VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Anssi Lumme

WEB-APPLIKAATIO TOIMINNAN -OHJAUSJÄRJESTELMÄN OSANA

Tekniikka
2018

VAASAN AMMATTIKORKEAKOULU
Tieto- ja tietoliikennetekniikka

TIIVISTELMÄ

Tekijä	Anssi Lumme
Opinnäytetyön nimi	Web-applikaatio toiminnanohjausjärjestelmän osana
Vuosi	2018
Kieli	suomi
Sivumäärä	29
Ohjaaja	Pirjo Prosi

Opinnäytetyön tilaajana oli Visma Software Oy. Työ toteutettiin osaksi Visma Nova -ohjelmistoa ja työn tarkoituksena oli selvittää potentiaaliset teknologiat web-applikaation liittämiseksi osaksi Visma Nova -ohjelmistoa. Lisäksi tarkoitus oli luoda testisovellus, jolla voi demonstroida rajapinnan mahdollisuuksia ja joustavuutta käyttäen esimerkkinä varastokirjanpidon toiminnallisuutta.

Web-applikaatio koostuu kahdesta osasta: web-palvelu ja web-sovellus, joista jälkimmäinen toimii käyttöliittymänä. Molemmat ovat omia kokonaisuuksiaan, vaikka toimivatkin yhdessä. Web-palvelu on rajapinta, johon voi ottaa yhteyttä muullakin kuin web-sovelluksella, esimerkiksi mobiilisovelluksella. Web-sovelluksen tarkoitus on toimia esimerkkinä rajapinnan toiminnoista. Käytettyinä teknologioina oli ASP.NET Web API 2 ja ASP.NET MVC 5.

Web-palvelua on helppo jalostaa toteuttamaan muitakin toimintoja varastokirjanpidon ulkopuolelta. Web-sovellus on toteutettu responsiiviseksi, eli se sopii sekä isoille että pienille näytöille.

Avainsanat Web-palvelu, Web-sovellus, MVC 5, Web API 2

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tieto- ja tietoliikennetekniikka

ABSTRACT

Author	Anssi Lumme
Title	Web Application as a Part of ERP Platform
Year	2018
Language	Finnish
Pages	29
Name of Supervisor	Pirjo Prosi

The thesis was made for Visma Software Ltd. The thesis was made as a part of Visma Nova -software and the main purpose was to examine the potential technologies to merge web application as a part of Visma Nova. Additionally, the goal was to develop a test application, which would be used to demonstrate the possibilities and flexibility of the interface and it would serve as an example of the functionality of stock management application.

The web application consists of two main parts: web service and web-based user interface. Albeit both being their separate entities, they work closely together. The web service is an interface, which can be used from other applications as well, such as mobile applications. The purpose of the web-based user interface is to provide an example of what the web service can be used for. The technologies used for this project were ASP.NET Web API 2 and ASP.NET MVC 5.

The web service was made easy to expand to provide other functionality outside from stock management. The web-based user interface was responsive, which means it fits well to both small and large displays.

Keywords Web service, Web application, MVC 5, Web API 2

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

LYHENTEET JA TERMIT

.NET	Microsoftin kehittämä framework.
ARM	Advanced RISC Machines. Mikroprosessoriarkkitehtuuri.
Assembly	Konekieli. Ohjelmointikielten alempi muoto.
Binääri	Numerojärjestelmä, jossa lukujen arvo voi olla vain 0 tai 1.
C	Yleiskäyttöinen ohjelmointikieli.
C++	C-ohjelmointikieleen perustuva ohjelmointikieli.
C#	Oliopohjainen ohjelmointikieli, joka hyödyntää .NET -teknologiaa.
Cross-platform	Kehitystapa, jolla sovellus toimii usealla eri alustalla yhdellä ohjelmakoodilla.
CSS	Cascading Style Sheets. Sitä käytetään kuvaamaan dokumentin ulkonäköä ja esitysmuotoja.
Framework	Ohjelmistokehys. Kirjasto, joka sisältää erilaisia komponentteja ohjelmiston kehittämiseen.
HTML	Hypertext Markup Language, kuvauskieli, jota käytetään esittämään web-sivun rakennetta.

HTTP	Hypertext Transfer Protocol. Protokolla, jota sovellukset, kuten web-selaimet käyttävät tiedonsiirtoon.
IDE	Integrated Development Environment, ohjelmointiympäristö.
IIS	Internet Information Services. Microsoftin kehittämä yleiskäyttöinen web-palvelin.
IL	Intermediate Language. Oliopohjainen konekieli.
Java	Oliopohjainen ohjelmointikieli, jota ajetaan Java -virtuaalikoneella.
Javascript	Dynaaminen komentosarjakieli.
JNI	Java Native Interface. Framework, joka mahdollistaa JVM (Java virtuaalikone) -sovelluksille kyvyn kutsua natiiveja sovelluksia.
JSON	Javascript Object Notation. Tiedonvälitykseen käytettävä yksinkertainen tiedostomuoto.
Kotlin	Ohjelmointikieli, jota ajetaan Java -virtuaalikoneella.
Mono	Avoimen lähdekoodin .NET -framework cross-platform kehittämiseen.
Objective-C	Oliopohjainen ohjelmointikieli. Perustuu C-ohjelmointikieleen.

OWIN	Open Web Interface for .NET. Määritelmä tavalle rakentaa ASP.NET web-sovelluksia, jolla ne eivät ole riippuvaisia IIS -web-palvelimesta.
Sandbox	Tapa, jolla sovellusta ajetaan omassa ympäristössään eristettynä muista sovelluksista.
SDK	Software Development Kit. Kokoelma työkaluja sovellusten kehittämiseen.
Swift	Ohjelmointikieli, joka on suunniteltu korvaamaan C-pohjaiset ohjelmointikielet.
URI	Uniform Resource Identifier. Osoite, joka määrittää datan sijainnin.
Visual Basic	Oliopohjainen ohjelmointikieli, joka hyödyntää .NET frameworkiä.

1 JOHDANTO

Työn tarkoituksena on toimia osviittaa antavana demotyönä Visma Software Oy:n Vaasan Visma Nova -tiimille. Työn tarkoitus on olla esimerkki siitä, mitä Visma Novan rajapinnalla voi toteuttaa. Demosovelluksen alustaksi valittiin web, koska niin saadaan toiminnallisuus yhtenäisemmin isommalle kohderyhmälle. Näin myös laiterajoitukset ovat vähäisemmät. Sovelluksessa demonstroitiin Visma Novan varastokirjanpidon perusominaisuuksia. Lisäksi työn avulla selvitettiin potentiaaliset toteutuksen teknologiat Visma Nova-tiimiä varten.

