(aTokenTime[1]);
                    int day = Convert.ToInt32(aTokenTime[0]);
                    DateTime oTokenTime=new DateTime(year,month,day,0,0,0);
                    DateTime timeNow = DateTime.Now;
                    int result = DateTime.Compare(oTokenTime,timeNow);

                    // Verrataan tietokannassa olevaa tokenia nykyhetkeen. Mikäli tietokannassa oleva aika
                    // on suurempi, toiminto on sallittu.
                    if (result == 0 || result > 0)
                    {

                        // Asetetaan viestit header-elementtiin ja lähetetään vastaus takaisin.
                        HttpContext.Current.Response.AddHeader("authenticationToken", authenticationToken);
                        HttpContext.Current.Response.AddHeader("AuthenticationStatus", "Authorized");
                        return;
                    }
                    else
                    {
                        HttpContext.Current.Response.AddHeader("authenticationToken", authenticationToken);
                        HttpContext.Current.Response.AddHeader("AuthenticationStatus", "NotAuthorized");
                        actionContext.Response = actionContext.Request.CreateResponse(HttpStatusCode.Forbidden);
                        return;
                    }

            }
            actionContext.Response = actionContext.Request.CreateResponse(HttpStatusCode.ExpectationFailed);
            actionContext.Response.ReasonPhrase = "Please provide valid inputs";
            base.OnAuthorization(actionContext);


    }
}
```