Juho Kauppi

jQuery Mobile -pohjainen mobiiliportaali

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

19.3.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Juho Kauppi jQuery Mobile -pohjainen mobiiliportaali 48 sivua 19.3.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaajat	yliopettaja Jaana Holvikivi tuotepäällikkö Sami Salmi

Insinöörityön tavoitteena oli tuottaa alustayhteensopiva mobiiliportaali, joka mahdollistaa yritysten tarjoamien palveluiden siirtämisen mobiililaitteilla käytettäviksi. Portaalin asiakaspuoli rakennettiin jQuery Mobile JavaScript -kirjastoa hyväksikäyttäen ja palvelinpuoli ASP.NET MVC 4 -ohjelmistokehyksen ympärille. Insinöörityössä käsitellään mobiiliportaalin toteuttamiseen käytettyä tekniikkaa ja mobiililaitteiden erityisvaatimuksia.

Työn alussa tutkittiin mobiiliympäristön vaatimuksia, asetettiin suunnittelutavoitteita ja perehdyttiin mobiiliportaalin toteutuksessa käytettyihin tekniikkoihin. Tavoitteiden yhteydessä tutustuttiin erilaisiin mobiilialustoihin, progressiiviseen web-kehitykseen, mobiililaitteiden ominaisuuksiin, lokalisointiin, suorituskykyä parantaviin tekniikoihin, ohjelmakoodin rakenteellisuuteen ja tietoturvan huomioimiseen web-ympäristössä. Tekniikoista esiteltiin asiakaspään ohjelmointiin käytetty JavaScript-ohjelmointikieli, selaimen DOM-puumalli, jQuery- ja jQuery Mobile -kirjastot ja palvelinpuolesta vastaava ASP.NET MVC -ohjelmistokehys. Samalla selitettiin jQuery Mobilen toimintaperiaate ja perusteltiin miksi ASP.NET MVC valittiin sen palvelinalustaksi.

Mobiiliportaalin toteutusta tarkasteltiin pääasiassa teknisestä näkökulmasta, mutta käyttöliittymän ja portaalin ominaisuuksien yhteydessä kiinnitettiin erityistä huomiota myös käytettävyyteen ja alustalle toteutettaviin palveluihin. Käyttöliittymässä keskityttiin visuaalisten elementtien esittelyyn, kun taas portaalin ominaisuuksissa keskityttiin esittelemään toteutettuja palveluita. Asiakaspään toteutuksessa käsiteltiin jQuery Mobilen tapahtumien ja komponenttien käyttöä. Samalla perehdyttiin asiakaspuolen arkkitehtuuriin ja progressiiviseen rakenteeseen. Palvelinpuolen toiminnoista esiteltiin AJAX-kutsujen käsittely ja yhteentoimivuus jQuery Mobilen kanssa. Muita käsiteltyjä palvelimen vastuualueita olivat portaalin lokalisointi, syötteiden validointi, tietoturva, käyttöoikeudet sekä tiedon tallennus ja synkronointi ulkoisiin järjestelmiin. Lopuksi tutustuttiin mobiiliportaalin testauksessa käytettyihin menetelmiin ja työkaluihin.

Insinöörityön tuloksena syntyi palvelualusta, joka mahdollistaa mobiilipalveluiden tuottamisen web-ympäristöön. Mobiiliportaali tarjoaa palveluille toimivan perusympäristön ja hyvät edellytykset jatkokehitykselle.

Avainsanat	mobiiliportaali, jQuery Mobile, ASP.NET MVC 4, Progressiivinen kehitys

Author	Juho Kauppi
Title	jQuery Mobile based mobile web portal
Number of Pages	48 pages
Date	19 March 2014
Degree	Bachelor of Engineering
Degree Program	Information Technology
Specialisation option	Software Engineering
Instructors	Jaana Holvikivi, Principal Lecturer Sami Salmi, Product Manager

The objective of the project described in this bachelor's thesis was to produce a cross platform mobile web portal that allows companies to transfer their services to be used with mobile equipment. The bachelor's thesis describes the technology used to implement the mobile web portal and addresses the special requirements of mobile equipment.

The work began with research for the requirements of mobile environment, setting the goals for the design and familiarizing with web technologies. Information was acquired about different mobile platforms and techniques needed for building secure and usable software in web environment.

The implementation of the mobile web portal was reviewed mainly from the technical point of view but also the usability and produced services had special emphasis when focusing on the user interface and the content of the portal. Client side functionalities introduced jQuery Mobile events and components. Server side responsibilities included localization, security and interoperability with jQuery Mobile. Finally the methods and tools used for testing the mobile web portal were documented.

In conclusion, a platform for producing mobile services in the web environment was produced. The mobile web portal offers a basic working environment for the services and supports the future development.

Keywords	Mobile Web, jQuery Mobile, ASP.NET MVC 4, Progressive enhancement

Sisällys

Lyhenteet

1 Johdanto 1

2 Tavoitteet 2

2.1 Yhteensopivuus 2
2.2 Käytettävyys 3
2.3 Suorituskyky 4
2.4 Ylläpidettävyys 4
2.5 Turvallisuus 5

3 Käytetyt tekniikat 6

3.1 JavaScript 6
3.2 DOM 7
3.3 jQuery 7
3.4 jQuery Mobile 8
3.5 ASP.NET MVC 10

4 Toteutus 13

4.1 Järjestelmä 13
4.2 Käyttöliittymä 14
4.3 Ominaisuudet 16
4.3.1 Kirjautuminen 16
4.3.2 Käyttäjien hallinta 17
4.3.3 Tilanne 19
4.3.4 Verkko 19
4.3.5 Tuki 20
4.4 Asiakas 21
4.4.1 Arkkitehtuuri 21
4.4.2 Progressiivinen kehitys 21
4.4.3 Tapahtumat 24
4.4.4 Komponentit 25
4.4.5 Dynaaminen sisältö 27
4.5 Palvelin 28
4.5.1 Arkkitehtuuri 28

4.5.2 AJAX 29
4.5.3 Tiedon tallennus 31
4.5.4 Validointi 32
4.5.5 Tietoturva 34
4.5.6 Lokalisointi 35
4.6 Testaus 36
4.6.1 Menetelmät 36
4.6.2 Palvelin 36
4.6.3 Emulaattorit 37
4.6.4 Työpöytäselaimet 39
4.6.5 Oikeat laitteet 42
5 Tulokset 44
Lähteet 46

Lyhenteet

.NET	Dot Net. Microsoftin ohjelmointialusta Windows-ympäristöön [17].
AJAX	Asynchronous JavaScript and XML. Kokoelma tekniikoita, joiden tarkoituksena on asiakkaan ja palvelimen välinen huomaamaton tiedonsiirto. [1, s. 221.]
ASP	Active Server Pages. Microsoftin tulkattava skriptikieli palvelinpuolen ohjelmointiin. [10, s. 2.]
CSS	Cascading Style Sheets. Web-sivuille kehitetty standardi muotoiluohjeista, jotka kertovat miten sivu tulee esittää. Uusin standardi on CSS 3, jota kehitetään moduuleittain [20].
DLL	Dynamic Link Library. Microsoftin toteutus jaetuista ohjelmakirjastoista.
DOM	Document Object Model. Ohjelmointirajapinta HTML- ja XML-dokumenteille. Uusin standardiehdotus DOM 4 [25].
IIS	Internet Information Services. Microsoftin palvelinalusta Windows-ympäristöön [18].
iOS	Applen mobiililaitteiden käyttöjärjestelmä, entiseltä nimeltään iPhone OS [1, s. 17].
JSON	JavaScript Object Notation. Kevyeen tiedonsiirtoon kehitetty puumallinen tietorakenne. Uusin standardi on ECMA-404 [24].
HTML	Hypertext Markup Language. Web-sivujen mallintamiseen tarkoitettu muotoilukieli. Uusin standardiehdotus on HTML 5 [23].
HTTP	Hypertext Transfer Protocol. Tiedonsiirtoprotokolla, johon WWW-sivujen siirto internetissä perustuu. Uusin standardi on HTTP 1.1 [22].
MVC	Model-View-Controller. Ohjelmistoarkkitehtuuri, jonka tarkoitus on erottaa käyttöliittymä sovelluslogiikasta.

SQL	Structured Query Language. Relaatiotietokantojen käsittelyyn suunniteltu ohjelmointikieli. Uusin standardi on SQL:2011 [19].
TLS/SSL	Transport Layer Security / Secure Sockets Layer. Salausprotokolla, jolla suojataan tiedonsiirto asiakkaan ja palvelimen välillä. Uusin standardi on TLS 1.2 [21].
XML	Extensive Markup Language. Tiedon käsittelyn suunniteltu tekstipohjainen kuvauskieli, jolla voidaan dokumentoida puumallisia tietorakenteita. Suositeltu standardi on XML 1.0 [16].
WWW	World Wide Web. Internetin kaikkien sivustojen kokonaisuus.

1 Johdanto

Insinöörityön aiheena on toteuttaa alustayhteensopiva mobiiliportaali yritysasiakaskäyttöön. Projektin asiakkaana toimii suomalainen tietotekniikka-alan yritys K&K Active Oy. Yritys on erikoistunut tietoliikennelaitteiden ja -ohjelmistojen maahantuontiin, myyntiin sekä asiakaskohtaisten ratkaisujen räätälöintiin.

Mobiililaitteiden yleistyessä siirtyvät yritysten tarjoamat palvelut yhä enemmän internettiin. Nykytrendi on toteuttaa jokaiselle mobiilialustalle oma ohjelmistonsa, jolla palveluita käytetään. Tämä hidastaa uusien palveluiden kehitystä ja kasvattaa käyttöönoton kustannuksia. Jos palvelu taas kohdistetaan vain tietylle mobiilialustalle, rajataan automaattisesti osa käyttäjistä palvelun ulkopuolelle. Eri maissa suositaan lisäksi erilaisia mobiililaitteita, joten kansainvälisille markkinoille on turha lähteä vain yhden alustan turvin. Palvelu, joka on käytettävissä ympäristöstä riippumatta, voittaa kilpailussa palvelun, jonka käyttö on sidottu tiettyyn ympäristöön.

Mobiiliportaalin on tarkoitus vastata tähän tarpeeseen ja tarjota alustayhteensopiva web-ohjelmisto mobiililaitteilla käytettävien palveluiden toteuttamiseen. Alustayhteensopivuus toteutetaan hyödyntämällä jQuery Mobile -kirjastoa, joka mukautuu mobiililaitteen ominaisuuksien mukaan ja mahdollistaa ominaisuuksien lisäämisen progressiivisesti. Palvelinpuoli toteutetaan ASP.NET MVC -ohjelmistokehyksellä, joka tukee jQuery Mobilen arkkitehtuuria ja tarjoaa turvallisen, helposti ylläpidettävän ja testattavan ohjelma-arkkitehtuurin.

Yritysten tarjoamat palvelut koostuvat yleensä useista taustajärjestelmistä. Mobiiliportaali tarjoaa mahdollisuuden taustajärjestelmien käyttöön niiden rajapintoja hyödyntämällä. Näin asiakkaan ei tarvitse tuntea taustajärjestelmien toteutusta käyttääkseen niiden palveluita. Mobiiliportaali integroidaan asiakasyrityksen myymään NETadmin-operaattoriverkonvalvontajärjestelmään. Integrointi mahdollistaa verkkolaitteiden, hälytysten ja palvelupyyntöjen käsittelyn mobiililaitteilla.

Jotta mobiiliportaali soveltuisi internet-palveluiden käyttöön, pitää sen suunnittelussa keskittyä helppokäyttöisyyteen, suorituskykyyn ja turvallisuuteen. Koska sen on tarkoitus toimia alustana erilaisille palveluille, tulee myös ylläpidettävyyteen kiinnittää erityistä huomiota. Portaali toteutetaan ketterää kehitystä hyväksikäyttäen. Suunniteltavat

ominaisuudet pilkotaan osiin ja niitä kehitetään yksi kerrallaan. Aina kun yksi ominaisuus on saatu valmiiksi, koko portaalia verrataan asetettuihin tavoitteisiin ja tarvittaessa muutetaan suunnitelmaa. Näin toteutus ohjautuu pienin askelin kohti tavoitetta. Insinöörityössä esitellään mobiililaitteille tarkoitettujen web-palveluiden kehitykseen tarvittavaa tekniikkaa ja työkaluja. Se antaa myös kuvan mobiiliympäristön mahdollisuuksista ja haasteista.

2 Tavoitteet

2.1 Yhteensopivuus

Projektin ensimmäinen tavoite oli tuottaa palvelualusta, joka toimii kaikilla mobiililaitteilla. Vaikka mobiilialustoja on vain kourallinen, on jokaisesta alustasta pahimmillaan kymmeniä ohjelmistoversioita ja jokainen niitä käyttävä laite on hieman erilainen. Tästä aiheutuu ongelmia suunnitellessa yhteensopivia ohjelmistoja.

Tärkeimmät mobiilialustat jakautuvat Android-, iOS- ja Windows Phone -käyttöjärjestelmiin ja niiden käyttämät ohjelmointikielet Java-, Objective C- ja .NET-kieliin. Lisäksi lähes jokainen valmistaja on kehittänyt oman käyttöjärjestelmänsä vanhempia puhelimia varten. Kieleltä toiselle kääntäminen ei onnistu kovin helposti, joten kaikilla alustoilla toimivan ohjelman luominen ja ylläpito olisi hyvin työlästä. [1, s. 16–31.]

Koska kaikki mobiilialustat sisältävät selaimen ja suurin osa näistä on vieläpä saman avoimen lähdekoodin WebKit-projektin johdannaisia, voidaan ongelma ratkaista hyödyntämällä selainta yhteisenä ohjelmistoalustana [1, s. 44]. Tällöin ohjelman käyttöliittymä rakennetaan kaikkien laitteiden tukemilla HTML-elementeillä ja CSS-määrittelyillä, ja dynaaminen toiminnallisuus toteutetaan joko palvelimella tai JavaScript-tekniikkaa tukevilla selaimilla.

Vaikka kaikki mobiilialustat osaavatkin tulkita HTML- ja CSS-muotoilukieliä, eivät ne välttämättä tue samoja toimintoja. Tällöin ohjelma voidaan rakentaa hyödyntämällä progressiivista kehitystä, jolloin perustoiminnot luodaan kaikkien selaimien tukemaan muotoon ja kehittyneemmät ominaisuudet rakennetaan kerroksittain näiden päälle. Kerroksittaisuus toteutetaan prosessoimalla HTML-muotoilukieli JavaScript-koodin

avulla ja käyttämällä useita CSS-määrittelyitä samanaikaisesti, jolloin yhteensopimattomat selaimet jättävät huomiotta ominaisuudet, joita ne eivät osaa käsitellä. [1, s. 63–64; 2, s. 55–57.]

2.2 Käytettävyys

Toinen projektin tavoitteista oli tuoda palvelut asiakkaan käytettäväksi ajasta ja paikasta riippumatta. Tämä edellyttää, että järjestelmä huomioi asiakkaan käyttökokemuksen ja tukee mobiilia käyttötapaa.

Mobiililaitteissa on niiden kannettavuusvaatimuksesta johtuen hyvin pieni näyttö ja koko ohjelman toiminnallisuus joudutaan ahtamaan tähän tilaan. Tällöin perinteiset käyttöliittymien suunnittelumallit eivät välttämättä istu mobiilimaailmaan suorilta käsin. Laitteet voivat näyttää sisällön myös eri tarkkuuksilla ja niitä voidaan käyttää vaaka- tai pystytasossa. Tällöin käyttöliittymän pitää mukautua käytettävän laitteen ominaisuuksiin. [1, s. 11–13.]

Yksi mobiililaitteiden käytettävyyden mullistavista ominaisuuksista on kosketusnäyttö, joka vaikuttaa näppäinten sijoitteluun, navigaatioon ja kaiken toiminnallisuuden toteutukseen. Kosketusnäytön avulla voidaan toiminnot muokata ihmiselle ominaisesti käytettävään muotoon. Kosketusnäytön yhteydessä tulee muistaa, että käyttöliittymän toimintoja käytetään sormella näpäyttämällä. Kosketuspinnat ovat huomattavasti suuremmat kuin hiirellä napsautettaessa ja virheellisten painallusten mahdollisuus kasvaa. Käyttöliittymän tulee myös huomioida erilaiset kosketuseleet, kuten vieritys ja tarkennus. [1, s. 70–71.]

Vaikka maailma onkin kansainvälistynyt, on oma kieli edelleen yksi käytettävyyden perusvaatimuksista. Lokalisointi parantaa ohjelmien käytettävyyttä ja mahdollistaa käyttäjäryhmät, joiden englannin kielen taito on rajoittunut. Lokalisointi mahdollistaa myös graafisen sisällön tai muun toimintalogiikan soveltamisen eri tavoin eri kansallisuuksille. [3.]