Toiminnanohjausjärjestelmä eli ERP (Enterprise Resource Planning) yhdistää yritystoiminnan eri prosessit yhteen palveluun. Toiminnanohjausjärjestelmät tarjoavat reaaliaikaista tietoa yrityksen eri toiminnoista. Näistä toiminnoista esimerkkinä talousasiat ja henkilöstön hallinta. /1/

Visma on Pohjoismaiden johtava yritysohjelmistojen ja IT-konsultoinnin tarjoaja. Visma on perustettu vuonna 1996 Norjan Oslossa. Vuonna 2001 Visma osti Liinos Oyj:n, joka vaihtoi nimensä Visma Software Oy:ksi. Suomessa Visma työllistää yli 900 työntekijää. /2/

Visma Software Oy:n palvelut koostuvat toiminnanohjaus- ja taloushallintajärjestelmista ja niihin liittyvistä koulutus- ja ylläpitopalveluista. Visma Software Oy:n pääasiallinen kohderyhmä on pk-yritykset. Sen tuotteita ovat esimerkiksi Visma Nova ja Visma.net. /3/

2 TEKNOLOGIOIDEN VALITSEMINEN

Työn alussa tuli selvittää käytettävät teknologiat. Tämä tapahtui käytännössä eliminoimalla eri teknologioita käytännöllisistä syistä, joista esimerkkinä jo käytössä olevien järjestelmien rajoitukset ja yhteensopivuus. Aluksi punnittiin vaihtoehtoa, että demosovellus olisi mobiiliapplikaatio web-applikaation sijaan. Näin ollen myös mahdollisen mobiiliapplikaation toteuttamisen mahdollistavat teknologiat valittiin mukaan vertailuun. Mobiiliapplikaation teknologioita ajateltiin cross-platform toteutuksen kannalta, vaikka alkuun punnittiin myös mahdollisuutta natiiveihin sovelluksiin. Tärkeä huomioon otettava seikka oli se, että työn tilaajalla oli ennestään Microsoftin .NET -teknologioita ja yleisesti Microsoftin ekosysteemi käytössä.

Ensimmäisenä tutkittiin natiivien mobiiliapplikaatioiden mahdollisuutta Android- ja iOS -alustoille. Kohteena olisivat nämä kaksi alustaa, koska niiden käyttäjämäärät täyttävät ylivoimaisen enemmistön mobiilimarkkinoilla. /4/

Natiivina ajettavassa sovelluksessa olisi mahdollisesti parempi käyttövarmuus kuin cross-platform sovelluksessa. Isona hyötynä olisi myös se, että molemmille alustoille olisi kaikki ohjelmointikirjastot käytettävissä. Tämä helpottaisi toteutusta ja ylläpitoa huomattavasti. /5/

2.1 Android (natiivi)

Android-käyttöjärjestelmä pohjautuu Linuxiin, joka on avointa lähdekoodia. Androidin kehitykseen osallistuu Linux yhteisön jäseniä ja satoja eri yhteistyökumppaneita. Androidin avoimuus on tehnyt siitä suositun, niin kehittäjien kuin kuluttajien keskuudessa. /6/

Natiiveja Android -sovelluksia voidaan kirjoittaa Kotlin-, Java- tai C++ -ohjelmointikielellä. Työn tekohetkellä Kotlin ei kuitenkaan ollut vielä virallisesti tuettu ohjelmointikieli, joten vaihtoehdot olivat näin ollen rajattu Javaan ja C++: aan. Kielien luonteen vuoksi Java olisi joka tapauksessa ollut luonnollisempi vaihtoehto, sillä se muistuttaa hyvin paljon C# -ohjelmointikieltä. /6-7/

Android -sovellukset ajetaan sandbox-ympäristössä. Android-käyttöjärjestelmä on usean käyttäjän järjestelmä, jossa jokainen sovellus toimii omana käyttäjänään. Näin varmistetaan, että sovellus ei pääse käsiksi toisen sovelluksen tiedostoihin ilman erillistä lupaa. Oletusarvoisesti jokainen sovellus on oma Linux -prosessinsa. Jokainen prosessi ajetaan omassa virtuaalikoneessaan, jolloin eri prosessit ovat erossa toisistaan. Tämä lisää tietoturvaa huomattavasti. /8/

Android -sovellukset koostuvat komponenteista. Näitä komponentteja on neljää laatua: Activity, Service, Broadcast receiver ja Content provider.

Activityt ovat yhden sivun näkymiä sovelluksen käyttöliittymästä. Sovelluksessa voi olla useita Activityjä. Tästä esimerkkinä pikaviestisovellus, jossa yhteystietoluettelo on yksi Activity ja keskustelunäkymä on toinen Activity. Activity kommunikoi järjestelmän kanssa ja näin pitää huolen siitä, että vaadittavat prosessit pysyvät käynnissä pitääkseen kyseisellä ajanhetkellä olevan Activityn laitteen näytöllä. Lisäksi kommunikaatio mahdollistaa aikaisempiin Activityihin palaamisen ja palauttaa niiden aikaisemman tilan. /8-9/

Servicet ovat yleisesti sovelluksen taustapalveluita. Palveluilla ei ole käyttöliittymää vaan ne toimivat taustalla ja suorittavat esimerkiksi pitkän aikavälin prosesseja, kuten vaikka etätietokannan manipulointia. Tämän ansiosta käyttöliittymän käyttö ei häiriinny, kun esimerkiksi tietokannan haut suoritetaan taustalla omassa prosessissaan, erillään käyttöliittymän prosessista. Käyttäjä ei usein ole edes tietoinen taustasovelluksen olemassaolosta. /8-9/

Broadcast receiver -komponentti mahdollistaa koko järjestelmän laajuisten tapahtumien käsittelyn. Esimerkiksi jos käyttäjä ottaa näyttökuvan, Broadcast receiver ilmoittaa siitä järjestelmälle. Tämän tyyppisiä ilmoituksia kuunteleva pilvitallennustila -sovellus antaa käyttäjälle ilmoituksen jossa kysytään, haluaako käyttäjä tallentaa näyttökuvan pilveen. /8-9/

Content provider tarjoaa nimensä mukaisesti sisältöä sovellukselle. Esimerkiksi yhteystiedot tai kuvagallerian kuvat, riippuen mitä lupia sovellukselle on annettu. Content provider on myös kätevä tapa käsitellä sovelluksen yksityistä dataa.

Esimerkiksi muistiosovelluksen dokumentit voivat olla yksityisiä, jolloin ne ovat vain kyseisen sovelluksen käytössä. /8-9/

Android -käyttöjärjestelmälle uniikki ominaisuus on se, että minkä tahansa sovelluksen minkä tahansa komponentin voi avata mistä tahansa sovelluksesta. Esimerkiksi jos pikaviestisovelluksesta haluaa jakaa videon, voi sovelluksen kautta avata suoraan kamerasovelluksen videokuvauksen. Videon kuvaamisen jälkeen kamerasovellus sulkeutuu ja palaa takaisin pikaviestisovellukseen. Otettu videoleike välittyy pikaviestisovellukseen siirtymän mukana. Tällaisesta arkkitehtuurista johtuen, Android-sovelluksissa ei ole ollenkaan main() -funktiota, toisin kuin monien muiden käyttöjärjestelmien sovelluksissa. Sovellukset eivät kuitenkaan voi suoraan avata toisten sovelluksien komponentteja. Sovelluksen pitää antaa järjestelmälle pyyntö avata toisen sovelluksen komponentti, jonka jälkeen järjestelmä avaa tämän pyydetyn komponentin, ellei sitä ole erikseen estetty toisen kyseessä olevan sovelluksen toimesta. /9/

Koska Android -käyttäjät usein vaihtelevat sovelluksia kesken käytön, on sovellusten oltava rakenteeltaan joustavia. Eri komponentit tulee olla mahdollista käynnistää missä tahansa järjestyksessä ja yksinään. Komponentteihin ei myöskään tule tallentaa mitään informaatiota. Syynä tähän on se, että käyttäjän ja käyttöjärjestelmän tulee voida sulkea käytössä olevia komponentteja tarpeen vaatiessa. Myöskään komponentit eivät saa olla riippuvaisia toisistaan vaan niiden tulee toimia itsenäisinä kokonaisuuksina. /9/

Jotta käyttöjärjestelmä osaisi avata sovelluksen komponentit, eri komponentit tulee esitellä manifest -tiedostossa. Tässä esittely käytännössä tarkoittaa sitä, että järjestelmä saa tietää tarvittavien komponenttien olemassaolon. Sovelluksen manifest -tiedosto tulee sijoittaa sovelluksen hakemiston juureen. Komponenttien esittelyn lisäksi manifest-tiedosto sisältää paljon muutakin informaatiota. Esimerkiksi, kaikki sovelluksen vaatimat luvat ilmoitetaan manifest-tiedostossa. /9/

Android -sovelluksien tulee siis olla modulaarisia. Yhtenä tärkeimpänä asiana tulee ottaa huomioon, että Activity -luokissa on ainoastaan käyttöliittymää käsittelevää koodia. Varsinkin aloittelijoiden keskuudessa on yleinen virhe, että koko ohjelman

koodi laitetaan toiseen edellä mainituista luokista. On myös hyvä pitää riippuvuudet näihin luokkiin mahdollisimman vähäisinä, jotta voidaan tarjota sujuva käyttökokemus loppukäyttäjälle. /9/

Datan säilyttämisen ja mahdollisten verkkoyhteyskatkojen takia on suositeltavaa käyttää Model -luokkia. Nämä Model -luokat käsittelevät kaiken sovelluksen datan. Model-luokat ovat eristettyinä sovelluksen komponenteista. /9/

2.2 iOS (natiivi)

iOS-sovelluksien ohjelmointikielinä toimii Swift ja Objective-C. Nykyään Apple kuitenkin suosittelee käyttämään Swiftiä. Swift pohjautuu C- ja Objective-C -ohjelmointikieliin. /10/

iOS-sovelluksien kehittämiseksi kehittäjällä tulee olla Intel-pohjainen Macintosh -tietokone. Lisäksi kehittäjän tulee noudattaa Applen laatimia sääntöjä ja vaatimuksia iOS-sovelluksiin liittyen. Esimerkkinä, sovelluksessa tulee aina olla tuki kaikkein uusimmille iOS-laitteille. /10/

Jokaisessa iOS-sovelluksessa tulee olla information property list -tiedosto, ilmoitus sovelluksen ominaisuuksista laitteiston suhteen (vrt. Permissionit Androidissa), yksi tai useampi sovelluskuvake sekä kuvatiedosto sovelluksen käynnistämistä varten. /11/

Information property list -tiedosto sisältää tietoa sovelluksen konfiguraatiosta ja resursseista. Tätä tiedostoa hyödynnetään myös iOS -käyttöjärjestelmän sovelluskaupassa, App Storessa. Xcode -IDE luo suuren osan tiedoston sisällöstä automaattisesti, mutta kehittäjän tulee varmistaa, että tiedot ovat oikein ja niissä ei ilmene vajavaisuuksia. Lisäksi tiedosto sisältää esittelyt sovelluksen laitetason ominaisuuksista, esimerkiksi, jos sovelluksen on määrä hyödyntää laitteen kameraa. Näiden tietojen ohella kehittäjän tulee tarjota käyttäjälle kuvaus siitä, mihin sovellus tarvitsee kyseistä ominaisuutta. Tämä kuvaus annetaan ominaisuuden esittelyn yhteydessä. /11/

Sovelluskuvake tarvitaan sovelluksen näyttämiseen käyttöjärjestelmän kotinäkymässä. Tämän lisäksi tulee tarjota kuvatiedosto, joka näytetään käyttäjälle sovelluksen käynnistyessä samalla kun odotetaan itse sovelluksen käynnistymistä. Näin käyttäjä saa heti tietoonsa, että hänen komentonsa sovelluksen avaamiseksi on rekisteröity ja sovelluksen käyttöliittymää avataan parhaillaan. Kun sovellus siirtyy etualalta taustalle, järjestelmä ottaa käyttöliittymästä kuvan. Tämä kuva näytetään kehittäjän tarjoaman kuvatiedoston sijasta mikäli mahdollista, kun sovellus siirtyy takaisin etualalle. Mikäli sovellus on ollut pitkään taustalla tai käyttäjä on sulkenut sen, järjestelmä poistaa ottamansa kuvan ja käyttää kehittäjän tarjoamaa kuvatiedostoa, kun sovellus seuraavan kerran avataan. /11/

Kun käytetään käyttäjän tai laitteen dataa, tulee siitä informoida käyttäjälle. Lisäksi tulee noudattaa vaadittavia lainsäädännöllisiä asetuksia datan käytön ja suojaamisen suhteen. Nämä lainsäädännölliset asetukset vaihtelevat alueittain. Käyttäjän lupa tulee kysyä vain sillä hetkellä, kun sovellus tarvitsee dataa. Mikäli sovellus säilyttää kerättyä dataa paikallisesti, voi kehittäjä hyödyntää sovelluksessaan iOS:n omaa datan suojausominaisuutta. Tämä ominaisuus salaa datan ja säilöö sen laitteeseen. /11/

Kun sovellus käynnistetään, se siirtyy ei-aktiivisesta tilasta aktiiviseen tai taustalla olevaan tilaan riippuen sovelluksen tarkoituksesta. Sovelluksen käynnistymisen aikana järjestelmä avaa sovellukselle oman prosessin ja kutsuu sovelluksen main()-funktiota omassa säikeessään. Main()-funktio antaa heti hallinnan UIKit -frameworkille, jonka tehtävä on valmistella ja avata käyttöliittymän komponentit laitteen näytölle. Jos sovellus käynnistetään taustalle, järjestelmä lataa sovelluksen käyttöliittymätiedostot, mutta ei näytä niitä laitteen ruudulla. /12/

Kuten kaikissa C-kieleen pohjautuvissa sovelluksissa, iOS-sovellusten suoritus alkaa main()-funktiosta. Tässä sen ainoa tehtävä on luovuttaa kontrolli UIApplicationMain-funktiolle, joka hyödyntää UIKit-frameworkiä ja suorittaa käyttöliittymän luomisen. UIApplication -objekti toimii jokaisen iOS-sovelluksen perustana. Se hallinnoi järjestelmän ja sovelluksen välistä kommunikaatiota. UIApplication luo

myös main run loopin. Main run loop on silmukka, joka prosessoi kaikki käyttäjään liittyvät sovellustapahtumat. /13/