2.3 Suorituskyky

Ohjelman suorituskyky korostuu suunniteltaessa sovelluksia mobiililaitteille. Riittävä suorituskyky on suhteellinen käsite ja riippuu tavoitteesta. Mobiiliportaalin yhteydessä suorituskyvyksi määritettiin asiakkaan kokeman viiveen minimointi mobiiliportaalia käytettäessä.

Mobiililaitteiden suuri kehitysnopeus aiheuttaa markkinoille laajan valikoiman eritasoisia laitteita. Tämä pitää ottaa huomioon portaalin suunnittelussa. Ratkaisujen on oltava nopeita, eivätkä ne saa kuluttaa paljoa muistia, jos niiden halutaan toimivan kaikilla laitteilla. Web-käytössä laskentatehoa ja muistia kuluttavat pääasiassa JavaScript-tulkkaus ja graafinen sisältö. Itse HTML-sivu on harvoin kovin suuri, mutta huonosti suunniteltu tiedonhaku saattaa kuluttaa hyvinkin paljon muistia ja vaatia paljon laskentatehoa, kun selain yrittää sitä esittää. [1, s. 451–456.]

Toinen suorituskykyyn vaikuttava asia on laitteen yhteys internetiin. Suuri osa mobiililaitteilla ajettavista sovelluksista vaatii jatkuvan yhteyden palvelimeen. Palvelin on yhteydessä tietokantaan tai muihin järjestelmiin ja tuottaa sovelluksen tarvitsemaa dataa. Mobiililaitteen ja palvelimen välinen yhteysnopeus voi vaihdella suuresti ja hidas yhteys heijastuu välittömästi ohjelman suorituskykyyn ja siten sen käytettävyyteen. 3G-verkossa käytettävälle data-yhteydelle on lisäksi ominaista, että tietoliikenneyhteys ei ole jatkuvasti aktiivisena, vaan se käynnistetään tarvittaessa. Tämä aiheuttaa viiveitä ohjelman suorituskykyyn. [4.]

Mobiiliportaalin suorituskykyä voidaan parantaa hyödyntämällä AJAX-kyselyitä ja selaimen välimuistia. AJAX-kyselyillä voidaan hakea tietoa, käyttäjän edes huomaamatta tiedonsiirtoa [1, s. 221]. Staattinen sisältö voidaan tallettaa selaimen välimuistiin, josta se on heti saatavissa ilman, että palvelimelle tarvitsee lähettää erillistä kyselyä [2, s. 284; 5].

2.4 Ylläpidettävyys

Tuotteen elinkaaren määrää sen ylläpidettävyys. Ohjelmakoodin rakenne ratkaisee sen mahdollistamat muutokset. Huonosti suunniteltu ohjelma tukee vain yhtä toimintatapaa ja muutokset vaativat koko alla olevan ohjelmakoodin muuttamista. Projektin neljäs

tavoite onkin taata mobiiliportaalin hyvä muokattavuus ja mahdollistaa sen käyttö mahdollisimman monenlaisten palveluiden alustana.

Testattavuus on yksi ylläpidettävyyden haasteista. Kun vanhaa ohjelmakoodia joudutaan muuttamaan tai laajentamaan, on varmistettava, ettei vanha toiminnallisuus lakkaa toimimasta tai toimi väärin. Tällöin on jokainen muutoksen vaikutusalueella oleva toiminto testattava uudestaan. Tämä voi olla hyvin työlästä ja aikaa vievää. Testattavuus korostuu ketteriä ohjelmistokehitysmenetelmiä käytettäessä. Muutokset kuuluvat ketterän kehityksen luonteeseen, ja ohjelmiston toiminta on varmistettava jokaisessa muutoksessa. [12, s. 4–7.]

Toinen ylläpidettävyyden kulmakivistä on ohjelmakoodin rakenne ja luettavuus. Jotta ohjelmakoodi pysyy luettavana ja siten myös muokattavana, on se jaettava loogisiin kokonaisuuksiin ja sijoitettava omiin tiedostoihinsa, jotka on järjestetty rakenteellisesti helposti hahmotettavaan muotoon. Looginen rakenne minimoi muutosten suunnitteluun käytetyn ajan. Se varmistaa, ettei muualta koodista löydy mitään yllätyksiä, joita ei osata ottaa suunnittelussa huomioon. Oliopohjainen ohjelmointi on kehitetty tukemaan näitä tavoitteita, joten se soveltuu luontaisesti ylläpidettäväksi tähtäävän ohjelmiston perustaksi. [13, s. 19–20.]

2.5 Turvallisuus

Mobiiliportaalin mahdollistamat palvelut voivat käsitellä tietoa, jota ei saa luovuttaa kuin tietoihin oikeutetuille henkilöille. Tämänkaltaista tietoa ovat esimerkiksi käyttäjätunnukset, salasanat ja yritysten tai asiakkaiden luottamuksellinen tieto. Jotta tiedon eheys ja luottamuksellisuus säilyisivät, on sen säilytys ja käsittely mobiiliportaalissa suunniteltava tarkasti. Arkaluontoinen tieto on eristettävä käyttöoikeuksien avulla (valtuuttaminen) ja käyttäjä on tunnistettava tietoa käsitellessä (todentaminen).

Yksi suurimmista tietoturvariskeistä sisältyy käyttäjän todentamiseen. Käyttöoikeuksien rajaamisesta ei ole paljoa hyötyä, jos hyökkääjä pääsee tunnistautumaan toisena käyttäjänä. Riskiä voidaan pienentää turvaamalla salasanojen säilytys ja tunnistustietojen siirto asiakkaalta palvelimelle. Asiakkaan ja palvelimen välissä voidaan käyttää suojattua TLS/SSL-protokollaa, jolloin tunnistautumistiedot välitetään salattuna. Salasanat

voidaan myös säilyttää salattuna vahvalla yksisuuntaisella algoritmilla, jolloin salasanoista ei ole hyökkääjälle hyötyä, vaikka hän ne saisikin käsiinsä. [14, s. 320–329.]

Käyttäjän todentaminen tulee suorittaa aina, kun käsitellään arkaluontoista tietoa. Olisi epäkäytännöllistä ja riskialtista vaatia käyttäjää antamaan salasanansa joka kerta, kun hän suorittaa uuden toiminnon. Tähän tarkoitukseen voidaan käyttäjälle luovuttaa aikarajoitettu todennusavain, joka vahvistaa käyttäjän identiteetin. Web-ympäristössä tunnistusavainta säilytetään yleensä selaimen Cookie-tiedostossa, joka välitetään palvelimelle jokaisen pyynnön yhteydessä. [15, luku 11.2.]

Toinen suuri tietoturvariski web-kehityksen yhteydessä on Cross Site Scripting (XSS) ja muut injektiotekniikat. XSS tarkoittaa hyökkääjän web-sivulle syöttämää toiminnallisuutta, joka suoritetaan toisen käyttäjän selaimella. Tällöin käyttäjän selain toimii hyökkääjän antamien ohjeiden mukaan ja oikein toteutettuna käyttäjän tietoa päätyy hyökkääjän käsiin tai käyttäjä saadaan suorittamaan toimintoja hyökkääjän puolesta. Samantyyppistä hyökkäystä käytetään myös SQL-injektioissa, jolloin hyökkääjän syöttämä tieto sisältää SQL-komentoja, jotka suoritetaan palvelimen tallentaessa tietoja tietokantaan. Molemmat hyökkäykset perustuvat käyttäjän syötteisiin, joten tehokkain tapa estää niitä on tarkistaa kaikki käyttäjältä peräisin oleva data ja poistaa niistä selaimen tai palvelimen suorittama koodi. Syöte on tarkistettava kaikista lähteistä, jotka eivät tule järjestelmän sisältä. [11, 7. luku.]

3 Käytetyt tekniikat

3.1 JavaScript

JavaScript on tulkattava oliopohjainen ohjelmointikieli. JavaScript pohjautuu ECMAScript standardiin, joka määrittelee ohjelmointikielen tukemat ominaisuudet ja syntaksin [26]. Se eroaa hieman kirjoitusasultaan muista olio-ohjelmointikielistä, mutta mahdollistaa kaikki modernien ohjelmointikielien ominaisuudet omalla tavallaan toteutettuina [6, s. 3–4]. JavaScript-ohjelmakoodi tulkitaan ja suoritetaan suoraan selaimessa, joka myös tarkoittaa, että selain on vastuussa koodin suoritustavasta. Tästä johtuen kaikki selaimet eivät myöskään tue kaikkia JavaScript-käskyjä, vaan tuki riippuu toteutuksesta. [6, s. 13–15.]

JavaScript on tarkoitettu web-sivujen dynaamiseen muokkaamiseen asiakaspäässä. Normaalisti selain lataa web-sivun internetistä, tulkitsee HTML-muotoilukielen ja piirtää sivun selaimen ikkunaan. Samalla se suorittaa sivulla olevan JavaScript-koodin. Suoritettavalla koodilla on pääsy kaikkiin selaimen ikkunassa oleviin elementteihin. JavaScript-koodia voidaan myös kiinnittää selaimen tapahtumiin, jolloin näkyvä web-sivu muuttuukin asiakaspäässä suoritettavaksi ohjelmaksi.

Dynaamisten web-sivujen luonti perustuu JavaScriptin tapahtumiin ja HTML-sivun esittävään DOM-puuhun. JavaScriptillä voidaan hallita lähes kaikkia selaimen tapahtumia. Sillä voidaan tunnistaa käyttäjän toimet reaaliaikaisesti ja vastata niihin muokkaamalla sivua. [6, s. 9.]

3.2 DOM

Document Object Model (DOM) koostuu XML-muotoisesta, puurakenteisesta tiedosta ja kuvaa toiminnot tiedon hakemiseen ja käsittelyyn. Jokainen web-sivu voidaan esittää DOM-muotoisena rakenteena. Tällöin HTML-elementin kuvataan DOM-solmuina ja sen ominaisuuksina. DOM-elementit voidaan päivittää dynaamisesti, jolloin web-sivun ulkoasu muuttuu. Lähes kaikki ohjelmointikielet tukevat DOM-mallia. [6, s. 77–79.]

Selaimet käyttävät DOM-puuta tallettamaan ladatun web-sivun HTML-elementit ja attribuutit. DOM-puusta voidaan lukea kunkin HTML-elementin sisältö reaaliaikaisesti. Sisältöä muuttamalla muuttuu myös selaimen esittämän web-sivun ulkoasu. Tietoa voidaan hakea siirtymällä DOM-puussa elementti elementiltä eteenpäin tai hakemalla elementtejä niiden nimellä tai ID-tunnisteella. [6, s. 85.]

3.3 jQuery

jQuery on JavaScript-kirjasto, jonka tarkoitus on helpottaa DOM-objektien ja tapahtumien käsittelyä. jQuery-projektia kehittää jQuery Foundation, joka koostuu joukosta vapaaehtoisia web-kehittäjiä [27]. Kirjastossa on toteutettu dynaamisilla web-sivustoilla yleisimmin tarvittavia ominaisuuksia ja se on yksi yleisimmin käytettyjä JavaScript-kirjastoja. [7, 1. luku.]

Tärkein jQuery-kirjaston ominaisuus on mahdollisuus hakea HTML-elementtejä DOM-puusta käyttäen hyvin yksinkertaisia hakuehtoja. Hakuehtoihin sisältyy elementtien sisältö, sijainti, CSS-luokat ja attribuutit. Lisäksi hakuehtoja voi ketjuttaa, jolloin yhdellä rivillä koodia voidaan hakea tietty HTML-elementti varsin monimutkaisillakin ehdoilla. [7, 5. luku.]

Toinen tärkeä ominaisuus liittyy tapahtumien käsittelyyn. Eri selaimet käsittelevät tapahtumia hieman eri tavoin. jQuery huolehtii eri toteutuksista ja tarjoaa yhteisen rajapinnan tapahtumien käsittelyyn. Se myös antaa perinteistä JavaScript-tapahtumaa tarkempaa tietoa tapahtuman aiheuttajasta. [7, 9. luku.]

Muita jQuery-kirjaston hyödyllisiä ominaisuuksia ovat tuki erilaisille animaatioille, kuten siirtymille tai tiedon piilottamiselle sekä AJAX-komennot, joilla sivuille voidaan ladata sisältöä ilman, että sivua tarvitsee ladata uudestaan. jQuery-kirjastoa voidaan myös laajentaa lataamalla siihen liitännäisiä.

3.4 jQuery Mobile

jQuery Mobile on progressiiviseen suunnittelumalliin perustuva JavaScript-kirjasto. jQuery-kirjaston tavoin sitä kehittää jQuery Foundation [28]. Kirjasto laajentaa jQuery-kirjaston yksittäiset ominaisuudet koskemaan koko sivustoa ja mahdollistaa perinteisen HTML-sivun muuntamisen nykyaikaiseksi web-sivuksi jopa ilman uuden koodin kirjoittamista. Dynaamisen web-sivun toteuttaminen vaatii kuitenkin jQuery Mobilen arkkitehtuuriin perehtymistä. [7, 26. luku.]

jQuery Mobilen arkkitehtuuri perustuu sivujen lataamiseen asynkronisesti DOM-puuhun. Sivu vaihtuu ruudulle vasta, kun se on kokonaan ladattu ja vaihtumiseen voidaan käyttää animaatiota. Sivu voidaan myös säilyttää välimuistissa, jolloin sivua ei tarvitse ladata uudelleen, vaan se voidaan ottaa suoraan käyttöön. Tällöin syntyy illuusio yhtenäisestä ohjelmasta. [7, 27. luku.]

Kun jQuery Mobilea käytetään dynaamisen sisällön yhteydessä, tulee sivujen toteutuksesta hieman monimutkaisempaa. Koska kaikki mobiililaitteet eivät tue JavaScriptiä ollenkaan, tulee valita rakennetaanko tuki myös näitä laitteita varten. Jos laite ei tue JavaScriptiä, perustuu koko toiminnallisuus perinteiseen HTML-sivujen lataukseen.

Tällöin kaikki toiminnot tulee rakentaa palvelimelle lähetettävien lomakkeiden varaan. [8, 1. luku.]

JavaScriptiä tukevissa laitteissa vaihtoehtoja on paljon enemmän. Dynaaminen sisältö voidaan ladata erikseen AJAX-kyselynä ja liittää latauksen valmistuessa DOM-rakenteeseen. Sisältö voi olla HTML-muodossa, jolloin se voidaan ottaa suoraan käyttöön tai esimerkiksi JSON- tai XML-muodossa, jolloin se pitää jatkokäsitellä JavaScriptin avulla. Tiedon muoto riippuu siitä, halutaanko käsittelykuorma siirtää laitteelle vai suorittaa palvelimella. AJAX-kyselyillä tietoa voidaan hakea pieni pala kerrallaan, jolloin suorituskyky kasvaa kyselyiden nopeutuessa. Tietoa voidaan myös hakea suurempi kokonaisuus josta näytetään vain pala kerrallaan. Tämä soveltuu hyvin ei-reaaliaikaisen tiedon näyttämiseen. [7, 14. luku.]