Sovellus tulee rakentaa siten, että käyttäjän data tallennetaan ja tärkeät toiminnot suoritetaan heti kun se on mahdollista. Näin sovellus varautuu siihen, että järjestelmä tai käyttäjä voi sulkea sovelluksen koska tahansa. Kun sovellus suljetaan, se tapahtuu välittömästi eikä järjestelmä anna siitä sovellukselle erillistä tiedoksiantoa. Kun sovellus suljetaan, järjestelmä sulkee prosessin ja antaa sovelluksen käytössä olleen käyttömuistin muiden sovellusten tai järjestelmän omaan käyttöön. /13/

2.3 Apache Cordova

Apache Cordova on Apache Software Foundationin kehittämä avoimen lähdekoodin framework mobiiliapplikaatioiden luomiseen. Sen avulla sovelluksien kehityksessä voi käyttää web-kehityksen tekniikoita kuten HTML5, CSS ja Javascript. Apache Cordovalla on mahdollista luoda cross-platform-mobiilisovelluksia. /14/

Pohjimmiltaan Apache Cordovalla luodut sovellukset ovat web-sovelluksia. Nämä web-sovellukset hyödyntävät Cordovan omia ohjelmointirajapintoja, joiden avulla kehittäjä pääsee käsiksi laitteen eri komponentteihin, kuten esimerkiksi kameraan ja mikrofoniin. Cordovan ohjelmointirajapinnat puolestaan hyödyntävät laitteen käyttöjärjestelmän rajapintoja, esimerkiksi ilmoitusten esittämiseen. Tämä siis mahdollistaa sovelluksen kommunikoinnin laitteen käyttöjärjestelmän kanssa helpottaen kehittäjän työtä. /14-15/

Apache Cordova oli potentiaalinen vaihtoehto, koska yhdellä koodilla, käyttöjärjestelmäkohtaisia kirjastoja lukuun ottamatta, olisi saanut luotua sovellukset sekä iOS:lle että Androidille. Lisäksi samoja komponentteja olisi voinut hyödyntää tulevaisuudessa mahdollisessa selainpohjaisessa ratkaisussa. Näin sovellusten ulkoasu olisi helpompi pitää yhtenäisenä, jolloin eri alustojen loppukäyttäjälle käyttökokemus olisi eheämpi. /14-15/

Itse sovellus käytännössä on web-sivu, jota ajetaan WebView-komponentilla. Näin web-sivu naamioidaan kuin se olisi tavallinen mobiilisovellus. WebView toimii tavallisen internetselaimen tavoin, joka suorittaa web-sivun sillä erotuksella,

että WebView on nimensä mukaisesti näkymä, joka esittää vain kyseisen web-sivun ja toteuttaa sen. Tämän vuoksi web-sivu joutuu ensin kommunikoimaan WebViewin kanssa, joka puolestaan kommunikoi laitteen käyttöjärjestelmän kanssa. Siksi Apache Cordovalla tehdyt sovellukset ovat suorituskyvyltään heikompia ja hitaampia kuin natiivit sovellukset. /14-15/

2.4 Xamarin

Kuten Apache Cordova, myös Xamarin toimii sekä iOS:llä että Androidilla. Web teknologioiden sijasta Xamarin käyttää ohjelmointikielenä C#:ia ja hyödyntää Mono .NET frameworkiä. Xamarin ei myöskään toimi vain ikkunana itse sovellukselle, vaan sillä tehdyt sovellukset ovat kohdekäyttöjärjestelmästä riippuen joko natiiveja tai integroituja .NET-sovelluksia, jotka ajetaan mukana tulevalla ajoympäristöllä. /16/

Kun sovellus käännetään iOS:lle, siitä tehdään natiivi kääntämällä suoraan ARM assembly -kielelle. Tämä siksi, että Apple ei salli ajoympäristöjen generoimista iOS:llä. Kaikkia kielellisiä ominaisuuksia ei ole mahdollista käyttää iOS:n rajoituksien vuoksi. Jotta Xamarinilla voisi kehittää iOS -sovelluksia, tulee kehittäjällä olla Mac -tietokone, jossa on käyttöjärjestelmänä macOS. /16/

Androidille käännettäessä sovelluksista tulee .NET sovelluksia, joita ajetaan .NET frameworkin käytön mahdollistavalla ajoympäristöllä. C# käännetään IL-kielelle ja mukaan pakataan MonoVM-virtuaalikone. IL-koodia käännetään tarvittaessa ajon aikana binääriksi ja sitä ajetaan Android ajoympäristön rinnalla. Ohjelma kommunikoi natiivien luokkien kanssa JNI:n avulla. Android-sovelluksien kehittämistä varten tulee olla asennettuna Android SDK ja Java. Sovelluksia voi kehittää sekä Windows- että Mac -käyttöjärjestelmillä. /16/

2.5 Johtopäätös ja tekniikan valinta

Natiivit mobiiliapplikaatiot eliminoituivat vaihtoehdoista hyvin nopeasti. Esimerkiksi iOS:lle kehittäminen olisi vaatinut rahallisia investointeja, koska olisi pitänyt hankkia Intel-pohjainen Macintosh tietokone. Näitä ei tilaajalla tuolloin ollut saatavilla ja käyttötarkoitus olisi siinä tilanteessa ollut ainoastaan mobiiliapplikaation

demon kehittäminen. Tämän jälkeen demosovelluksen kehittäminen Androidille ei ollut enää järkevää, koska tällöin sovellus kattaisi vain yhden mobiilikäyttöjärjestelmän.

Apache Cordova ja Xamarin olivat siis hyviä vaihtoehtoja, koska niillä olisi päässyt käsiksi molempiin käyttöjärjestelmiin. Tilaajalla oli pääasiallisesti käytössä Visual Basic ohjelmointikielenään, joten ei tuntunut luontevalta käyttää Apache Cordovaa liiallisen eroavaisuuden vuoksi. Xamarin puolestaan käyttää C# -ohjelmointikieltä sovelluksissaan. Vaikka C#:n syntaksi on erilainen verrattuna Visual Basicin syntaksiin, se käyttää kuitenkin samoja kirjastoja ja pääosin on kuitenkin hyvin samankaltainen rakenteeltaan. Molemmat ohjelmointikielet hyödyntävät .NET -framework:iä. Työn tekohetkellä Xamarinin lisensointipolitiikka oli hieman sekava. Itse olisin ollut oikeutettu käyttämään Xamarinia veloituksetta, mutta tilaajan kohdalla asia ei ollut niin selkeä. Koin tarpeettoman hankalaksi selvittää lisensointiseikkoja tilaajayrityksen ylemmiltä tahoilta, joten päädyin hylkäämään myös Xamarinin vaihtoehdoista ja näin ollen mobiiliapplikaation mahdollisuus hylättiin.

Seuraava vaihe oli alkaa miettimään web-applikaation vaihtoehtoa. Se tuntui luonnolliselta seuraavalta askeleelta, koska web-palvelu kuitenkin toteutettaisiin hyödyntäen ASP.NET -teknologiaa. Järkevä vaihtoehto oli ASP.NET MVC 5, sillä se oli nykyaikaista teknologiaa ja antaisi jatkokehitysmahdollisuudet pitkälle eteenpäin. Lisäksi sen modulaarinen rakenne helpottaisi uusien ominaisuuksien lisäämistä jälkikäteen ja se toimisi hyvin web-palvelun rinnalla. Ennen kaikkea, se olisi toteutettavissa käyttämällä Visual Basic -ohjelmointikieltä. Vaikka MVC 5 -web -sovellusta ajetaankin laitteen internetselaimen kautta, se tarjoaisi silti responsiivisen käyttöliittymän mobiilikäyttäjällekin. Lisäksi tämä vaihtoehto antaa mahdollisuuden helpompaan ohjelman päivitettävyyteen, koska käyttäjän ei tarvitse huolehtia sovelluksen päivittämisestä. ASP.NET MVC 5 ja ASP.NET Web API 2 yhdistelmä web-applikaationa valikoitui lopulliseksi työn teknologiaksi.

3 TOTEUTUS

Lopullinen työ koostuu kahdesta isommasta osiosta, web-palvelu sekä web-käyttöliittymä.

3.1 ASP.NET Web API 2

ASP.NET Web API 2 on http-protokollaa käyttävien web-palveluiden kehittämiseen tarkoitettu framework. Web API vastaa sille lähettyihin pyyntöihin JSON tai XML -formaatissa. Web API hyödyntää MVC eli Model-View-Controller -mallia. Model hallinnoi sovelluksen toiminnallisuutta ja dataa ja antaa tietoja muille luokille. View tarjoaa nimensä mukaisesti näkymän sovellukselle eli View toimii käyttöliittymänä. Web API:n kohdalla View ei ole pakollinen luokka. Controller puolestaan tulkitsee käyttäjän toimia, kuten näppäimistön ja hiiren toimintaa sovelluksessa. Controller myös vastaa yleisesti sovelluksen toiminnallisuudesta ja välittää tietoa Model- ja View -luokille. MVC-malli helpottaa sovelluksen koodin testausta ja ylläpitoa. /17-18/

Web API:a voi isännöidä joko IIS:n sisällä tai käyttää OWINia. IIS, eli Internet Information Services on Microsoftin kehittämä web-palvelin Microsoft Windows -alustoille. OWIN puolestaan on rajapinta web-palvelimien ja web-applikaatioiden välille. Sen tarkoitus on antaa mahdollisuus käyttää muita frameworkejä IIS:n kanssa tai vaihtoehtoisesti ajaa ASP.NET -sovelluksia muissa ympäristöissä. OWIN myös mahdollistaa ASP.NET -sovelluksien rakentamisen modulaarisemmiksi, paremmin skaalautuviksi ja helpommin portattavaksi muille alustoille. /19-20/

Kun palvelin saa HTTP pyynnön, palvelin muuntaa kyseisen pyynnön HttpRequestMessage objektiksi. Luotu objekti tarjoaa vahvasti tyypitetyn rajapinnan HTTP-pyynnölle. Tämä antaa mahdollisuuden päästä helpommin käsiksi pyynnön eri ominaisuuksiin. Lisäksi kaikki pyynnöt ovat vahvan tyypityksen mukaan rakenteeltaan samankaltaisia, joten eri pyyntöjen käsittely koodissa on selkeämpää. HTTP-käsittelijät prosessoivat HTTP-pyyntöjä niiden saapuessa ja HTTP-

vastauksia niiden lähtiessä. Nämä käsittelijät voivat suoraan antaa vastauksen pyyntöön tai ohjata pyynnön eteenpäin, esimerkiksi Controller-luokalle. /21-22/

Controller-luokassa määritetään logiikka HTTP-pyyntöjen käsittelyyn. Controller myös noutaa ja päivittää Model-luokan dataa ja sen perusteella muotoilee vastauksen pyyntöön. Model-luokka antaa nimensä mukaisesti mallin datalle eli mitä ominaisuuksia datalla on. Jotta Web API tietäisi mihin toimintoon HTTP-pyyntö tulee ohjata, se hyödyntää Routing Table:a. Jokainen reitti routing table:ssa sisältää mallipohjan URI:n tyylille ja osoituksen siitä mihin URI:a vastaava pyyntö ohjataan. /21-22/

3.2 Web-palvelu

Web-palvelun toteutus alkoi Model-luokkien muodostamisesta. Näissä Model-luokissa määritellään eri objektien ominaisuuksia, esimerkiksi artikkeli -objektille artikkelinimi, tyyppi ja artikkelikoodi. Nämä ominaisuudet vastaavat NovaSDK:n vaatimia ominaisuuksia eri objekteille varastokirjanpidon kontekstissa. Lisäksi tämä helpottaa pyyntöihin vastaamista, kun data on yhtenäistä.

Controller-luokat jaettiin myös omiin osa-alueisiinsa koodin selkeyttämiseksi. Esimerkiksi tuotteen tietojen manipulointia käsittelevä luokka omakseen ja tilausta käsittelevä luokka omakseen. Otetaan esimerkiksi tuotetta käsittelevän luokan GetProduct() -funktio, joka palauttaa vastauksena halutun tuotteen tiedot. Funktio on tyyppiä HTTP GET -mikä tarkoittaa, että pyynnön tarkoitus ilmoitetaan jo pyynnön URI:ssa. Tässä tapauksessa URI:ssa tulee myös ilmoittaa pyyntömuuttuja eli tuotteen nimi tai EAN-numero. Pyyntömuuttujan perusteella ohjelma ottaa yhteyden tietokantaan ja palauttaa tiedot vastausviestin rungossa. Vaihtoehtoisesti funktiota voisi käyttää HTTP POST -tyypin kanssa, jolloin pyyntömuuttuja olisi pyynnön rungossa. Tässä esimerkissä se ei kuitenkaan selkeyden vuoksi ollut tarpeellista. Jos pyyntömuuttuja olisi pidempi merkkijono tai jopa kokonainen tiedosto, silloin HTTP POST olisi järkevä valinta. /23-24/

Taulukko 1. Otanta web-palvelun julkisista metodeista.

Pyynnön tyyppi	URI	Selitys
GET	api/Warehouse/GetProduct/{aName}	Palauttaa haetun tuotteen tiedot.
POST	api/SalesOrder/SaveOrder	Päivittää tilauksen tiedot ja palauttaa kuittauksen.
GET	api/SalesOrder/GetOrder/{oNum}	Palauttaa haetun tilauksen tiedot.

Taulukossa 1 on havainnollistettu web-palvelun kolmea funktiota esimerkkinä. SaveOrder()-funktio ottaa vastaan tilauksen tiedot pyynnön rungosta. Nämä tiedot tulee olla JSON-formaatissa. Sisällön typpi on määritelty pyynnön header-osiossa. Mikäli sisältö on väärässä formaatissa, web-palvelu voi hylätä pyynnön header-osion tietojen perusteella. Funktio päivittää tilauksen tiedot tietokantaan ja palauttaa vastauksen joko onnistuneesta tai epäonnistuneesta päivityksestä. GetOrder() -funktio on toiminnaltaan samankaltainen aiemmin mainitun GetProduct() -funktion kanssa. Pyynnön URI:ssa annetaan tilauksen numero, jonka perusteella funktio hakee tietokannasta oikean tilauksen. Tämän jälkeen funktio palauttaa tilauksen tiedot vastauksen rungossa. Mikäli kyseisellä numerolla ei ole tilausta, vastauksen runko on tyhjä. Näin ollen on asiakassovelluksen vastuulla tutkia, mikäli vastaus on tyhjä, ja ilmoittaa halutulla tavalla hakutuloksien puutteesta käyttäjälle. Web-palvelun toiminta testattiin ensimmäisessä demovaiheessa, jossa olin luonut työpöytäsovelluksen kutsumaan web-palvelua. Työpöytäsovelluksella pystyi myös tarkastelemaan pyyntö- ja vastausviestejä kokonaisuudessaan.