jQuery Mobilen arkkitehtuurissa yksi sivu voi sisältää useampia jQuery Mobile -sivuja. Sivut ja niiden osat erotetaan toisistaan data-attribuuteilla, jotka ovat vapaasti määritettäviä HTML5 erikoistunnisteita. [8, 2. luku.] Esimerkkikoodissa 1 nähdään jQuery Mobile -sivun rakenne ja data-attribuuttien käyttö.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <link rel="stylesheet"
href="http://code.jquery.com/mobile/1.3.1/jquery.mobile-1.3.1.min.css" />
        <script src="http://code.jquery.com/jquery-2.0.3.min.js"></script>
        <script src="http://code.jquery.com/mobile/1.3.1/jquery.mobile-
1.3.1.min.js"></script>
    </head>
    <body>
        <div data-role="page" data-title="New Title" data-url="/Home/Index">
            <div data-role="header" data-id="header" data-theme="b">
                <h1>
                    This is the page header
                </h1>
            </div>
            <div data-role="content">
                <h2>
                    This is the page content
                </h2>
            </div>
        </div>
    </body>
</html>
```

Esimerkkikoodi 1. jQuery Mobilen sivurakenne.

Kun sivu ladataan, jQuery Mobile erottelee sivusta ainoastaan page-attribuutilla merkityt osat ja jättää muun sivun huomiotta. Koska useita sivuja säilytetään samassa DOM-puussa, tulee olla erityisen tarkkana, ettei kahdella sivulla käytetä samoja ID-tunnisteita. [7, 27. luku.]

3.5 ASP.NET MVC

ASP.NET on dynaamisten web-sivujen rakentamiseen tarkoitettu ohjelmointialusta. Alustaa suoritetaan Internet Information Services (IIS) -palvelimen päällä, joka huolehtii kutsujen välityksestä oikealle ohjelmalle. ASP.NET perustuu Microsoftin .NET-alustalle, joten se mahdollistaa saman koodin suorittamisen web-sovelluksissa, jota käytetään perinteisissä työpöytäsovelluksissa. [11, 1. luku.]

ASP.NET-alustasta on kaksi rinnakkaista kehityshaaraa: Web Forms ja MVC. Web Forms on ASP.NET:n perinteinen ohjelmointimalli. Sen tarkoitus on tehdä web-sovelluksien tekemisestä yhtä helppoa kuin työpöytäsovellusten kehityksestä ilman, että tarvitsee ymmärtää paljoakaan HTML-muotoilukielestä tai JavaScriptistä. [10, s. 2–3.] MVC-malli taas lähestyy kehitystä web-maailman tilattomasta näkökulmasta. WWW:n perustana oleva HTTP-protokolla on rakennettu tilattomaksi ja jokainen sivustoon kohdistuva kysely on itsenäinen eikä ole riippuvainen aiemmasta tilasta. ASP.NET:n MVC-malli tukee tätä lähestymistapaa ja keskittyy välittämään pyynnöt oikealle käsittelijälle. Tällöin voidaan tehokkaasti erottaa käyttöliittymän näkymä, toimintalogiikka ja käsiteltävä tieto. [10, s. 43–44.]

Koska Web Forms -ohjelmointimalli sisältää ohjelman tilan, ei se ole kovin yhteensopiva tilattomuuteen turvaavien ohjelmistokehysten, kuten jQuery Mobilen kanssa. Ongelmaksi muodostuu Web Formsin perusajatus sisällyttää tila jokaiseen kyselyyn. Koska jQuery Mobile hakee sisällön AJAX-kyselyillä ja poimii vain tarvittavan tiedon, syntyy ongelmia ID-tunnisteiden ja tilan sisältämän ns. ViewState-koodin kanssa. Tilan aiheuttamia ongelmia voidaan yrittää kiertää, mutta tällöin menetetään suurin osa Web Formsin hyödyistä ja lisäksi puutteet rajoittavat lähes kaikkea kehitystä. [10, s. 3–4.]

MVC-ohjelmointimalli on kuin tehty jQuery Mobilen palvelinalustaksi. Toiminnot on suunniteltu tilattomiksi, ja ohjelmoija saa vapaat kädet HTML-koodin kirjoittamiseen ja kyselyihin vastaamiseen. Kaikki MVC-mallin komponentit ovat vaihdettavia ja ohjelmoi-

ja pystyy muokkaamaan ne mieluisekseen. MVC-malli helpottaa myös ohjelmistotestausta, koska toiminta ei riipu tilasta ja kaikki komponentit voidaan testata erikseen. [10, s. 7–8.]

ASP.NET MVC jakaa sivuston useisiin osiin. Näkymä sijaitsee ohjelmointikielestä riippuen CSHTML- ja VBHTML-tiedostoissa (C# ja Visual Basic). Näkymätiedostot ovat HTML-muotoilukieltä, johon on upotettu mallin käsittelyä ohjeistavaa ohjelmakoodia. Esimerkki C#-kielisestä näkymätiedostosta nähdään esimerkkikoodissa 2.

```
@using MobilePortal;
@model MobilePortal.Models.LoginModel
<div data-role="page" data-title="@MobilePortal.Language.Translate("Mobile Por-
tal") - @Language.Translate("Login")" data-url="@Url.Action(null)">
    <div data-role="header" data-id="header" data-theme="b">
        <div class="portal-ui-logo ui-bar-b">
            <img src="~/Content/images/logo.png" alt="Company Logo" />
            <span>@Language.Translate("Mobile Portal")</span>
        </div>
    </div>
    <div data-role="content">
        <h2>@Language.Translate("Login")</h2>

        @using (Html.BeginForm())
        {
            <p>
                @Html.LabelFor(m => m.Username, Language.Translate("Username") +
":")
                @Html.EditorFor(m => m.Username)
                @Html.LabelFor(m => m.Username, Language.Translate("Password") +
":")
                @Html.EditorFor(m => m.Password)
                @Html.LabelFor(m => m.Username, Language.Translate("Select lan-
guage") + ":")
                @Html.DropDownListFor(m => m.Language, new
List<SelectListItem>() {
                    new SelectListItem() {
                        Text = Language.Translate("Finnish"),
                        Selected = Model.Language == "fi",
                        Value = "fi"
                    },
                    new SelectListItem() {
                        Text = Language.Translate("English"),
                        Selected = Model.Language == "en",
                        Value = "en"
                    },
                    new SelectListItem() {
                        Text = Language.Translate("Swedish"),
                        Selected = Model.Language == "sv",
                        Value = "sv"
                    }
                }, new { data_submit = "true" })
            </p>
            <p>
                @Html.ValidationSummary(false)
```

```
            </p>
            <p><input type="submit" value="@Language.Translate("Login")" /></p>
            @Html.HiddenFor(m => m.PreviousLanguage)
        }
    </div>
</div>
```

Esimerkkikoodi 2. C#-kielinen näkymätiedosto.

MVC-mallissa näkymä ainoastaan tuottaa sivun annettuun malliin perustuen. Itse toimintalogiikka ja mallit sijaitsevat omissa luokkatiedostoissaan. Ajoympäristö reitittää HTTP-pyynnöt kontrolleriluokille, jotka valitsevat tilanteeseen sopivan näkymän ja mallin. [11, 1. luku.] Esimerkit C#-kielisistä kontrolleri- ja malliluokista nähdään esimerkkikoodeissa 3 ja 4.

```
namespace MobilePortal.Controllers
{
    [Authorize(Roles = "admin")]
    public class AccountController : Controller
    {
        //
        // GET: /Account
        [AllowAnonymous]
        [HttpGet]
        public ActionResult Index()
        {
            if (Request.IsAuthenticated)
            {
                if (User.IsInRole("admin") && !User.IsInRole("user"))
                    return RedirectToAction("Administration", "Account");
                else
                    return RedirectToAction("Index", "Status");
            }
            else
            {
                var currentLanguage = Language.getCurrentLanguage();
                var model = new LoginModel() { Language = currentLanguage, Pre-
viousLanguage = currentLanguage };

                return View("Login", model);
            }
        }
    }
}
```

Esimerkkikoodi 3. C#-kielinen kontolleriluokka.

```
namespace MobilePortal.Models
{
    public class LoginModel
    {
        [Required]
```

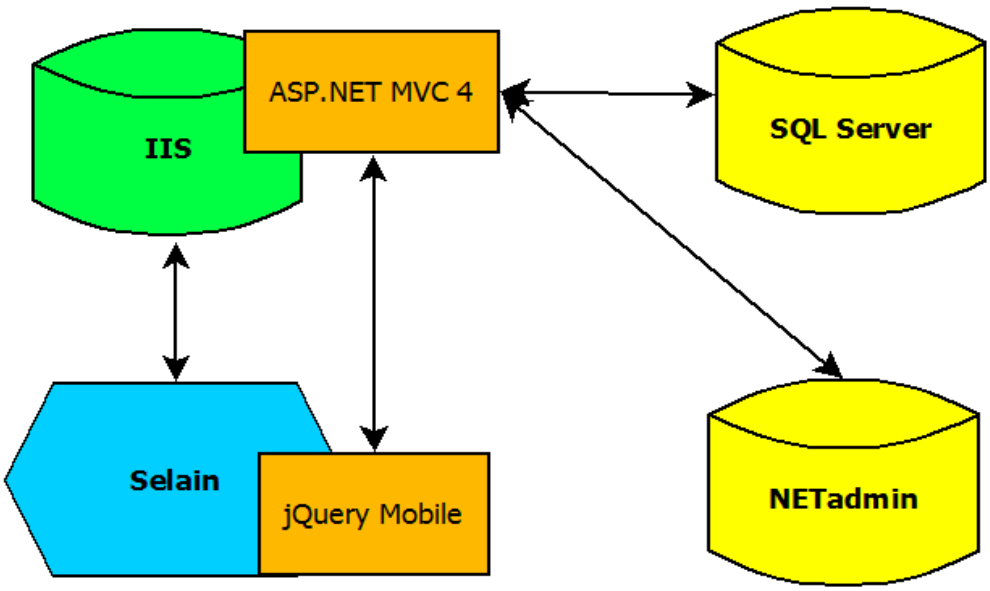
```
        public string Username { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        public string Language { get; set; }