3.3 ASP.NET MVC 5

MVC 5 on rakenteeltaan hyvin samankaltainen Web API 2:n kanssa. Molemmat hyödyntävät MVC, Model-View-Controller, arkkitehtuuria. Tosin Web API ei varsinaisesti toteuta näkymiä (View), mutta MVC 5:ssa ne sen sijaan ovat erittäin olennainen osa sovellusta. View eli näkymä on käytännössä MVC 5 -web-sovelluksen käyttöliittymä. Virallisesti näkymät ovat mallipohjia, joiden perusteella sovellus dynaamisesti generoi HTML vastauksia. Nämä HTML vastaukset näkyvät

esimerkiksi käyttäjän internetselaimessa web-sivuina. MVC 5 sallii HTML -kielen ohella C# tai tässä tapauksessa Visual Basic -ohjelmointikielien käytön näkymissä. Tämän vuoksi näkymät on generoitava HTML-dokumenteiksi. /21, 25, 28/

MVC 5 tukee Bootstrap -kirjastoa. MVC 5:n mukana tulevassa valmiissa projektin mallipohjassa on jo Bootrap käytössä. Mallipohjaa on helppo muokata omiin tarkoituksiin sopivaksi. Bootstrap -kirjasto tarjoaa useita eri komponentteja, joiden avulla web-sovelluksen käyttöliittymästä saadaan tehtyä responsiivinen. Toisin sanoen, Bootstrap saa web-sovelluksen näyttämään hyvältä ja näyttöön sopivalta riippumatta siitä millä laitteella web-sovellusta käytetään. Lisäksi Bootstrap on ilmainen käyttää ja toimii kaikkien modernien internetselainten kanssa. /25, 29, 30/

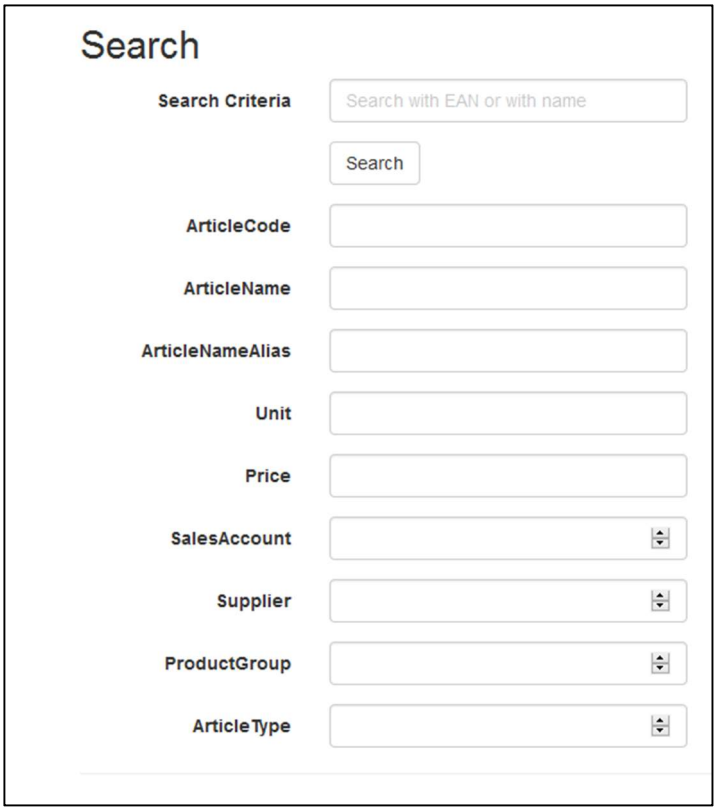
Näkymät ja Controller-luokat toimivat tiiviisti yhteistyössä. Esimerkiksi kun käyttäjä haluaa siirtyä MVC 5 web-sovelluksen sivulle, selain lähettää pyynnön sovelluksen aloitussivusta palvelimelle. Palvelin ohjaa pyynnön routing tablessa ilmoitetulle aloitussivun, tässä aloitusnäkymän, Controller -luokalle. Controller-luokasta sovellus etsii Index() -metodin, joka palauttaa Index -näkymän. Tämä Index -näkymä, tai aloitusnäkymä avataan käyttäjän internetselaimessa. Tämä toimintamalli toistuu sovelluksen kaikkien näkymien kohdalla. Vastaavasti esimerkiksi tavallisessa yksinkertaisessa lomakkeessa, jonka tarkoitus on tallentaa annetut henkilötiedot tietokantaan, ”lähetä”-painike välittää lomakkeen tiedot Controller -luokan funktiolle, joka puolestaan tallentaa tiedot oikeassa muodossa tietokantaan. Eli samaan tapaan kuin Web API:ssa, Controller -luokat sisältävät pääosan sovelluksen logiikasta. /22, 25, 27, 28/

Model -luokat noudattavat Web API:sta tuttua kaavaa eli ne ovat mallipohjia datalle. Näiden mallipohjien perusteella muodostetaan, esimerkiksi objekteja datasta, joita voidaan manipuloida Controller -luokissa tai esittää näkymissä. Näkymille voidaan määrittää suoraan koodissa Model -luokka, jonka rakennetta näkymä noudattaa. Controller -luokat toimivat välikätenä näkymien ja Model -luokkien välillä. /26, 28/

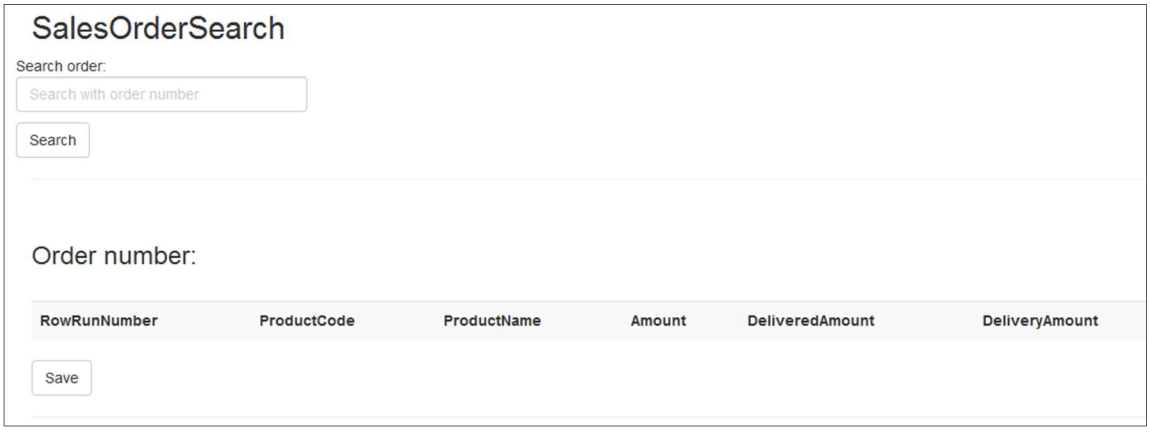
3.4 Web-sovellus

Web-sovelluksen muodostaminen alkoi ehkä hieman poikkeuksellisesti Controller -luokkien suunnittelusta. Koin helpommaksi ajatella kokonaisuutta, kun sovelluksen logiikka oli hahmoteltuna. Koska ASP.NET sallii MVC- ja Web API -projektien yhdistämisen samaan projektiin, koin tässä tapauksessa järkevimmäksi kutsua web-palvelun metodeita suoraan web-sovelluksen Controller -luokista. Web-palvelun toiminta oli kuitenkin jo testattu ensimmäisessä demovaiheessa, joten toisen demovaiheen kannalta tämä tuntui fiksuimmalta ratkaisulta. Web-sovelluksen Controller -luokkien metodit on kuitenkin luotu niin, että on helppo vaihtaa käyttämään HTTP -metodeita. Model -luokat suunniteltiin vastaamaan web-palvelun Model -luokkia, jolloin datan käsittely helpottuu myös web-sovelluksessa.

Näkymistä luotiin mahdollisimman yksinkertaisia ja asettelultaan sellaisia, että responsiivisuutta on hyvä esitellä. Esimerkiksi tuotteen haku -näkymä (**Kuva 1.**) on pystysuunnassa toteutettu, joten se sopii hyvin älypuhelimen näytölle. Sen sijaan tilauksen haku -näkymä (**Kuva 2.**) on toteutettu enemmän vaakatasoon, jotta se sopisi parhaiten työpöytä- tai tablet-tietokoneen ruudulle.



Kuva 1. Kuvakaappaus web-sovelluksen tuotehaku -näkymästä.



Kuva 2. Kuvakaappaus web-sovelluksen tilaushaku -näkymästä.

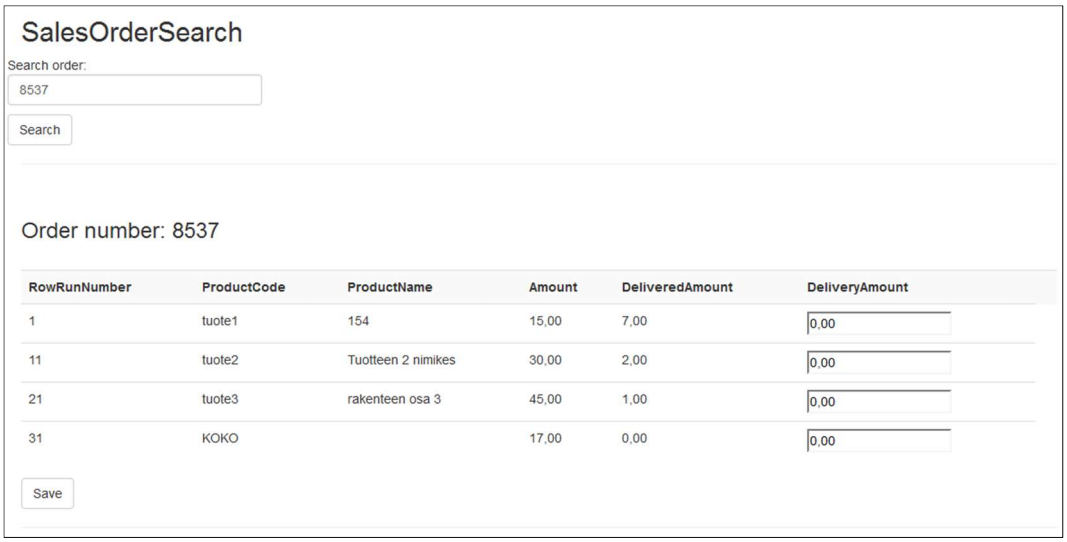
Tuotehaku -lomakkeen (**Kuva 1.**) hakukenttään voi syöttää tuotteen nimen tai EAN-koodin. Painamalla ”Search” -painiketta näkymä välittää hakukentän sisällön ArticleController -luokan Search() -funktiolle. Funktiolta hakukriteeri välitetään web-palvelun GetProduct() -metodille ja siellä selvitetään tehtiinkö haku käyttämällä tuotenimeä vai EAN -koodia. Riippuen siitä kumpaa hakuun käytettiin, hakukriteeri välitetään oikealle salatulle funktiolle, joka suorittaa tietokantahaun. Tämä salattu funktio palauttaa GetProduct() -funktiolle haun tuloksen. Tuloksen arvot sijoitetaan vaaditun Model -luokan objektiksi ja luotu objekti välitetään takaisin ArticleControllerin Search() -funktiolle. Siellä objekti sijoitetaan näkymälle määritettyyn Model -luokan objektiin ja lopuksi funktio palauttaa näkymän päivitetyllä Model -luokan objektin arvoilla. Tämän olisi voinut toteuttaa käyttämällä samaa Model -luokkaa sekä web-sovelluksen että web-palvelun puolella, mutta tässä tapauksessa järkevämpi pitää ne erillään toisistaan jatkokehityksen helpottamiseksi.

Tilaushaku -näkymän (**Kuva 2.**) hakutoiminto toimii tilauksen numeron perusteella. Kun käyttäjä on syöttänyt numeron hakukenttään ja painaa ”Search” -painiketta, näkymä välittää tilausnumeron SalesOrderClientController -luokan GetOrder() -funktiolle. GetOrder() -funktio kutsuu web-palvelun saman nimistä metodia, joka suorittaa tietokantahaun tilausnumeron perusteella. Mikäli haku on onnistunut, web-palvelun GetOrder() -metodi palauttaa tilauksen tuoterivit tietoineen SalesOrderClientController -luokalle. Jälleen jos hakutuloksia ei ole, palautus sisältää tyhjän objektin. Olettaen, että hakutuloksia löytyy, Controller -luokan GetOrder() -funktio palauttaa näkymän uusilla Model -luokan arvoilla. Tässä tapauksessa hakutuloksilla (**Kuva 3.**)

Käyttötapausesimerkkinä (**Kuva 3.**) voidaan ajatella, että käyttäjä on saanut tehtäväkseen kerätä ja toimittaa tilauksen tuotteet oikeine määrineen. Hän jatkaa edellisen käyttäjän toimeksiantoa kerätä tilauksen tuotteet, joten osa tilauksen tuotteista on jo kerättynä. Käyttäjä näkee web-sovelluksesta ”DeliveredAmount” -sarakkeesta, kuinka paljon mitäkin tilauksen tuotetta on jo kerättynä. Näin hänen tarvitsee kerätä vain puuttuvat määrät, mikäli tuotteita on tarpeeksi varastossa. Käyttäjä täyttää keräämänsä määrän ”DeliveryAmount” -sarakkeeseen oikean tuotteen

kohdalle. Painettaessa ”Save” -painiketta, täytetyt tiedot välitetään SalesOrderClientController -luokan SaveOrder() -funktiolle ja sieltä taas web-palvelulle.