        public string PreviousLanguage { get; set; }
    }
}
```

Esimerkkikoodi 4. C#-kielinen malliluokka.

4 Toteutus

4.1 Järjestelmä

Mobiiliportaali koostuu selaimella suoritettavasta jQuery Mobile -pohjaisesta käyttöliittymästä ja Microsoft Internet Information Services (IIS) -palvelimella suoritettavasta ASP.NET MVC 4 -ohjelmasta. Portaalin tarvitsema tieto haetaan paikallisesta SQL Server -tietokannasta ja ulkoisesta NETadmin-verkonvalvontajärjestelmästä. Järjestelmän osien yhteydet esitetään kuvassa 1.



Kuva 1. Järjestelmäkaavio.

4.2 Käyttöliittymä

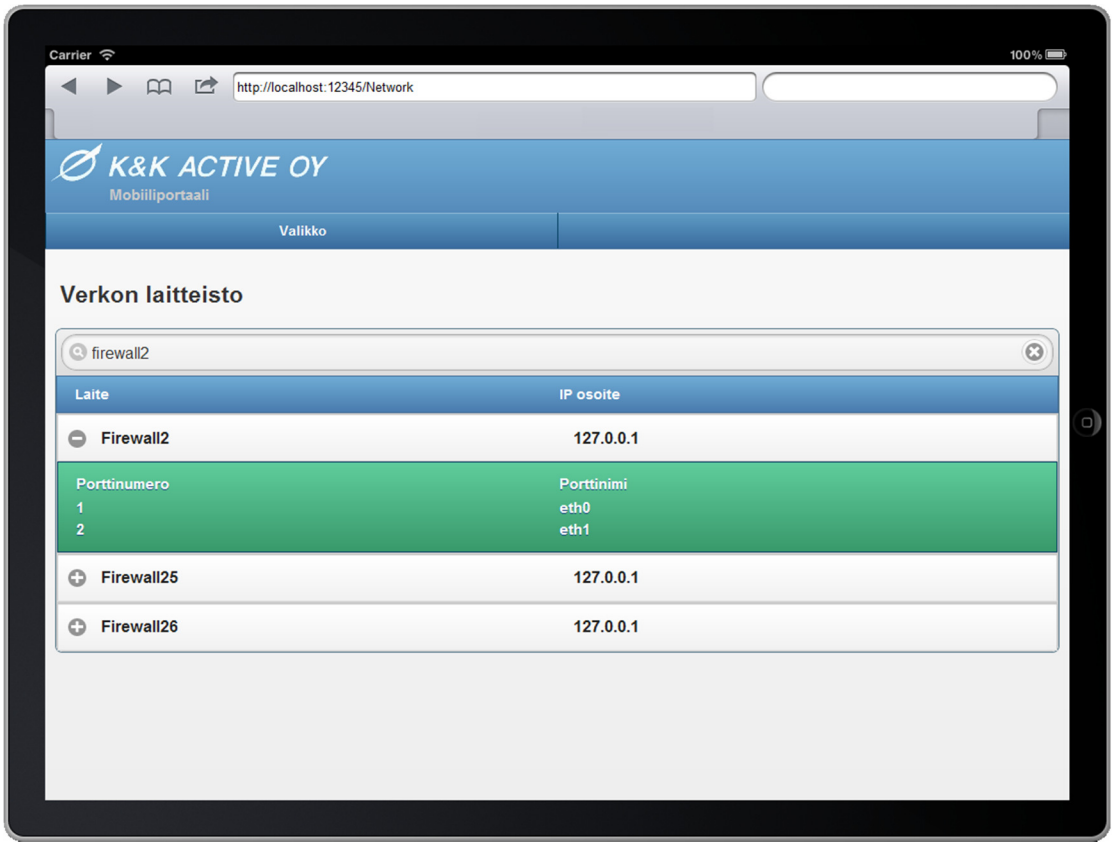
Mobiiliportaalin käyttöliittymä suunniteltiin mahdollisimman helppokäyttöiseksi ja muokattavaksi. Palvelut jaettiin omiin sivuihinsa, ja liikkuminen tapahtuu yhteisen navigaatiovalikon kautta. Navigaatiovalikko toteutettiin esiin aukeavana alasvetovalikkona, jolloin se vie vain vähän tilaa, mutta on aina käytettävissä, mahdollistaen nopean liikkumisen portaalin eri osien välillä. Valikko nähdään kuvassa 2. Useissa tilanteissa tarjotaan myös paluunappi valikon oikealle puolelle, jos looginen siirtyminen sitä edellyttää.

Kuva 2. Navigaatiovalikko.

Visuaalisesti käyttöliittymä koostuu erilaisista listoista, napeista ja tekstikentistä. Jotta portaali olisi käytettävissä myös pienemmän näyttökoon laitteilla, on esitettävä tieto jaettu kapeisiin elementteihin, jotka seuraavat toisiaan sivuilla. Kuvassa 3 nähdään esimerkkilistaus portaalin laitehälytyksistä ja tukipyyntölomake.

Kuva 3. Listaus laitehälytyksistä ja tukipyyntölomake.

Jotta tiedon määrä ei sekoittaisi käyttäjää, on aluksi esillä vain tarpeellinen yleistieto ja elementtejä näpäyttämällä saadaan esiin tarkempaa tietoa. Käyttöliittymä hyödyntää myös erilaisia hakuvalikoita, jolloin esiin voidaan näyttää vain haluttu määrä parhaiten sopivia vaihtoehtoja. Esimerkki hakutoiminnosta ja tarkentuvasta sisällöstä nähdään kuvassa 4. Kuvassa on haettu koko verkon laitteista Firewall2-hakusanan sisältävät laitteet ja hakutuloksista ensimmäinen on avattu tarkempaan tarkasteluun.



Kuva 4. Käyttöliittymän hakutoiminto ja tarkentuva sisältö.

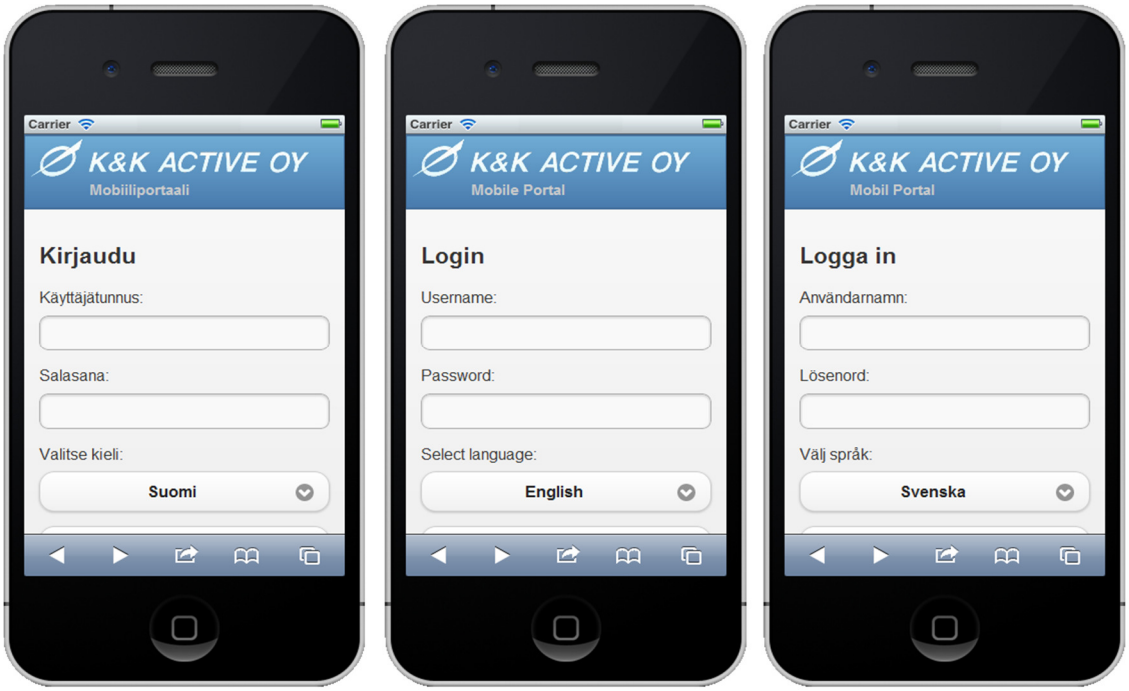
4.3 Ominaisuudet

Mobiiliportaali on suunniteltu toimimaan palvelualustana, joten sen sisältö määräytyy asiakkaan tarpeiden mukaan. Portaaliin on kuitenkin toteutettu muutamia palveluita esittelemään sen käyttömahdollisuuksia. Palvelut on tarkoitettu muokattaviksi käyttökohteesta riippuen.

4.3.1 Kirjautuminen

Kirjautuminen mobiiliportaaliin tapahtuu omalta sivultaan. Käyttäjä ei voi ohittaa kirjautumista, vaan portaali ohjaa tunnistautumattoman käyttäjän aina kirjautumissivulle. Portaali tunnistaa käyttäjän selaimen kieliasetuksen automaattisesti ja valitsee käytettävän kielen sen perusteella. Mobiiliportaali tukee oletuksena suomen, ruotsin ja englannin kieltä, mutta myös muita kieliä on mahdollista lisätä. Jos selaimen kielivalintaa ei portaalista löydy tai se halutaan muuten vain vaihtaa, löytyy kirjautumissivulta myös kie-

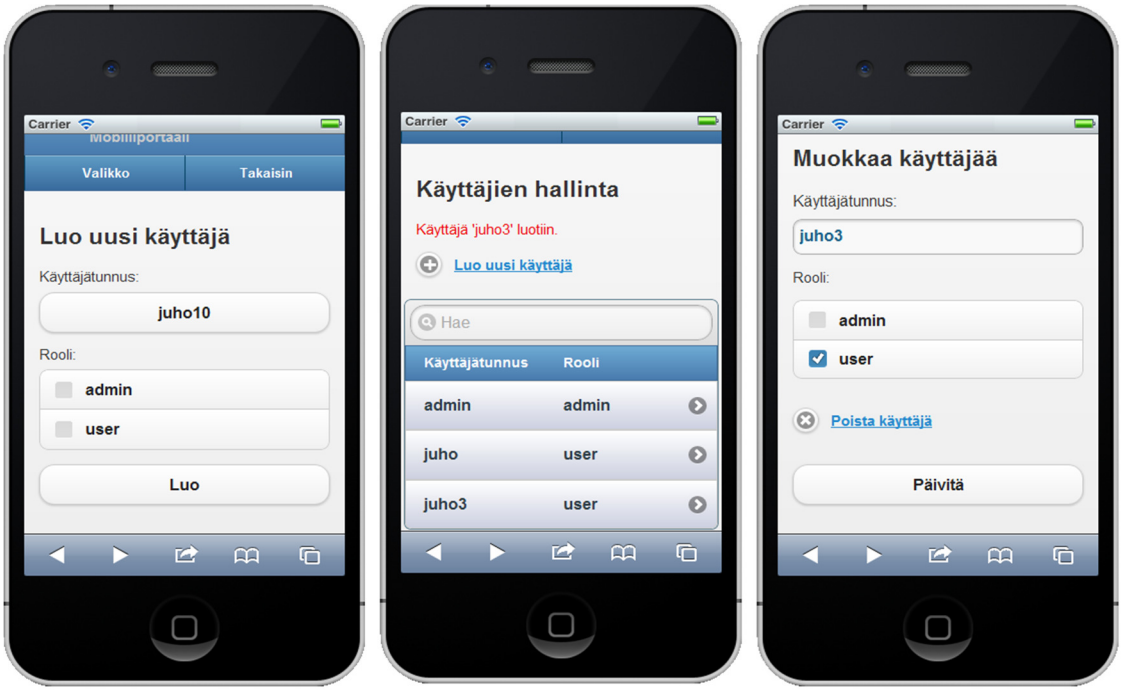
lenvalintapainike. Kuvassa 5 nähdään portaalin kirjautumissivu eri kielivalintoja käytettäessä.



Kuva 5. Kielivalinta kirjautumissivulla.

4.3.2 Käyttäjien hallinta

Käyttäjien hallintaa varten on portaaliin toteutettu hallintanäkymä, jossa voidaan lisätä, poistaa ja muokata portaalin käyttäjiä. Käyttäjät synkronoidaan ulkoisesta NETadmin-verkonvalvontajärjestelmästä, joten käyttäjän luonti tarjoaa ainoastaan käyttäjähaun ja roolien valinnan. Kuvassa 6 nähdään esimerkki uuden käyttäjän lisäyksestä ja hallintanäkymistä.



Kuva 6. Käyttäjän luonti ja hallinta.

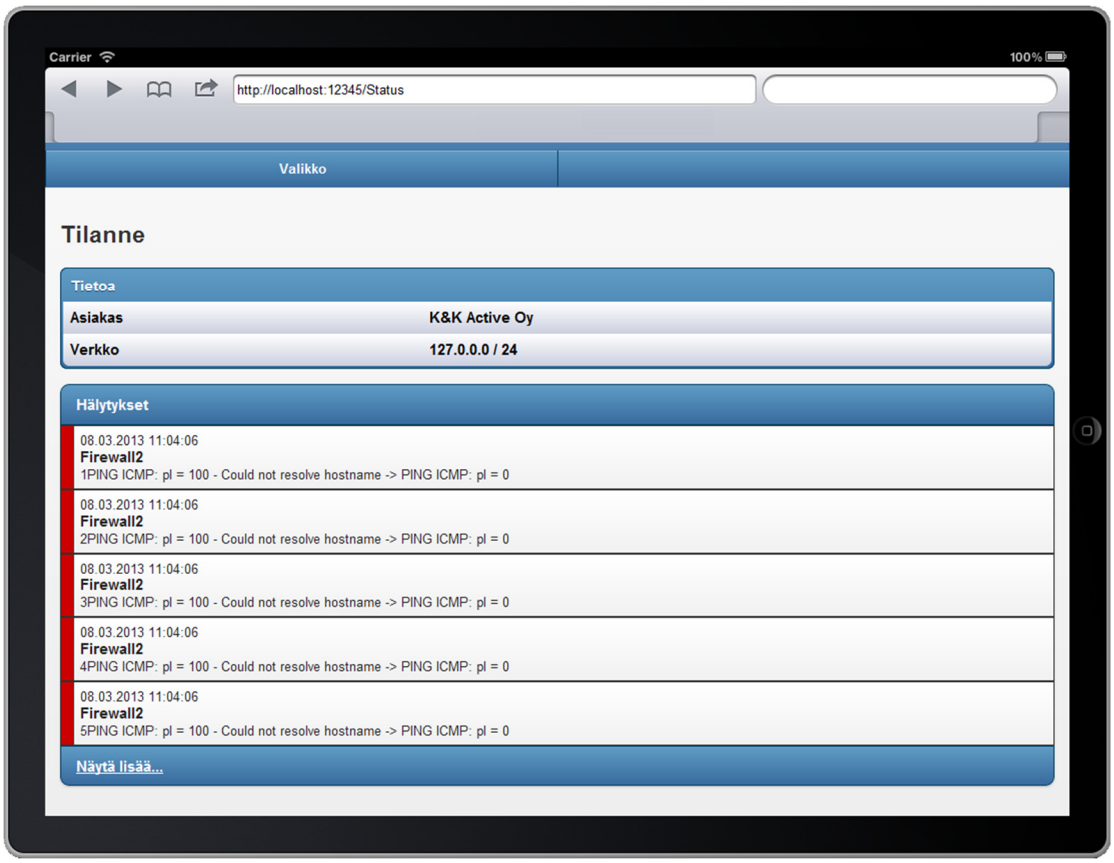
Portaalin sisältö mukautuu käyttöoikeuksien mukaan. Hallintasivulla asetettu käyttäjärooli määrää, mitä toimintoja käyttäjä voi suorittaa ja minkälaista sisältöä hänelle näytetään. Käyttöoikeuksien mukaan mukautuva valikko nähdään kuvassa 7.

Kuva 7. Käyttöoikeuksien mukaan mukautuva valikko.

Kuvan ensimmäisessä näkymässä on käyttäjällä pelkät hallintaoikeudet ja toisessa pelkät käyttäjänoikeudet.

4.3.3 Tilanne

Tilanne-sivun tarkoitus on antaa asiakkaalle palvelun tilannekuva. Sivulla esitetään asiakkaan palvelutietoja, kuten verkko-osoite ja listaus asiakkaan laitteita koskevista hälytyksistä. Kuvassa 8 nähdään portaalin tilannenäkymä. Näytä lisää -painike lataa esiin lisää hälytyksiä AJAX-pyyntöinä.



Kuva 8. Tilanne-näkymä.

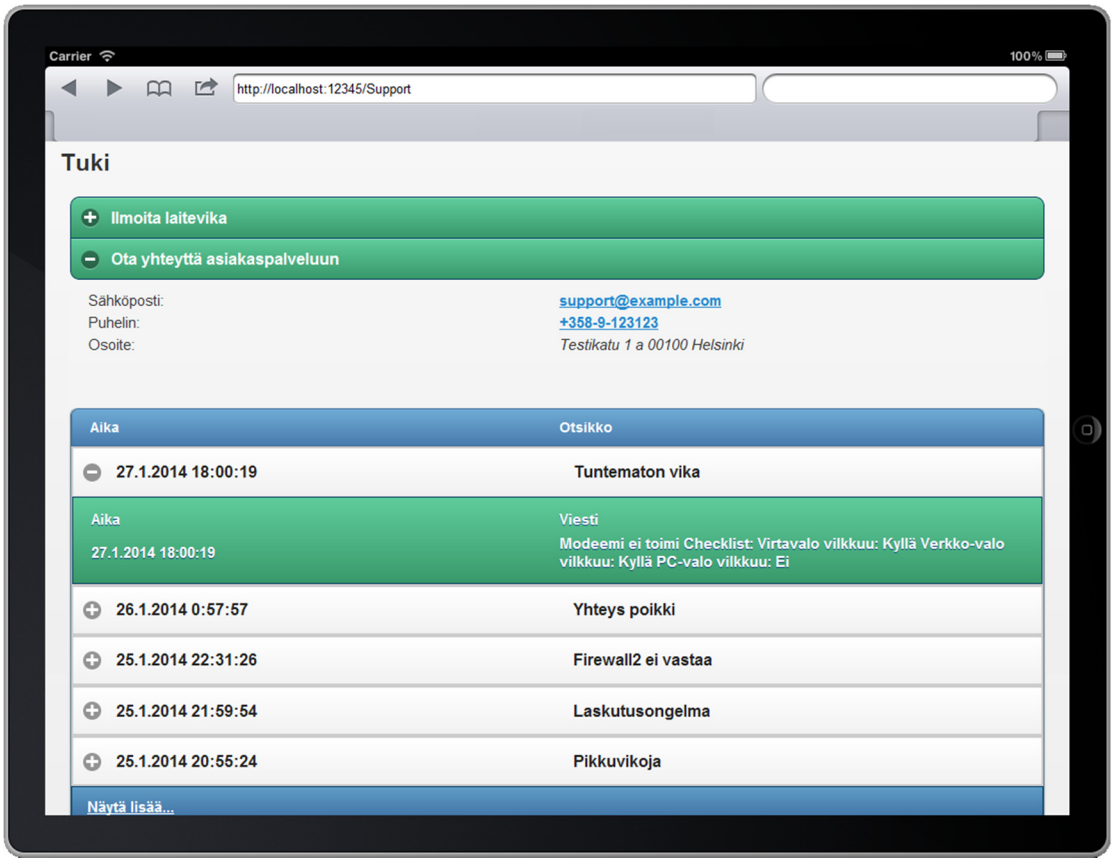
4.3.4 Verkko

Verkon laitteisto listaa asiakkaan verkon laitteet ja tarjoaa lisätietoja laitteista, kuten IP-osoitteen ja porttitiedot. Laitetiedot haetaan ulkoisesta NETadmin-verkonvalvontajärjestelmästä. Näkymä tarjoaa hakukentän, jolla voidaan etsiä laitteita laitteen nimellä tai sen lisätiedoilla, esimerkiksi portin nimellä. Hakuun sopivista laitteis-

ta näytetään kymmenen ensimmäistä ja lisää vaihtoehtoja saa tarkentamalla hakua. Sivun toiminta esitettiin jo aiemmin käyttöliittymän yhteydessä kuvassa 4.

4.3.5 Tuki

Tuki-sivu listaa asiakkaan tekemät tukipyynnöt ja antaa mahdollisuuden ottaa yhteyttä asiakaspalveluun. Kännykällä portaalia käytettäessä voidaan näpäyttää puhelinnumeroa, jolloin selain tarjoaa mahdollisuuden soittaa suoraan asiakaspalveluun. Lisäksi on tarjolla lomake laitevikoja varten ja asiakaspalvelun sähköpostiosoite. Tukipyyntölomake esiteltiin jo aiemmin käyttöliittymän yhteydessä kuvassa 3. Tukipyynnöt välitetään ulkoiseen NETadmin verkonvalvontajärjestelmään. Kun asiakas on tehnyt tukipyynnön saa hän vahvistuksen tukipyynnön lähettämisestä ja pyyntö ilmestyy tukipyyntölistaan. Listan otsikkokenttiä näpäyttämällä saadaan esiin tukipyynnössä tallennetut tiedot, jotka nähdään kuvassa 9.



Kuva 9. Tukipyynnöt.

4.4 Asiakas

4.4.1 Arkkitehtuuri

Selaimella suoritettava jQuery Mobile -kirjasto vastaa mobiiliportaalin käyttöliittymästä. Se tarjoaa tapahtumapohjaisen ohjelmointiympäristön portaalin sivurakenteelle ja vastaa käyttöliittymäkomponenttien alustayhteensopivuudesta. Asiakasarkkitehtuuri rakentuu jQuery Mobilen kehysrakenteen ympärille, joka perustuu JavaScript-tapahtumiin ja progressiiviseen kehitykseen.

Portaalin sivuilla on yhteiset JavaScript-tiedostot, jotka on jaettu seuraaviin nimiavaruuksiin:

- Portal – Portaaliluokka
- Portal.EventHandlers – Portaalin tapahtumankäsittelijät
- Portal.EventHandlers.Menu – Valikkotapahtumat
- Portal.EventHandlers.Controls – Käyttöliittymäkomponenttien hallinta
- Portal.EventHandlers.Messages – Viestitapahtumat
- Portal.EventHandlers.Ajax – Asynkroninen tiedon lataus
- Portal.EventHandlers.System – Järjestelmätason tapahtumat
- Portal.Configuration – Portaalin konfiguraatio
- Portal.Events – jQuery Mobilen tapahtumat

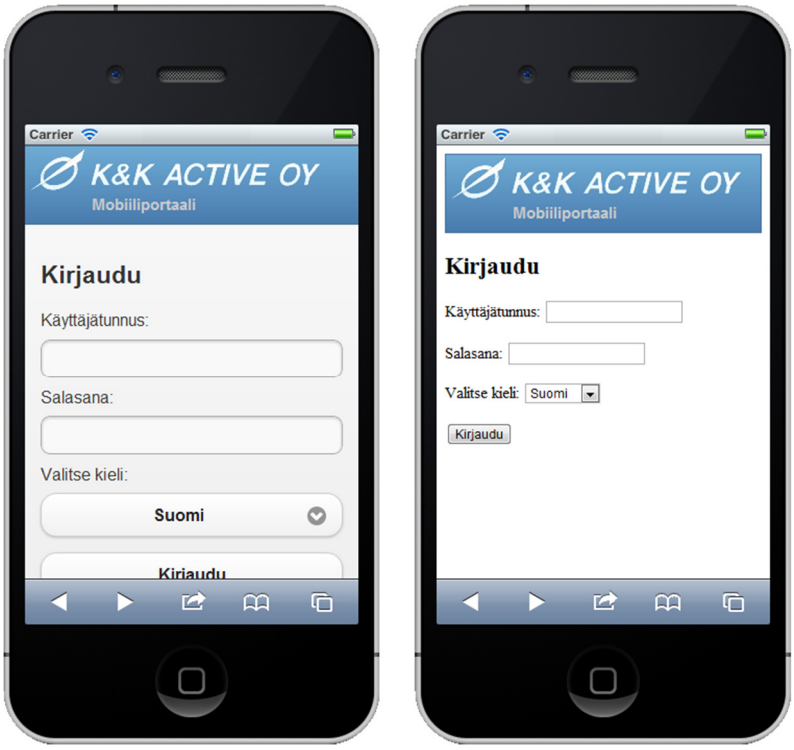
Lisäksi portaalilla on yhteinen CSS-tyylitiedosto. jQuery Mobile sisältää omat JavaScript- ja CSS-tiedostonsa, jotka ladataan portaaliin saavuttaessa. Tämän jälkeen skriptit pysyvät muistissa ja ne luovat selaimessa suoritettavan asiakasohjelman.

4.4.2 Progressiivinen kehitys

Laiteyhteensopivuuden säilyttämiseksi on mobiiliportaali suunniteltu progressiivisesti kerrostaen. Mobiiliportaalin sivut rakennetaan palvelimella kaikkien selainten tukemia HTML-elementtejä ja CSS-määrittelyitä käyttäen. Perustoiminnallisuus toteutetaan lo-

makkeilla ja linkeillä ja käytettävyyttä parannetaan käsittelemällä sivut JavaScript-koodin avulla.

Mobiiliportaalia on mahdollista käyttää myös ilman JavaScript-tukea. Tällöin kaikki jQuery Mobilen automatiikka kuitenkin menetetään ja toiminnallisuus perustuu palvelimella luotuihin HTML- ja CSS-ominaisuuksiin. Kuvassa 10 nähdään ero käyttöliittymän ulkoasussa JavaScript-tuen puuttuessa.



Kuva 10. Mobiiliportaalin ulkoasu JavaScript-tuen kanssa ja sen puuttuessa.

Erot sivujen ulkoasussa johtuvat jQuery Mobilen tekemästä työstä. Palvelimen lähettämä kirjautumissivun HTML-koodi ennen jQuery Mobilen prosessointia nähdään esimerkkikoodissa 5.