Web-palvelussa SaveOrder() -metodi suorittaa annettujen tietojen päivittämisen tietokantaan ja palauttaa web-sovellukselle tiedon tietojen tallennuksen onnistumisesta. Controller -luokan SaveOrder() -funktio palauttaa näkymälle tiedon tallennuksen onnistumisesta ja päivittää Model- luokan objektin arvot näkymälle. Näin ollen juuri tallennetut ”DeliveryAmount” -sarakkeen arvot on lisätty ”DeliveredAmount” -sarakkeen arvoihin.



Kuva 3. Kuvakaappaus tilaushaku -näkymästä, kun haku on suoritettu.

4 LOPPUSANAT

Pilvipohjaisten sovellusten määrä lisääntyy jatkuvasti. Syynä tähän lienevät trendi ja uudet liiketoimintamallit. Lisäksi kehittäjien on helpompaa ylläpitää sovelluksia, koska niitä ajetaan heidän omilla palvelimillaan. Tämän vuoksi web-applikaatio on hyvä valinta nykypäivään, sekä tulevaisuutta ajatellen. Web-applikaatio on helposti laajennettavissa tarpeen mukaan, mikä tekee ajan mukana pysymisestä huomattavasti kätevämpää.

Työtä tehdessä kului paljon aikaa vaihtoehtoisten teknologioiden tutkimiseen ja niitä käyttäen testisovelluksien tekemiseen. Oli hyvä kuitenkin opetella uusia osa-alueita ja tarkemmin selvittää, mikä teknologia on tähän työhön sopivin. Lisäksi MVC 5 ja Web API 2 olivat aihepiirinä itselleni lähes täysin uusi asia, joten niiden opettelemisessa meni myös huomattavan paljon aikaa. Monissa tilanteissa dokumentaatio ja menetelmät oli tehty C# -ohjelmointikielelle ja tämän vuoksi oli välillä vaikeaa löytää tietoa. Joitakin ongelmia joutui ratkaisemaan vain kokeilemisen ja epäonnistumisen kautta. Välillä sai onnistumisen tunteen, kun onnistui itse ratkaisemaan jonkin ongelman, johon ei löytynyt tietoa. Toisinaan ongelmien ratkaisu onnistui ensi yrittämällä. Tein työn etätyöpöytäyhteyden kautta ja toisinaan verkkoyhteys katkeili, joten siitä koitui ylimääräistä harmia.

Jos olisi ollut enemmän aikaa, olisin halunnut tehdä työstä huomattavasti kattavamman. Käyttöliittymästä olisi tullut tyylikkäämpi ja ominaisuuksia olisi ollut paljon enemmän. Olen kuitenkin yleisesti tyytyväinen lopputulokseen ja sain tilaisuuden oppia uutta suuren ohjelmistotalon alaisuudessa. Ennen työn tekemistä en ollut vielä osannut ajatella mitä haluaisin tehdä opintojeni päätyttyä. Nyt kuitenkin uskon, että haluaisin jatkaa mobiili -tai pilviohjelmistojen kehittäjänä.

LÄHTEET

/1/ What is ERP?
Viitattu 9.1.2018.
https://www.sap.com/products/what-is-erp.html

/2/ Tietoa Vismasta.
Viitattu 9.1.2018.
https://www.visma.fi/tietoa-vismasta/

/3/ Visma Software Oy.
Viitattu 9.1.2018.
https://www.visma.fi/ota-yhteytta/visma-yritykset/visma-software-oy/

/4/ Smartphone OS.
Viitattu 9.1.2018.
https://www.idc.com/promo/smartphone-market-share/os

/5/ Cross-platform Vs. Native Mobile App Development: Choosing the Right Dev Tools for Your Project.
Viitattu 9.1.2018.
https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-appdevelopment-
choosing-the-right-dev-tools-for-your-app-project-47d0abafee81

/6/ Android, the world's most popular mobile platform.
Viitattu 11.1.2018.
https://developer.android.com/about/android.html

/7/ C# and Java: Comparing Programming Languages.
Viitattu 11.1.2018.
https://msdn.microsoft.com/en-us/library/ms836794.aspx

/8/ Android Application Fundamentals.
Viitattu 27.2.2018.
https://developer.android.com/guide/components/fundamentals.html

/9/ Android Guide to App Architecture.
Viitattu 6.3.2018.
https://developer.android.com/topic/libraries/architecture/guide.html

/10/ About iOS App Architecture.
Viitattu 26.3.2018.
https://developer.apple.com/library/content/documentation/iPhone/Conceptual/
iPhoneOSProgrammingGuide/Introduction/Introduction.html#//apple_
ref/doc/uid/TP40007072

/11/ iOS Expected App Behaviors.
Viitattu 27.3.2018.
https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ExpectedAppBehaviors/ExpectedAppBehaviors.html#//apple_ref/doc/uid/TP40007072-CH3-SW2

/12/ iOS Strategies for Handling App State Transitions.
Viitattu 28.3.2018.
https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/StrategiesforHandlingAppStateTransitions/StrategiesforHandlingAppStateTransitions.html#//apple_ref/doc/uid/TP40007072-CH8-SW1

/13/ iOS The App Life Cycle.
Viitattu 28.3.2018.
https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html#//apple_ref/doc/uid/TP40007072-CH2-SW1

/14/ Apache Cordova Introduction Overview.
Viitattu 23.2.2018.
https://cordova.apache.org/docs/en/latest/guide/overview/index.html

/15/ Cordova vs. Native apps.
Viitattu 23.2.2018.
http://mubaloo.com/cordova-vs-native-apps/

/16/ Understanding the Xamarin Mobile Platform.
Viitattu 27.2.2018.
https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/

/17/ Model-View-Controller.
Viitattu 29.3.2018.
https://msdn.microsoft.com/en-us/library/ff649643.aspx

/18/ Working of Asp.Net Web API.
Viitattu 29.3.2018.
https://www.c-sharpcorner.com/UploadFile/2b481f/working-of-Asp-Net-webapi/

/19/ What is OWIN?
Viitattu 1.4.2018.
http://www.codedigest.com/posts/1/what-is-owin-a-beginners-guide

/20/ Internet Information Services (IIS).
Viitattu 1.4.2018.
http://searchwindowsserver.techtarget.com/definition/IIS

/21/ Get Started With ASP.NET Web API 2.
Viitattu 3.4.2018.
https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-withaspnet-web-api/tutorial-your-first-web-api

/22/ MVC 5: Adding a Controller.
Viitattu 3.4.2018.
https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-controller

/23/ HTTP Post.
Viitattu 5.4.2018.
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

/24/ HTTP Get.
Viitattu 5.4.2018.
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET

/25/ ASP.NET MVC 5.
Viitattu 5.4.2018.
https://docs.microsoft.com/en-us/aspnet/mvc/mvc5

/26/ MVC 5: Adding a Model.
Viitattu 5.4.2018.
https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-model

/27/ MVC 5: Adding a View.
Viitattu 5.4.2018.
https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-view

/28/ What Is MVC and Why Do We Use MVC?
Viitattu 5.4.2018.
https://www.c-sharpcorner.com/article/what-is-mvc-and-why-we-use-mvc/

/29/ Bootstrap Get Started.
Viitattu 5.4.2018.
https://www.w3schools.com/bootstrap/bootstrap_get_started.asp

/30/ Bootstrap Introduction.
Viitattu 5.4.2018.
http://getbootstrap.com/docs/4.0/getting-started/introduction