```
<p><label for="Username">Käyttäjätunnus:</label>
    <input class="text-box single-line" data-val="true" data-val-
required="Käyttäjätunnus pakollinen" id="Username" name="Username" type="text"
value="" />
</p><p>
    <label for="Username">Salasana:</label>
    <input class="text-box single-line password" data-val="true" data-val-
required="Salasana pakollinen" id="Password" name="Password" type="password"
value="" />
</p><p>
```

```
    <label for="Language">Valitse kieli:</label>
    <select data-submit="true" id="Language" name="Language"><option select-
ed="selected" value="fi">Suomi</option>
        <option value="en">Englanti</option>
        <option value="sv">Ruotsi</option>
    </select>
</p>
```

Esimerkkikoodi 5. Kirjautumissivun HTML-koodi ennen jQuery Mobilen prosessointia.

Esimerkkikoodissa 6 nähdään sama HTML-koodi prosessoinnin jälkeen. Koodista nähdään, että jQuery Mobile on lisännyt lukuisia lisäkerroksia ja ulkoasua muokkaavia luokkia alkuperäisen HTML-sivun elementteihin.

```
<p><label for="Username" class="ui-input-text">Käyttäjätunnus:</label>
    <div class="ui-input-text ui-shadow-inset ui-corner-all ui-btn-shadow ui-
body-c">
        <input class="text-box single-line ui-input-text ui-body-c" data-
val="true" data-val-required="Käyttäjätunnus pakollinen" id="Username"
name="Username" type="text" value="">
    </div>
</p><p>
    <label for="Username" class="ui-input-text">Salasana:</label>
    <div class="ui-input-text ui-shadow-inset ui-corner-all ui-btn-shadow ui-
body-c">
        <input class="text-box single-line password ui-input-text ui-body-c"
data-val="true" data-val-required="Salasana pakollinen" id="Password"
name="Password" type="password" value="">
    </div>
</p><p>
    <label for="Language" class="ui-select">Valitse kieli:</label>
    <div class="ui-select">
        <div data-corners="true" data-shadow="true" data-iconshadow="true" data-
wrapperels="span" data-icon="arrow-d" data-iconpos="right" data-theme="c"
class="ui-btn ui-shadow ui-btn-corner-all ui-btn-icon-right ui-btn-up-c">
            <span class="ui-btn-inner">
                <span class="ui-btn-text">
                    <span>Suomi</span>
                </span>
                <span class="ui-icon ui-icon-arrow-d ui-icon-
shadow"> </span>
            </span>
            <select data-submit="true" id="Language" name="Language"><option
selected="selected" value="fi">Suomi</option>
                <option value="en">Englanti</option>
                <option value="sv">Ruotsi</option>
            </select>
        </div>
    </div>
</p>
```

Esimerkkikoodi 6. Kirjautumissivun HTML-koodi jQuery Mobilen prosessoinnin jälkeen.

Koska portaalin koodi on suunniteltu toimimaan sellaisenaan, ei JavaScript-tuen puuttuminen estä koko portaalin käyttöä, vaan tarjoaa mobiililaitteelle ne toiminnot, joita se tukee. JavaScript-tuen löytyessä voidaan ottaa käyttöön lisätoimintoja, jotka mahdollistavat paremman käyttökokemuksen ja hienomman ulkoasun.

4.4.3 Tapahtumat

Mobiiliportaalin asiakaspään toiminta perustuu jQuery Mobilen tapahtumiin. Kun jQuery Mobile lataa uuden sivun, se aiheuttaa pagecreate-tapahtuman. Kun sivu näytetään selaimessa, aiheutuu pageshow-tapahtuma. Tapahtumiin voidaan kiinnittää JavaScript-tapahtumankäsittelijöitä, jotka aktivoituvat käyttäjän siirtyessä sivulta toiselle. Esimerkkikoodissa 7 nähdään tapahtumankäsittelijä, joka lisää data-confirm-attribuutilla varustettuihin elementteihin ylimääräisen varmistusviestin, joka näytetään niitä näpäytettäessä.

```
Portal.EventHandlers.Messages.ConfirmMessage = function (e) {
    var currentPage = $(e.target);

    var click = function (e) {
        var message = $(this).attr("data-confirm");
        return confirm(message);
    };
    currentPage.find("[data-confirm]").on("click", click);
}
```

Esimerkkikoodi 7. Pagecreate-tapahtumankäsittelijä.

Tapahtumankäsittelijän aktivoiva koodi nähdään esimerkkikoodissa 8.

```
<div class="single-button" data-confirm="Oletko varma että haluat poistaa käyt-
täjän admin?">
    <a data-icon="delete" data-iconpos="notext" data-role="button"
href="/Account/DeleteUser/admin">Poista käyttäjä</a>
    <a href="/Account/DeleteUser/admin">Poista käyttäjä</a>
</div>
```

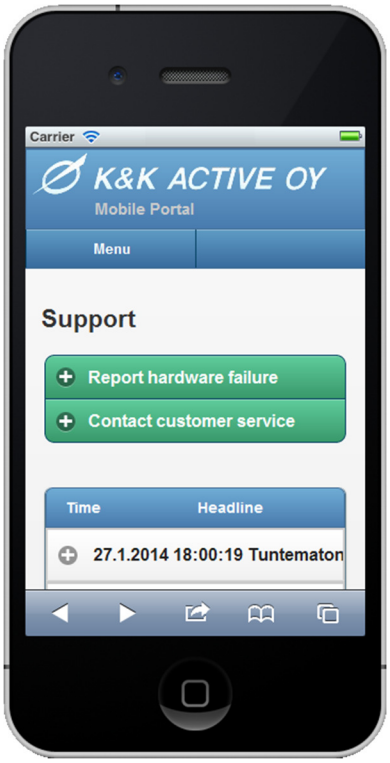
Esimerkkikoodi 8. Tapahtuman käsittelijän aktivoiva HTML-koodi.

Myös muut selaimella tehtävät toiminnot aiheuttavat tapahtumia, kuten ruudun näpäytys. Näitä toimintoja hyödynnetään mobiiliportaalin käytettävyyden parantamisessa. Normaalisti mobiililaitteet lisäävät noin 300 millisekunnin viiveen ennen kuin ne suorittavat näpäytyksestä aiheutuvat click-tapahtuman. Tämä aiheuttaa käyttöliittymän tun-

tumisen hitaalta. Mobiiliportaalissa osa näpäytyksistä, esimerkiksi valikon avaus, hyödyntää ontouchstart-tapahtumaa, jolloin tapahtumankäsittelijä suoritetaan välittömästi.

4.4.4 Komponentit

Mobiiliportaalin käyttöliittymä hyödyntää jQuery Mobilen visuaalisia komponentteja. Peruskomponentit, kuten napit ja tekstikentät, muunnetaan automaattisesti pyöreäreunaisiksi, ja ne mukautuvat sivun levyisiksi mahdollistaen helpon kosketuksen. Peruskomponenttien lisäksi portaalissa käytetään erilaisia listoja ja valikkoja, jotka aktivoidaan käyttämällä tiettyjä data-role-attribuutteja HTML-elementtien joukossa. Kuvassa 11 on esimerkki Collapsible Set -komponentista, jonka otsikoita näpäyttämällä saadaan esiin lisää tietoa. Kuvan vihreää nappia näpäyttämällä aukeaa tukipyyntölomake tai asiakaspalvelun yhteystiedot.



Kuva 11. Collapsible Set -komponentti.

Collapsible Set -elementti koostuu tavallisesta div-elementistä, johon on lisätty attribuutti data-role=”collapsible-set”. jQuery Mobile prosessoi div-elementin sisällön ja luo siitä graafisen valikon. Esimerkkikoodissa 9 nähdään Collapsible Set -komponentin

luova koodi. Komponentin otsikot otetaan H3-elementeistä ja sisältö niiden rinnalla olevista div-elementeistä.

```
<div data-role="collapsible-set" data-theme="b" class="green list">
    <div data-role="collapsible">
        <h3>@Language.Translate("Report hardware failure")</h3>
        <div>
            …
        </div>
    </div>
    <div data-role="collapsible">
        <h3>@Language.Translate("Contact customer service")</h3>
        <div>
            …
        </div>
    </div>
</div>
```

Esimerkkikoodi 9. Collapsible Set -komponentin luova koodi.

Toinen portaalissa yleisesti käytetty jQuery Mobilen komponentti on listview-elementti, joka luo graafisen listan normaaleista ul- ja li-elementeistä. Listview-komponentissä voidaan myös määritellä data-filter-attribuutti, jolla jQuery Mobile lisää sivulle automaattisen hakupalkin. Portaalin verkkolaitteet listaava Listview-komponentti nähdään kuvassa 12.

Kuva 12. Listview-komponentti.

Listan muodostava ohjelmakoodi nähdään esimerkkikoodissa 10. Listan otsikot määritellään attribuutilla data-role=”list-divider” ja sitä voidaan käyttää myös rivien välierottimena. Hakupalkki toimii automaattisesti ja suodattaa listan sisällön annetun hakusanan mukaisesti.

```
<ul data-role="listview" data-filter="true" data-filter-
placeholder="@Language.Translate("Search")">
    <li data-role="list-divider">
        <table><tr>
            <td>@Language.Translate("Equipment")</td>
            <td>@Language.Translate("IP address")</td>
        </tr></table>
    </li>
    @foreach (Equipment equipment in Model.equipment)
    {
        <li class="collapsible">
            <div data-role="collapsible"
                …
            </div>
        </li>
    }
</ul>
```

Esimerkkikoodi 10. Listview-komponentin luova koodi.

4.4.5 Dynaaminen sisältö

Käytettävyyden parantamiseksi on portaaliin lisätty data-attribuutteja, jotka jQuery Mobilen komponenttien tavoin prosessoidaan, kun käyttäjä siirtyy sivulle. Attribuutti data-portal-content on tarkoitettu sisällön lataamiseen AJAX-kyselynä. Attribuutti saa parametrikseen ladattavan sisällön osoitteen. Lisäksi määritetään attribuutit data-target ja data-insert, jotka kertovat, mihin ja miten ladattu tieto sijoitetaan. Esimerkkikoodissa 11 nähdään dynaamisen sisällön lataukseen käytetyt data-attribuutit.

```
<ul data-role="listview" id="alertsList"
    data-inset="true"
    data-portal-content="@Url.Action("Alerts")"
    data-target="dynamicAlerts"
    data-insert="@InsertType.BEFORE"
    data-show-min="5"
    data-show-count="10"
    data-show-offset="@Model.showOffset">

    <li class="showMore" data-theme="b" id="dynamicAlerts">
    </li>
</ul>
```

Esimerkkikoodi 11. Dynaamisen sisällön määrittely.

Koodissa esiintyviä data-show-attribuutteja käytetään rajoittamaan kerralla näkyvien elementtien määrää. Sisältöä ladataan listaan aina hieman enemmän kuin siinä näytetään. Käyttäjän näpäyttäessä lisää-nappia, voidaan seuraavat listan rivit näyttää välittömästi ja alkaa ladata uutta sisältöä taustalla. Esimerkkikoodissa 12 nähdään dynaamisen sisällön lataava JavaScript-koodi, joka käyttää AJAX-hakuun data-attribuutteja.

```
var count = parseInt(container.attr("data-show-count"));
var offset = container.attr("data-show-offset");

currentPage.find("[data-portal-content]").each(function () {
    var contentUrl = container.attr("data-portal-content");
    if (contentUrl !== undefined) {
        $.ajax({
            type: "POST",
            url: contentUrl,
            data: "count=" + count + "&offset=" + offset,
            success: function (data, status, request) {
               var insertMethod = container.attr("data-insert");
               var target = $("#" + container.attr("data-target"));

               switch (insertMethod) {
                    case "after":
                        target.after(data);
                        break;
                    case "before":
                        target.before(data);
                        break;
                    case "replace":
                        target.replaceWith(data);
                        break;
                    case "into":
                    default:
                        target.html(data);
                }
            }
        });
    }
});
```

Esimerkkikoodi 12. Dynaamisen sisällön lataus.

4.5 Palvelin

4.5.1 Arkkitehtuuri

Mobiiliportaalin palvelinarkkitehtuuri rakentuu ASP.NET MVC:n kehysrakenteen ympärille. Jokainen kutsu reititetään omalle kontrollerilleen, joka valitsee tilanteeseen sopivan mallin ja rakentaa näkymän mallin perusteella. Palvelimen tehtävä on sekä vastata käyttöliittymän kutsuihin että rakentaa itse käyttöliittymä. Kaksijakoinen tehtävä tuottaa

haasteita ohjelmakoodin rakenteeseen. ASP.NET MVC pakottaa kuitenkin käyttämään MVC-suunnittelumallia, josta seuraa rakenteellista ja helposti ylläpidettävää ohjelmakoodia. Luontaisen tilattomuutensa vuoksi se tarjoaa erinomaisen rajapinnan jQuery Mobilen ohjelma-arkkitehtuurille.

Mobiiliportaalissa on hyödynnetty ASP.NET MVC:n mahdollistamaa komponenttien laajentamista. Jokaista komponenttia on mahdollista laajentaa tai ne voidaan korvata kokonaan omilla versioilla. Osa vakiokomponenteista integroituu suoraan jQuery Mobilen JavaScript-kirjastoihin ja tuottaa sen ymmärtämää aktiivista HTML-muotoilukieltä. Mobiiliportaalissa käytetään myös kolmannen osapuolen DLL-kirjastoja ulkoisten tietokantayhteyksien käsittelyyn.

Palvelinpuolen luokkatiedostot on ryhmitelty seuraaviin nimiavaruuksiin:

- Ajax – Dynaamisen sisällön hallinta
- Database – Tietokannan käsittely
- Localization – Kielenkäännökset ja metatiedon hallinta
- Netadmin – Ulkoisen verkonvalvontajärjestelmän rajapintaluokat
- Security – Käyttäjien todentaminen ja synkronointi
- Controllers – Kontrollerit
- Models – Mallit
- Views – Näkymät

Luokat on ohjelmoitu C#-kieltä käyttäen ja näkymien esittämisessä käytetään Razor-moottoria.

4.5.2 AJAX

jQuery Mobile pyrkii lataamaan kaikki sivut AJAX-pyyntöjä käyttäen. ASP.NET MVC mahdollistaa AJAX-pyyntöjen erottamisen, ja vastaus voidaan kohdistaa tarkasti haluttuun tarkoitukseen. Kun portaaliin saavutaan ensimmäisen kerran, pitää jQuery Mobilea varten ladata erilaisia JavaScript-kirjastoja ja CSS-tiedostoja. Mobiiliportaali käyttää näkymien pohjana yhteistä Layout-tiedostoa, joka sisältää viittaukset tarvittaviin tiedos-

toihin. Layout-tiedoston sisältämät yhteiset otsikkotiedot nähdään esimerkkikoodissa 13.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>@MobilePortal.Language.Translate("Mobile Portal")</title>
        <meta name="viewport" content="initial-scale=1,user-scalable=no,maximum-
scale=1" />
        <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
        @Styles.Render("~/bundles/css")
        @Scripts.Render("~/bundles/jquerymobile")
        @Scripts.Render("~/bundles/portal")
    </head>
    <body>
        @RenderBody()
    </body>
</html>
```

Esimerkkikoodi 13. Sivuille yhteinen Layout-tiedosto.

Koska sivut säilytetään selaimen DOM-puussa, ei skripti- ja tyylitiedostoja tarvitse ladata kuin kerran. Kun mobiiliportaalissa liikutaan sivulta toiselle, lähettää jQuery Mobile palvelimelle AJAX-pyynnön. Jokainen selaimen lähettämä pyyntö käsitellään ViewStart-tiedostossa, joka valitsee käytettävän Layout-tiedoston. AJAX-pyyntöjen tapauksessa Layout-tiedostoa ei tarvita ollenkaan, ja näin vastauksessa lähetetään pelkkä jQuery Mobile -sivu. Tämä pienentää lähetettävän tiedon määrää ja nopeuttaa sivujen lataamista. ViewStart-tiedoston toiminta nähdään esimerkkikoodissa 14.

```
@{
    if (Request.IsAjaxRequest())
    {
        Layout = null;
    }
    else
    {
        Layout = "~/Views/Shared/_Layout.cshtml";
    }
}
```

Esimerkkikoodi 14. Oletuspohjan valitseva ViewStart-tiedosto.

AJAX-pyyntöjä käytetään myös parantamaan portaalin palveluiden käytettävyyttä. Tällöin palvelimelta pyydetään ainoastaan pieni pala sivua. Portaalin tilanne-sivulla näytettävä listaus laitehälytyksistä haetaan palvelimelta tällä tekniikalla. Pyyntö kohdistetaan osoitteeseen /Status/Alerts, joka palauttaa vastauksena pätkän HTML-koodia. Palau-

tettava koodi liitetään suoraan selaimen DOM-puuhun, joten mitään otsikkotietoja ei tarvita. AJAX-vastauksen palauttava ohjelmakoodi nähdään esimerkkikoodissa 15.

```
@model MobilePortal.Models.StatusModel
@foreach (Alert alert in Model.alerts)
{
    <li style="padding:0 0; border: 1px solid #333" data-type="data">
        <div class="cell alert-color" style="background: @alert.Color;">
             
        </div>
        <div class="cell alert-data">
            <div class="small-text">@alert.StartTime.ToString("dd.MM.yyyy
HH:mm:ss")</div>
            <div>@alert.Equipment</div>
            <div class="small-text">@alert.Message</div>
        </div>
    </li>
}
```

Esimerkkikoodi 15. AJAX-vastauksen palauttama Alerts-näkymä.

4.5.3 Tiedon tallennus

Mobiiliportaali sisältää oman SQL Server -tietokannan, mutta sen tarkoitus ei ole säilyttää tietoa kuin tilapäisesti. Portaalin käsittelemät tiedot on suunniteltu synkronoitavaksi ulkoisesta järjestelmästä ja paikallista tietokantaa käytetään ainoastaan parantamaan käytettävyyttä. Käyttäjän kirjautuessa portaaliin synkronoidaan käyttäjätiedot paikalliseen tietokantaan. Käyttäjätiedot ovat tämän jälkeen nopeasti saatavissa, eikä niitä tarvitse hakea kokoajan uudestaan.

Käyttäjätietojen tallentamiseen portaali hyödyntää ASP.NET:n ProfileBase-luokkaa. Luokka on tarkoitettu käyttäjän profiilitietojen tallentamiseen ja soveltuu hyvin käyttäjätietojen tilapäissäilytykseen. Käytettävissä olevat profiilitiedot konfiguroidaan Web.config-tiedostoon. Esimerkkikonfiguraatio nähdään esimerkkikoodissa 16.

```
<profile defaultProvider="DefaultProfileProvider">
    <providers>
        <add name="DefaultProfileProvider"
type="System.Web.Providers.DefaultProfileProvider, System.Web.Providers, Ver-
sion=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" connection-
StringName="DefaultConnection" applicationName="/" />
    </providers>
    <properties>
        <add name="CustomerId" type="System.Int64" />
        <add name="CompanyId" type="System.Int64" />
    </properties>
</profile>
```

Esimerkkikoodi 16. Web.config tiedoston Profile-määritykset.

Kun profiilitiedot on konfiguroitu, ASP.NET huolehtii tietokannan rakentamisesta ja SQL-kyselyiden muodostamisesta. Profiilitiedot talletetaan SQL Server -tietokannan Profile-tauluun, josta ne voidaan hakea ProfileBase-luokan avulla. Esimerkkikoodissa 17 nähdään esimerkki käyttäjätunnukselle tallennettujen profiilitietojen hakemisesta.

```
var data = new UserData();
ProfileBase userProfile = ProfileBase.Create(user.UserName);

data.CustomerId = (long)userProfile.GetPropertyValue(UserProperties.CustomerId);
data.CompanyId = (long)userProfile.GetPropertyValue(UserProperties.CompanyId);
```

Esimerkkikoodi 17. Käyttäjätietojen haku ProfileBase-luokan avulla.

Mobiiliportaalin vastaanottamia tietoja voidaan myös välittää ulkoisiin järjestelmiin. Portaalin vastaanottamat tukipyynnöt siirretään ulkoiseen NETadmin verkonvalvontajärjestelmään. Tähän hyödynnetään järjestelmän valmistajan tarjoamaa DLL-rajapintaa. Kun tietoa säilytetään ulkoisessa järjestelmässä, siirtyy myös vastuu tiedon hallinnasta, ja portaali voi keskittyä sen päätarkoitukseen, eli palvelujen tarjoamiseen.

4.5.4 Validointi

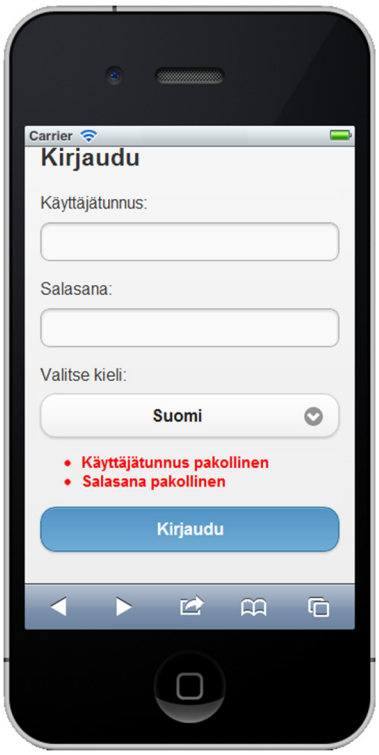
Mobiiliportaalin lomakkeiden validointi pyritään tekemään jo asiakaspäässä. ASP.NET MVC tarjoaa palvelimelle ja asiakkaalle yhteisen validoinnin jQuery Unobtrusive Validation -skriptiä käyttämällä. Validoitavat tiedot määritetään kontrollerin vastaanottamissa malleissa. ASP.NET MVC sisältää eri tarkoituksiin soveltuvia validointiattribuutteja ja niitä voidaan tarvittaessa laajentaa. Esimerkkikoodissa 18 nähdään mobiiliportaalin kirjautumissivulla validoitavat tiedot. Käyttäjätunnus ja salasana on määritetty pakolli-

siksi ja salasana on määritetty Password-tyyppiseksi, jolloin sitä ei näytetä kirjoitettaessa.

```
public class LoginModel
{
    [Required]
    public string Username { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    public string Language { get; set; }
    public string PreviousLanguage { get; set; }
}
```

Esimerkkikoodi 18. Kirjautumissa validoitavien tietojen asetus.

Validoinnin virheilmoitukset syntyvät RequiredAttribute-luokassa ja ne tulostetaan ruudulle, jos attribuutin määrittämä pakollisuus ei täyty. Kuvassa 13 nähdään käyttäjätunnuksen ja salasanan puuttumisesta aiheutuneet virheilmoitukset.



Kuva 13. Validoinnin virheilmoitukset.

Jotta validointi toimisi myös asiakaspäässä, kirjoittaa ASP.NET MVC virheilmoitukset ja validointisäännöt suoraan HTML-elementtien data-attribuutteihin. jQuery Unobtrusive

Validation -skripti lukee data-attribuutit ja estää lomakkeen lähetyksen, jos niiden ehdot eivät täyty. Esimerkkikoodissa 19 nähdään mobiiliportaalin kirjautumissivun validointiin käytettävät data-attribuutit.

```
<label for=”Username” class=”ui-input-text">Käyttäjätunnus:</label>
<div class="ui-input-text ui-shadow-inset ui-corner-all ui-btn-shadow ui-body-
c">
    <input class="text-box single-line ui-input-text ui-body-c input-validation-
error" data-val="true" data-val-required="Käyttäjätunnus pakollinen"
id="Username" name="Username" type="text" value="">
</div>

<label for="Username" class="ui-input-text">Salasana:</label>
<div class="ui-input-text ui-shadow-inset ui-corner-all ui-btn-shadow ui-body-
c">
    <input class="text-box single-line password ui-input-text ui-body-c input-
validation-error" data-val="true" data-val-required="Salasana pakollinen"
id="Password" name="Password" type="password" value="">
</div>
```

Esimerkkikoodi 19. Kirjautumisen validoiva HTML-koodi.

4.5.5 Tietoturva

Portaalin käyttöoikeudet tarkastetaan jokaisella pyynnöllä. Tämä tapahtuu ASP.NET:n Forms Authentication -todentamista käyttämällä. Jos selain ei pysty esittämään pyyntöön oikeuttavaa aitoustodistusta, ohjataan käyttäjä takaisin kirjautumissivulle.

Portaalin käyttöoikeudet voidaan myös tarkistaa ulkoisesta järjestelmästä. Tämän mahdollistaa laajennettu PortalMembershipProvider-luokka, joka huolehtii käyttöoikeuksien tarkistamisesta. Koska luokka vain laajentaa ASP.NET:n omaa MembershipProvider-luokkaa, säilyy tietoturva yhtä hyvänä, kuin Microsoft on sen suunnitellut. Voidaan myös määritellä virheellisten kirjautumisyritysten määrä tietyssä ajanjaksossa, jonka ylittäminen aiheuttaa käyttäjän lukitsemisen ulos järjestelmästä. Tämän tarkoitus on estää koneellinen salasanojen selvittäminen.

Kun käyttäjä on todennettu, voidaan ulkoisesta järjestelmästä synkronoida lisätietoja määrittelemään käyttäjälle sallitut palvelut. Eritasoiset palvelut suojataan asettamalla käyttäjille rooleja, jotka varmistetaan jokaisen sivulatauksen yhteydessä. Myös yksittäiset toiminnot voidaan rajata vain tietyille käyttäjäryhmille. Esimerkki järjestelmänvalvojille rajatusta kontrolleriluokasta nähdään esimerkkikoodissa 20. Yritys käyttää rajoitettua toimintoa palauttaa käyttäjän aloitussivulle.

```
[Authorize(Roles = "admin")]
public class AccountController : Controller
{
    // GET: /Account
    [HttpGet]
    public ActionResult Index()
    {
    }
}
```

Esimerkkikoodi 20. Admin-roolille rajoitettu kontrolleriluokka.

4.5.6 Lokalisointi

Tuki usealle kielelle on toteutettu laajentamalla ASP.NET MVC:n DataAnnotationsModelMetadataProvider-luokkaa. Luokan tehtävä on tarjota malleissa määriteltyjä virheilmoituksia ja kuvauksia näkymälle. Laajennettu luokka kääntää nämä ilmoitukset automaattisesti kulloinkin käytössä olevalle kielelle. Luokan käyttäminen mahdollistaa muun muassa oikeankieliset virheilmoitukset asiakaspäässä käsiteltäville lomakkeille.

Kielenkäännösten helppoa lisäämistä ja käyttöä varten toteutettiin portaaliin siihen erikoistunut Language-luokka. Luokan kääntäjä etsii sanavarastosta vastinetta käännettävälle sanalle ja antaa oikeankielisen vastineen sen löytyessä. Jos vastinetta ei löydy, palauttaa kääntäjä käännettävän sanan sellaisenaan. Tämä mahdollistaa käännettävien sanojen lisäämisen myös myöhemmin, eikä kehittäjä joudu jatkuvasti päivittämään sanastoa. Esimerkki uusien käännösten lisäämisestä nähdään esimerkkikoodissa 21.

```
words.Add("Login", new Words("Login", "Kirjaudu", "Logga in"));
words.Add("Username", new Words("Username", "Käyttäjätunnus", "Användarnamn"));
words.Add("Password", new Words("Password", "Salasana", "Lösenord"));
```

Esimerkkikoodi 21. Uusien kielenkäännöksien lisääminen.

Esimerkkikoodissa 22 nähdään käännösten käyttö mobiiliportaalin kirjautumissivulla.

```
@Html.LabelFor(m => m.Username, Language.Translate("Username") + ":")

@Html.LabelFor(m => m.Username, Language.Translate("Password") + ":")

<input type="submit" value="@Language.Translate("Login")" />
```

Esimerkkikoodi 22. Kielenkäännösten käyttö kirjautumissivulla.

Language-luokka on suunniteltu niin, että sanat ladataan jaettuun muistiin ohjelmaa käynnistettäessä ja ne ovat nopeasti saatavissa kaikille käyttäjille. Luokan rakenne mahdollistaa myös sanojen hakemisen tietokannasta.

4.6 Testaus

4.6.1 Menetelmät

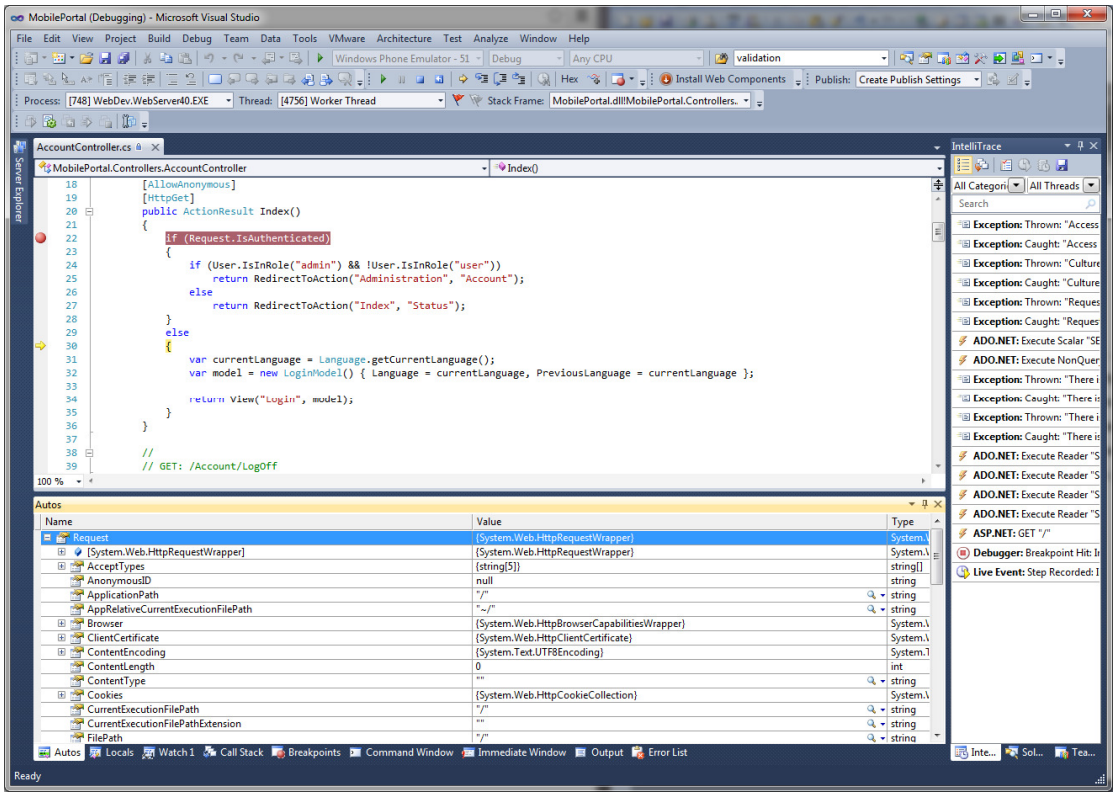
Mobiiliportaalin toteutus vaati jatkuvaa testaamista. Vaikka kaikki mobiililaitteet tukevat web-toteutusta, eroaa niiden HTML-, CSS- ja JavaScript-koodin käsittely suuresti toisistaan. Tämä vaatii jokaisen portaalin ominaisuuden testaamisen jokaisella mobiilialustalla.

Mobiiliportaalia testattiin eri tavoin kehityksen eri vaiheissa. Palvelinominaisuuksia kehitettäessä testaus keskittyi lähinnä Visual Studion testiohjelmaan, kun taas käyttöliittymää testattaessa jouduttiin hyppimään testiympäristöstä toiseen, jotta varmistettiin portaalin yhteensopivuus eri laitteilla.

Käyttöliittymätestauksessa hyödynnettiin työpöytäselaimia, mutta niillä ei voi korvata aitoa laiteympäristöä. Peukalosääntönä voidaan sanoa, että jos sovellus toimii mobiililaitteessa, niin se toimii myös työpöytäselaimessa. Samaa ei voi sanoa toisinpäin, vaan sovellus on aina testattava myös todellisilla laitteilla tai emulaattoreilla.

4.6.2 Palvelin

Mobiiliportaalin palvelinpuolen testaaminen sujui varsin helposti. ASP.NET MVC:n ohjelmointiympäristöksi tarkoitettu Microsoft Visual Studio 2010 -ohjelmisto sisältää erinomaisen testiohjelman, jolla ohjelmakoodin toimintaa voidaan tutkia reaaliajassa. Ohjelmakoodin suoritusta voidaan seurata rivi riviltä ja ohjelman tilaa voidaan tarkastella millä hetkellä tahansa asettamalla ohjelmakoodiin pysähdysmerkkejä. Kun ohjelma saavuttaa pysähdysmerkin, Visual Studio pysäyttää suorituksen ja tarjoaa kaiken tarpeellisen tiedon ohjelman käyttämistä resursseista. Visual Studion testiohjelma nähdään kuvassa 14.

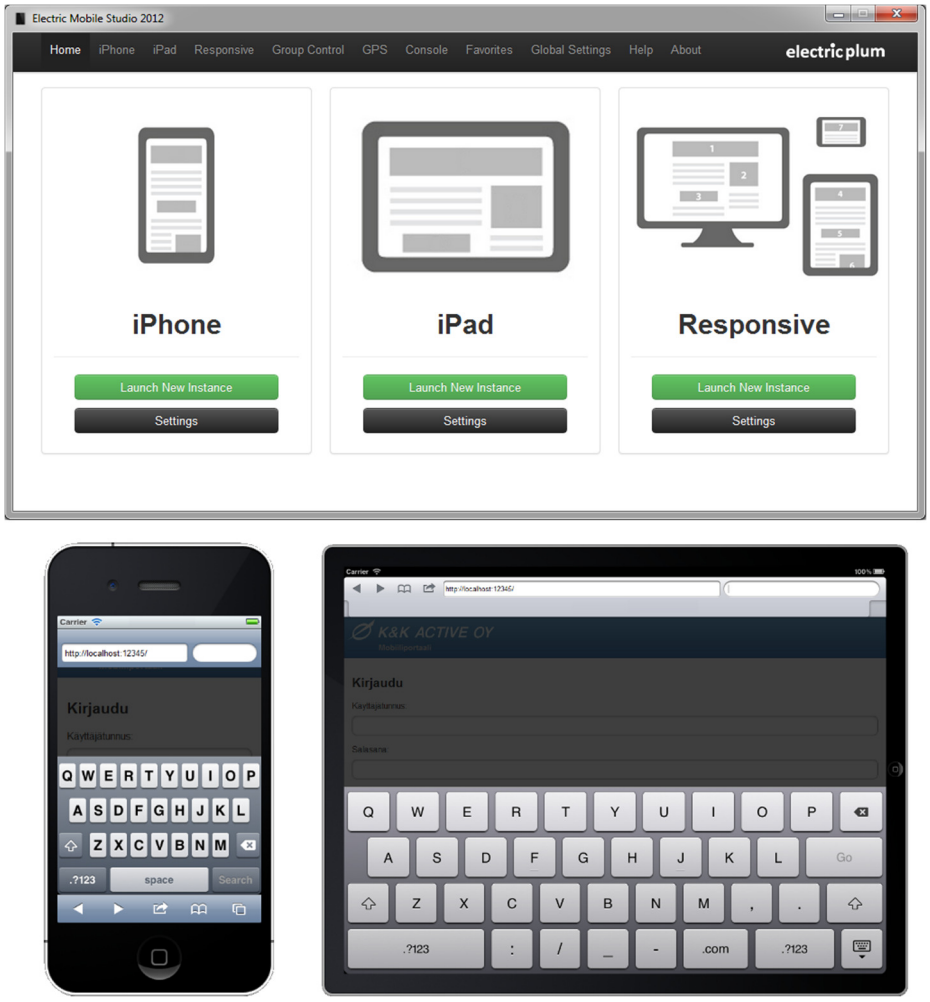


Kuva 14. Microsoft Visual Studio 2010:n testiohjelma.

Koska mobiiliportaali on rakennettu ASP.NET MVC:n arkkitehtuuria noudattaen, voidaan jokainen komponentti testata erikseen. Näkymät rakennetaan niille välitetyn mallin perusteella, joten testiohjelmasta on helppo nähdä, mitä arvoja malli sisältää ja miten kontrolleri niitä käsittelee. Testausta varten voidaan myös toteuttaa erityinen testiluokka, joka syöttää kontrollerille halutut parametrit ja tarkistaa, että tuloksena syntynyt malli on oikea.

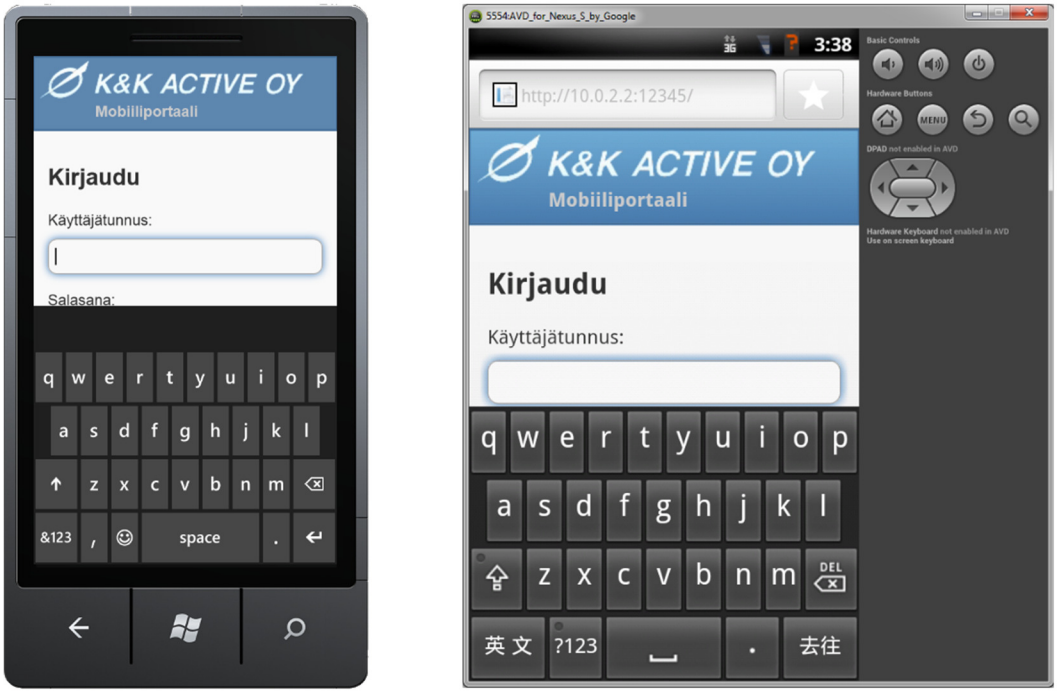
4.6.3 Emulaattorit

Käyttöliittymätestaukseen käytettiin Electric Mobile Studio 2012 -emulaattoria, jolla voidaan simuloida iPhone- ja iPad-laitteita. Ohjelma soveltuu erityisesti web-sovellusten ulkoasun ja JavaScript-toimintojen testaamiseen. Hienon ulkoasun lisäksi se sisältää työpöytäselaimista tutut testausominaisuudet ja mahdollistaa simuloitavan laitteen ominaisuuksien karsimisen. Electric Mobile Studio 2012 -emulaattori nähdään kuvassa 15.



Kuva 15. Electric Mobile Studio 2012.

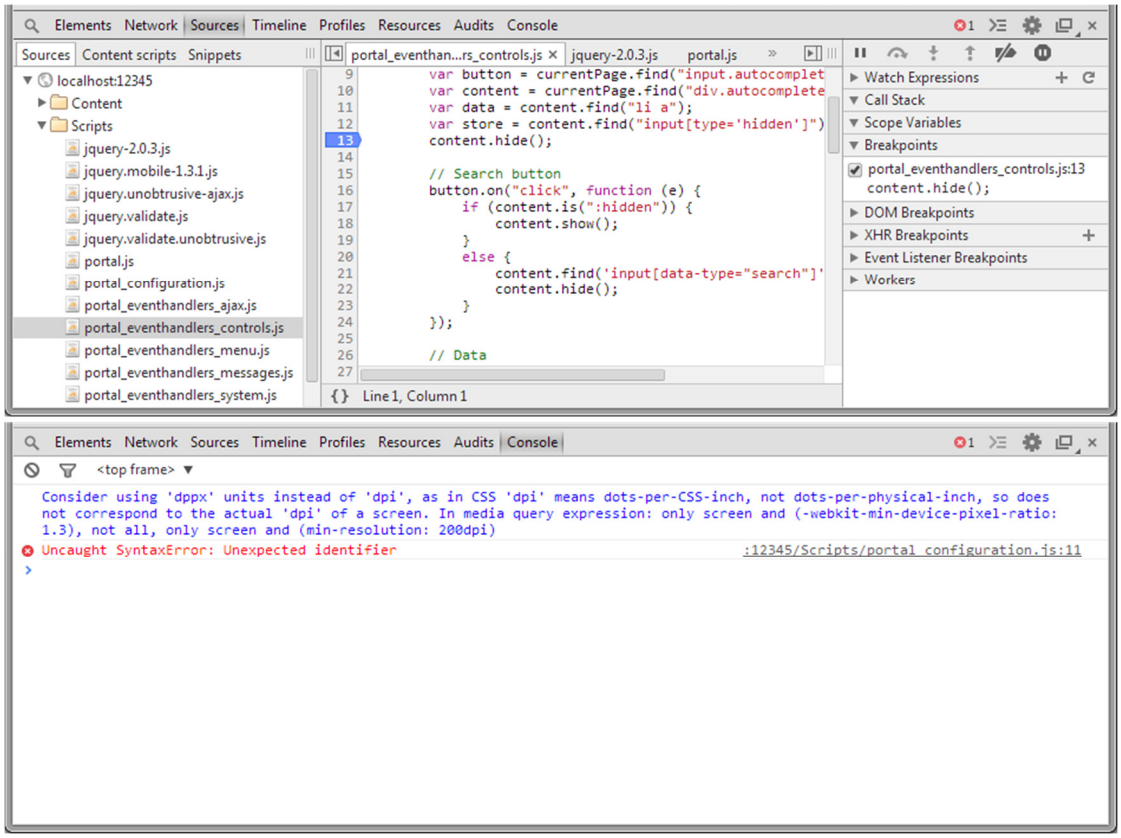
Käyttöliittymätestauksessa käytettiin myös laitevalmistajien tarjoamia Windows Phone- ja Android-emulaattoreita, jotka on tarkoitettu kyseisten laitteiden sovelluskehitykseen. Nämä emulaattorit toimivat kuten todelliset laitteet, ja ne sisältävät myös internet-selaimen, joten niillä voidaan myös testata web-sovelluksia. Ne eivät kuitenkaan tarjoa kunnon testiohjelmaa, joten niillä on vaikea selvittää JavaScript-virheitä ja muita yhteensopivuusongelmien syitä. Windows Phone- ja Android-emulaattorit nähdään kuvassa 16.



Kuva 16. Windows Phone- ja Android-emulaattorit.

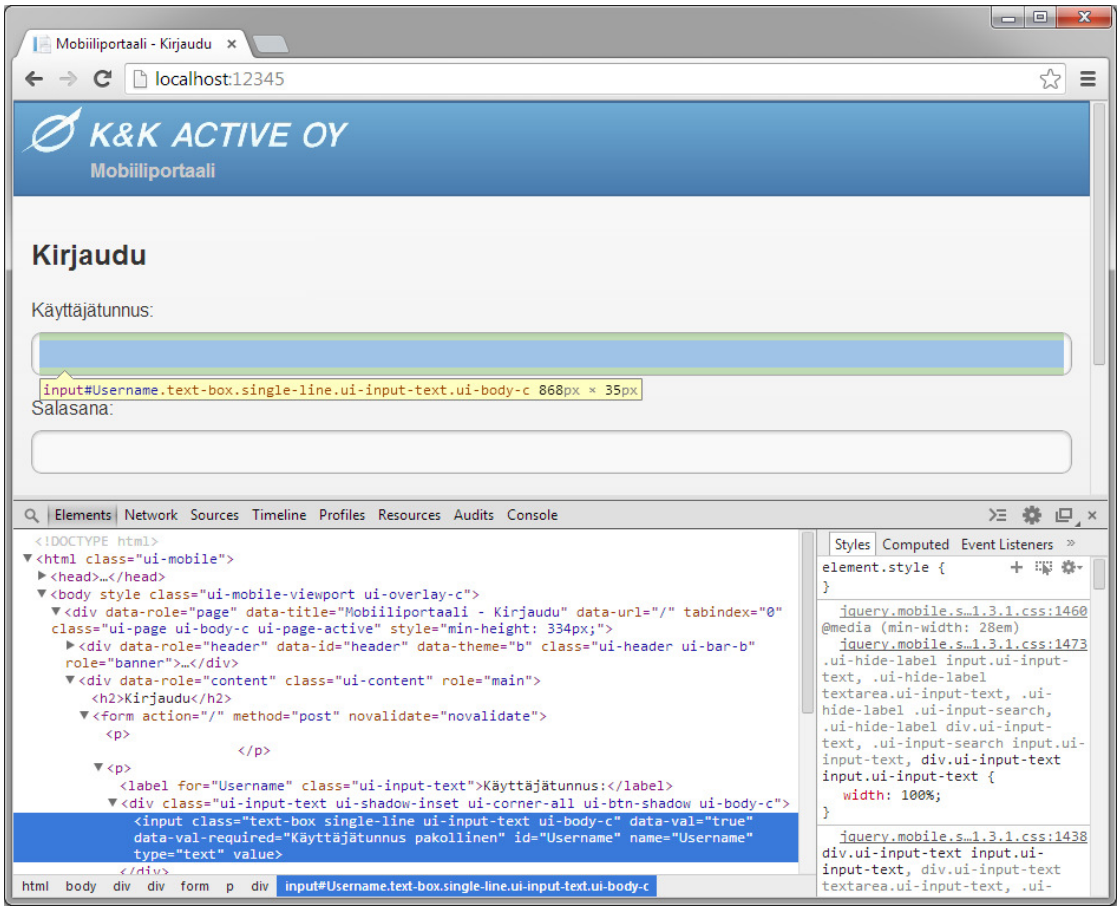
4.6.4 Työpöytäselaimet

Työpöytäselaimet sisältävät web-kehityksessä hyödyllisiä toimintoja, kuten JavaScript-testiohjelman, virhekonsolin, dynaamisen lähdekoodin ja selaimen ja palvelimen välisen liikenneseurannan. JavaScript-testiohjelmalla voidaan jäljittää asiakaspään toiminnallisuuksien virheitä ja ongelmakohtia. Testiohjelmalla voidaan suorittaa JavaScript-koodia rivi riviltä ja tutkia muuttujien arvoja kyseisellä hetkellä. Mahdolliset virheet tulostuvat virhekonsoliin. Google Chrome -selaimen JavaScript-testiohjelma ja virhekonsoli nähdään kuvassa 17.



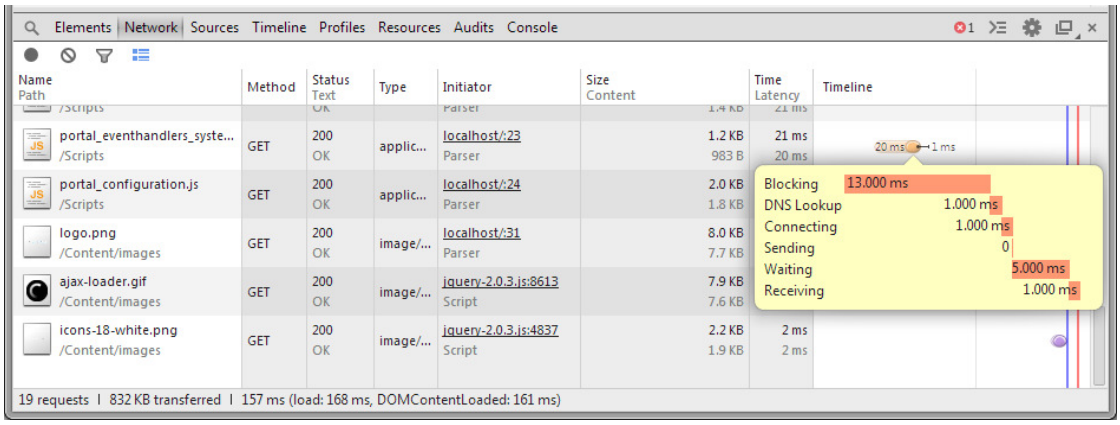
Kuva 17. Google Chrome -selaimen JavaScript-testiohjelma ja virhekonsoli.

Käyttöliittymää toteutettaessa joudutaan usein kokeilemaan erilaisia vaihtoehtoja. Jos jokin näkymä syntyy dynaamisesti, esimerkiksi jQuery Mobilen muokkaamana, on hyvin työlästä kirjoittaa testattavia arvoja suoraan lähdekoodiin ja ajaa ohjelmaa selaimella uudestaan ja uudestaan. Tähän avuksi tulee työpöytäselainten dynaaminen lähdekoodi. Dynaamisesta lähdekoodista voidaan tarkistaa sen hetkinen HTML-koodi ja sitä voidaan muokata lennossa. Tämä mahdollistaa eri vaihtoehtojen helpon kokeilun, ennen muutosten siirtoa itse lähdekoodiin. Google Chrome -selaimen dynaaminen lähdekoodi nähdään kuvassa 18.



Kuva 18. Google Chrome-selaimen dynaaminen lähdekoodi.

Yksi tärkeimmistä työpöytäselaimien tarjoamista työkaluista on selaimen ja palvelimen välisen liikenteen seuranta. Työkalu näyttää jokaisen asiakkaan tekemän kutsun palvelimelle ja niihin kuluneen ajan. Raportista voidaan löytää suorituskyvyn ongelmakohdat. Jokainen kutsu voidaan myös avata, jolloin nähdään pyynnön ja vastauksen tarkat otsikkotiedot. Tämä auttaa selvittämään, sijaitseeko ongelma selaimen vai palvelimen päässä. Google Chrome -selaimen liikenneseuranta nähdään kuvassa 19.



Kuva 19. Google Chrome -selaimen liikenneseuranta.

Portaalin testaamisessa käytettyjä työpöytäselaimia olivat: Google Chrome, Mozilla Firefox ja Microsoft Internet Explorer. Ne kaikki sisältävät lähes samanlaiset kehitystyökalut. Ne ovat myös tällä hetkellä eniten käytetyt työpöytäselaimet ja antavat kattavan kuvan eri selainympäristöistä. [29.] Safari-selainta ei käytetty testeissä, koska Electric Mobile Studio 2012 -emulaattori soveltui paremmin iPad- ja iPhone-laitteiden testaamiseen ja sisälsi kaikki tarvittavat kehitystyökalut.

4.6.5 Oikeat laitteet

Vaikka emulaattorit ja työpöytäselaimet ovatkin erinomaisia web-kehityksen työkaluja, paljastuvat todelliset ongelmat yleensä vasta oikeilla laitteilla testattaessa. Mobiiliportaalia testattiin Windows Phone 7.8-, iOS 7.1- ja Android 2.3-alustoilla. Testeissä käytettyjä mobiililaitteita olivat Nokia Lumia 800, iPad Mini ja HTC Wildfire. Käytetyt laitteet antoivat hyvän kuvan portaalin toiminnasta erilaisissa laiteympäristöissä. Testauksen tarkoitus oli varmistaa portaalin tekninen toimivuus kehityksen yhteydessä, joten erillistä käyttäjätestausta ei tarvittu.

Laitevalinnoilla pyrittiin kattamaan erilaisen suorituskyvyn laitteet ja mahdollisimman suuri osa yleisimmistä mobiilialustoista. Uusimmissa markkinakatsauksissa Android-laitteet ovat selvästi myydyimpiä mobiililaitteita [30]. Seuraavaksi eniten myydään iOS- ja kolmanneksi Windows Phone -laitteita. Muiden käyttöjärjestelmien osuus on niin pieni, ettei niitä kannata erikseen testata. Android-laitteita testeissä edusti suorituskyvyltään heikko HTC Wildfire. Laitteen tehtävä oli testata mobiiliportaalin toimintaa vanhemmilla laitteilla. Uudempien Android-laitteiden odotettiin käyttäytyvän työpöytäselainten ja emulaattorien tapaan, joten niitä ei erikseen testattu. iOS-pohjaisia laitteita edusti

iPad Mini, jonka tehtävä oli testata mobiiliportaalin toimintaa iOS-alustalla. Tehokas laite antoi myös parhaan kuvan mobiiliportaalin tulevista käyttöympäristöistä. Windows Phone -laitteita edusti Nokia Lumia 800. Kotimaisen valmistuksen vuoksi Lumia-malleilla odotettiin olevan kohtuullisen suuri käyttäjäryhmä Suomessa. Windows Phone -alusta eroaa myös sen verran paljon muiden valmistajien käyttöjärjestelmistä, että sen testaaminen oli perusteltua.

Suurimmat ongelmat löytyivät Windows Phone- ja iOS-laitteista. Windows Phone 7.8:n internet-selain pohjautuu Microsoftin Internet Explorer 9:ään. Se toimii melko hyvin jQuery Mobilen kanssa, mutta tuetut CSS-ominaisuudet eroavat hieman muiden valmistajien toteutuksesta. Tästä syystä visuaalinen ulkonäkö on syytä tarkistaa eri mobiilialustojen kesken. Android-ympäristöön voidaan asentaa lähes mikä tahansa selain, joten yhteensopivuutta rajoittaa lähinnä laitteen suorituskyky. Testauksessa käytetty HTC Wildfire edustaa suorituskyvyssä kosketusnäytöllisten laitteiden häntäpäätä, joten sen hitaus oli odotettavissa. Mobiiliportaali on kuitenkin käyttökelpoinen myös hitaammilla laitteilla.

Windows Phone -puhelimet tarkistavat sertifikaatit tarkemmin kuin muut mobiililaitteet. Tästä seurasi ongelmia käytettäessä ns. itse allekirjoitettuja varmenteita. Koska jQuery Mobilen sivut ladataan AJAX-kyselyinä, ei virheellistä varmennetta päässyt ohittamaan, eikä portaalissa päässyt etenemään kirjautumista pidemmälle. Vika korjaantui korvaamalla varmenteet allekirjoitetuilla versioilla. Toinen Internet Explorerin aiheuttama ongelma oli sivujen tallentaminen selaimen välimuistiin. Kun Windows Phone oli tallentanut sivun selaimen välimuistiin, se ladattiin aina sieltä, vaikka sisältö olisikin muuttunut. Ongelman ratkaisemiseksi piti IIS-palvelimelle lisätä välimuistiin tallentamisen kieltäviä otsikkotietoja. Muilla mobiililaitteilla ei samoja ongelmia esiintynyt, joten ongelmien syiden paikallistaminen oli varsin haastavaa.

Yksi hankalimmista ongelmista selvittää oli iPad Minillä ilmenevä vika JavaScript-tapahtumankäsittelyssä. Vika ilmeni siten, että ruutua näpäytettäessä saattoi selain suorittaa satunnaisesti myös aiempia tapahtumia uudestaan, jolloin toimintoja suoritettiin moneen kertaan. Syy vikaan löytyi Safari-selaimen tapahtumapinosta, joka sotkeutui ilmoituksien näyttämisestä ontouchstart- ja ontouchend-tapahtumien välissä. Tämä rajoittaa kyseisten tapahtumien käyttöä mobiiliportaalin käyttöliittymässä. Muuten iPad Mini oli testilaitteiston parhaimmistoa ja soveltui erinomaisesti mobiiliportaalin käyttöön.

5 Tulokset

Projektin tavoite oli tuottaa palvelualusta mobiilikäyttöön, joka on yhteensopiva eri mobiililaitteilla. Sen tuli olla myös helppokäyttöinen, suorituskykyinen, turvallinen ja helposti ylläpidettävä.

jQuery Mobilen ja progressiivisen kehityksen aikaansaama yhteensopivuus tekevät mobiiliportaalista erittäin yhteensopivan järjestelmän. ASP.NET MVC tukee jQuery Mobilen arkkitehtuuria ja takaa turvallisen ja helposti ylläpidettävän palveluympäristön. jQuery Mobilen arkkitehtuuri ja JavaScript-ohjelmointi web-sivujen yhteydessä vaatii hieman enemmän suunnittelua ja on selvästi vielä kehitysvaiheessa. Selainten erilaiset toteutukset ja koodin suorituksen arvaamattomuus aiheuttavat päänvaivaa testauksessa. jQuery Mobile pyrkii kuitenkin huolehtimaan näistä yhteensopivuuden perusongelmista ja tarjoamaan vakaan ohjelmointialustan. Käyttämällä testattuja vakiokomponentteja voidaan portaalin palveluita rakentaa luotettaviksi ja toimintavarmoiksi.

Mobiiliportaali pyrittiin toteuttamaan helppokäyttöiseksi ja suorituskykyiseksi. Käyttöliittymän kontrollien suuret pinta-alat ja yksinkertaiset muodot mahdollistavat käytön myös pienemmillä laitteilla. Käyttöliittymä on pyritty toteuttamaan niin, että se toimii hyvin keskitason laitteilla. Jos mobiiliportaalin palveluita suunnitellaan käytettäväksi myös vanhemmilla laitteilla, täytyy tämä ottaa huomioon palveluiden toteutuksessa ja välttää toimintojen suorittamista asiakaspäässä. Uudemmilla laitteilla voidaan hyödyntää portaalin koko potentiaalia. Mobiililaitteen tarjoamaa paikkatietoa voidaan hyödyntää tarjoamalla asiakkaan sijaintiin parhaiten sopivaa tietoa. Kosketusnäyttöä voidaan käyttää helpottamaan karttapalveluiden tai kuva-arkistojen käsittelyä. Sähköposti- tai puhelinnumerolistoista voidaan suoraan valita vastaanottaja puhelua tai viestiä varten. Portaalista voidaan tarjota linkkejä muihin sovelluksiin ja muista sovelluksista voidaan linkittää portaalin palveluihin.

Projektin tuotetta voidaan pitää tavoitteisiin nähden onnistuneena. Web-kehitys mahdollistaa varteenotettavan vaihtoehdon mobiilialustakohtaisille erillisohjelmistoille. Se laajentaa potentiaalista asiakasryhmää ja helpottaa ohjelmiston ylläpidettävyyttä. Yksi kehitysympäristö mahdollistaa myös nopeammat muutokset ja ominaisuuksien kehittämisen kaikille mobiilialustoille samanaikaisesti. Koska web-ympäristö ei ole riippuvainen käytettävästä mobiilialustasta, on ohjelmiston elinkaari pitkäikäisempi ja saman ohjelmiston voidaan olettaa toimivan myös tulevaisuuden laitteilla. Mobiiliportaalin pal-

velinohjelmistoa voidaan laajentaa ja käyttöliittymäkomponentteja standardoida testatuiksi moduuleiksi. Tämä nopeuttaa tuotekehitystä ja parantaa ylläpidettävyyttä. Alusta mahdollistaa integroinnin mihin tahansa järjestelmään, joten se mahdollistaa minkä tahansa palvelun toteuttamisen mobiililaitteilla käytettäväksi. Portaali tarjoaa siis hyvät edellytykset jatkokehitykselle ja mahdollistaa web-palveluiden siirtämisen mobiiliaikaan.

Lähteet

1 Maximiliano, Firtman. 2010. Programming the Mobile Web. 1. painos. Sebastopol: O'Reilly Media.

2 Gardner, Lyza Danger & Grigsby, Jason. 2012. Head First Mobile Web. 1. painos. Sebastopol: O'Reilly Media.

3 Afana, Nadeem. 2011. ASP.NET MVC 5 Internationalization. Verkkodokumentti. Nadeem Afana's blog. <http://afana.me/post/aspnet-mvc-internationalization.aspx>. Päivitetty 14.1.2011. Luettu 11.1.2014.

4 Grigorik, Ilya. 2012. Latency: The New Web Performance Bottleneck. Verkkodokumentti. Ilya Grigorik. <http://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/>. Päivitetty 19.6.2012. Luettu 11.1.2014.

5 Raykov, Nikolay. 2010. Using IIS7 Output Caching Capabilities. Verkkodokumentti. Some thoughts on web 2.0 development. <http://nraykov.wordpress.com/2010/04/08/using-iis-7-output-caching-capabilities/>. Päivitetty 8.4.2010. Luettu 12.1.2014.

6 Resig, John. 2006. Pro JavaScript Techniques. Berkeley: Apress.

7 Freeman, Adam. 2012. Pro jQuery. E-kirja. New York: Apress.

8 Maximiliano, Firtman. 2012. jQuery Mobile: Up and Running. E-kirja. Sebastopol: O'Reilly Media.

9 API Documentation: Data Attributes. 2014. Verkkodokumentti. The jQuery Foundation. <http://api.jquerymobile.com/data-attribute/>. Luettu 12.1.2014.

10 Freeman, Adam. 2012. Pro ASP.NET MVC 4. E-kirja. New York: Apress.

11 Galloway, Jon, Haack, Phil, Wilson, Brad & Allen, K. Scott. 2012. Professional ASP.NET MVC 4. E-kirja. Birmingham: Wrox Press.

12 Bender, James & McWherter, Jeff. 2011. Professional Test-Driven Development with C#: Developing Real World Applications with TDD. Indianapolis: Wiley Publishing.

13 Sintes, Anthony. 2002. Sams Teach Yourself Object Oriented Programming in 21 Days. Indianapolis: Sams Publishing. 2. painos.

14 Microsoft Corporation. 2003. MCSD Self-Paced Training Kit: Analysing Requirements and Defining Microsoft .NET Solution Architectures, Exam 70-300. Redmond: Microsoft Press.

15 Wheeler, A. David. 2003. Secure Programming for Linux and Unix HOWTO. Verkkodokumentti. The Linux Documentation Project. <http://www.tldp.org/HOWTO/Secure-Programs-HOWTO/web-authentication.html>. Päivitetty 3.3.2003. Luettu 18.1.2014.

16 Extensible Markup Language (XML). 2008. Verkkodokumentti. World Wide Web Consortium (W3C). <http://www.w3.org/TR/REC-xml/>. Päivitetty 7.2.2013. Luettu 6.3.2014.

17 .NET Framework 4. 2014. Verkkodokumentti. Microsoft. <http://msdn.microsoft.com/en-us/library/w0x726c2(v=vs.100).aspx>. Luettu 6.3.2014.

18 Templin, Reagan. 2007. Introduction to IIS Architectures. Verkkodokumentti. Microsoft. <http://www.iis.net/learn/get-started/introduction-to-iis/introduction-to-iis-architecture>. Päivitetty 16.11.2007. Luettu 6.3.2014.

19 Zemke, Fred. 2012. What's new in SQL:2011. Verkkodokumentti. Oracle Corporation. <http://www.sigmod.org/publications/sigmod-record/1203/pdfs/10.industry.zemke.pdf>. Luettu 6.3.2014.

20 CSS Color Module Level 3. 2011. Verkkodokumentti. World Wide Web Consortium (W3C). <http://www.w3.org/TR/2011/REC-css3-color-20110607/#introduction>. Päivitetty 7.6.2011. Luettu 6.3.2014.

21 The Transport Layer Security (TLS) Protocol Version 1.2. 2008. Verkkodokumentti. Internet Engineering Task Force (IETF). <http://tools.ietf.org/html/rfc5246>. Luettu 6.3.2014.

22 Hypertext Transfer Protocol – HTTP/1.1. 1999. Verkkodokumentti. World Wide Web Consortium (W3C). <http://www.w3.org/Protocols/rfc2616/rfc2616.html>. Luettu 6.3.2014.

23 HTML5. 2014. Verkkodokumentti. World Wide Web Consortium (W3C). <http://www.w3.org/TR/html5/introduction.html#introduction>. Päivitetty 4.2.2014. Luettu 6.3.2014.

24 Standard ECMA-404. 2013. Verkkodokumentti. Ecma International. <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>. Luettu 6.3.2014.

25 W3C DOM4. 2014. Verkkodokumentti. World Wide Web Consortium (W3C). <http://www.w3.org/TR/domcore/>. Päivitetty 4.2.2014. Luettu 6.3.2014.

26 Standard ECMA-262. 2011. Verkkodokumentti. Ecma International. <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>. Luettu 8.3.2014.

27 The jQuery Team. 2014. Verkkodokumentti. jQuery Foundation. <https://jquery.org/team/>. Luettu 8.3.2014.

28 About jQuery Mobile. 2014. Verkkodokumentti. jQuery Foundation. <http://jquerymobile.com/about/>. Luettu 8.3.2014.

29 Browser Statistics. 2014. Verkkodokumentti. W3Schools. <http://www.w3schools.com/browsers/browsers_stats.asp>. Luettu 8.3.2014.

30 Whitney, Lance. 2014. iPhone market share shrinks as Android, Windows Phone grow. Verkkodokumentti. CNET. <http://news.cnet.com/8301-13579_3-57616679-37/iphone-market-share-shrinks-as-android-windows-phone-grow/>. Päivitetty 6.1.2014. Luettu 8.3.2014.