Arvind Sedha

CSI Lawyer® Windows Phone® Client Development and Data Synchronization

Helsinki Metropolia University of Applied Sciences
Master of Engineering
Information Technology
Thesis
15 May 2012

Metropolia

Abstract

Author(s) Title Number of Pages Date	Arvind Sedha CSI Lawyer® Windows Phone® Client Development and Data Synchronization 87 pages + 6 appendices 15 May 2012
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Mobile Programming
Instructor(s)	Jukka Juslin, Senior Lecturer Seppo Kabongo, Development Manager

CSI Helsinki Oy has ERP software called CSI Lawyer® which is intended for law and consultant companies. CSI Helsinki Oy has great demand to develop the mobile clients for CSI Lawyer® for the most common mobile platforms available at present. The purpose of this project was to develop the Windows Phone® client application called CSI Mobile® for CSI Lawyer® including backend data synchronization business logic.

CSI Mobile® allows CSI Lawyer® users to manage their transactions and view transaction related information such as customers, assignments, and transaction templates from their Windows Phone® device. CSI Mobile® was designed to work with any kind of device using Windows Phone® 7.5 or above.

CSI Mobile® was developed using Silverlight, C# programming language and .Net Framework. The backend data synchronization between CSI Mobile® and CSI Lawyer® has been implemented using Microsoft Sync Framework. The data was synchronized between CSI Mobile® and CSI Lawyer® using an intermediate database located on SQL Azure Platform. The data between intermediate database and CSI Mobile® is synchronized using an OData web service.

This project was finished with all the objectives completed on time designated for the project. The result of the project was a CSI Mobile® application with all necessary components required for synchronizing data between CSI Lawyer® and CSI Mobile®.

Keywords	CSI Helsinki Oy, Windows Phone®, Mobile Application Development, Microsoft Sync Framework, XML, Silverlight, SSL, OData, CSI Mobile®, CSI Lawyer®

Contents

Figures

Tables

Listings

Abbrevations

.Net or .Net Framework	Software framework developed by Microsoft
API	Application Programming Interface
C#	Software development language for .Net Framework
CSI Lawyer®	Desktop based ERP application developed by CSI Helsinki Oy
CSI Mobile®	Mobile client application for CSI Lawyer®
ERP	Enterprise Resource Planning
HTML5	Hyper Text Markup Language Version 5
HTTP	Hypertext Transfer Protocol to transfer data over web.
HTTPS	Hypertext Transfer Protocol Secure is a secure version of HTTP which uses SSL
JSON	JavaScript Object Notation is lightweight text-based open standard for exchanging structure information
Microsoft Sync Framework	Open source data synchronization platform developed by Microsoft
OCA	Occasionally Connected Application
OData	Open Data Protocol for querying and updating data
SOAP	Simple Object Access Protocol used for exchanging structure information
SQL	Structured Query Language for managing relation data
SSL	Secure Sockets Layer provides communication security using certificates
UI	User Interface
WP	Windows Phone®
WSDL	Web Service Description Language
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

1 Introduction

CSI Helsinki Oy has ERP software called CSI Lawyer® which is intended for law and consulting firms. CSI Lawyer® is a complete ERP solution which allows companies to keep track of their customers, assignments, worked hours, transactions and invoices.

Law and consulting firms' employees travel often to different customer sites and do their work mainly outside their offices. Due to the number of smartphones used by law and consulting firms' employees, there is a great need for a mobile application which allows them to access customer information and log transactions using mobile devices.

In October 2010, Microsoft's latest mobile operating system, Windows Phone® was announced at Mobile World Congress. Microsoft has the longest computing heritage and a suite of services that ranges from business to entertainment, browsing and searching. The Windows Phone® hooks all these together in a way that puts the market on its head. Instead of emulating what others have done, Microsoft has taken a radical approach to the phone. [1]

Windows Phone® features a clean and minimalistic interface known as Metro UI. The operating system features a start screen with "Live Tiles" to display notifications and information at a glance, "hubs" to show content from social networks and device's local data stores. Released in fall 2011, a major update known as "Mango" has brought many more features such as compact database support, multitasking, Twitter integration and a mobile version of Internet Explorer 9 with full support for HTML5.

Windows Phone® started its journey in the early stages with few mobile phone makers such as LG, Samsung, Dell, Asus and HTC. [2] In February 2011, Nokia and Microsoft officially announced a strategic alliance that made Windows Phone® Nokia's primary smartphone platform. Nokia's first Windows Phone® was released in Q4 2011 equipped with the latest major release of Windows Phone® version 7.5 known as code name Mango. [3] Acer, Fujitsu and ZTE are new confirmed mobile phone manufactures committed to ship the Windows Phone® smartphone running Mango release during fall 2011. [4]

CSI Helsinki Oy is a Microsoft Silver Partner company with long term experience in selling and developing business applications using Microsoft technologies. The launch of the Windows mobile phone platform has created a great opportunity for the company to offer the Windows Phone® client application an extension to their existing line of business ERP applications. The expected growth of the Windows Phone® in different business sectors due to out-of-box integration with many existing Microsoft technologies and services such as Microsoft Exchange, Azure and SharePoint, CSI Helsinki Oy has decided to develop the Windows Phone® client application for the CSI Lawyer® product. The CSI Lawyer® Windows Phone® application will be called by the name CSI Mobile®.

The project objectives can be categorized into two major parts. The first part is to study and develop a backend solution to synchronize the data between the CSI Lawyer® database and the Windows Phone® device. The backend solution should make use of any existing technologies and frameworks available in the market at the time of writing this project and should be suitable for a backend synchronization scenario.

The second objective is to study the Windows Phone® and other Microsoft technologies which can be utilized while developing the prototype the Windows Phone® client for CSI Lawyer®. The final client application at a minimum should allow the CSI Lawyer® users to create transaction securely using the Windows Phone®. The study also includes researching possibilities to implement some or all of these features which might be implemented during the next stage of client application development.

- Manage customer information
- Manage working hours
- Manage client transactions
- Manage critical task calendar/reminders
- Easily navigate between different client information
- Send client email, calling client phone numbers, view customer addresses on a map
- Minimal Windows Phone® client application setup procedure.

The theoretical part includes comparing different Microsoft technologies which can be utilized to implement the CSI Lawyer® WP client application without the need of any rapid change in the existing CSI Lawyer® application architecture including data and business layer. Testing, error handling and logging is out of scope during this project.

1.1 Project background

CSI Helsinki Oy has been designing and implementing innovative, easy to use software solutions for streamlining day-to-day tasks of professional service companies since 1985. Law firms have been the major sector of business during past 25 years. CSI Lawyer® is one of the major products developed by CSI Helsinki Oy in collaboration with leading Finnish and foreign law firms who have strived to streamline their operations with user friendly software. CSI Lawyer® has been completely developed using Microsoft technologies with rich Outlook like user interface which is easy to learn and use as illustrated by figure 1. [5]

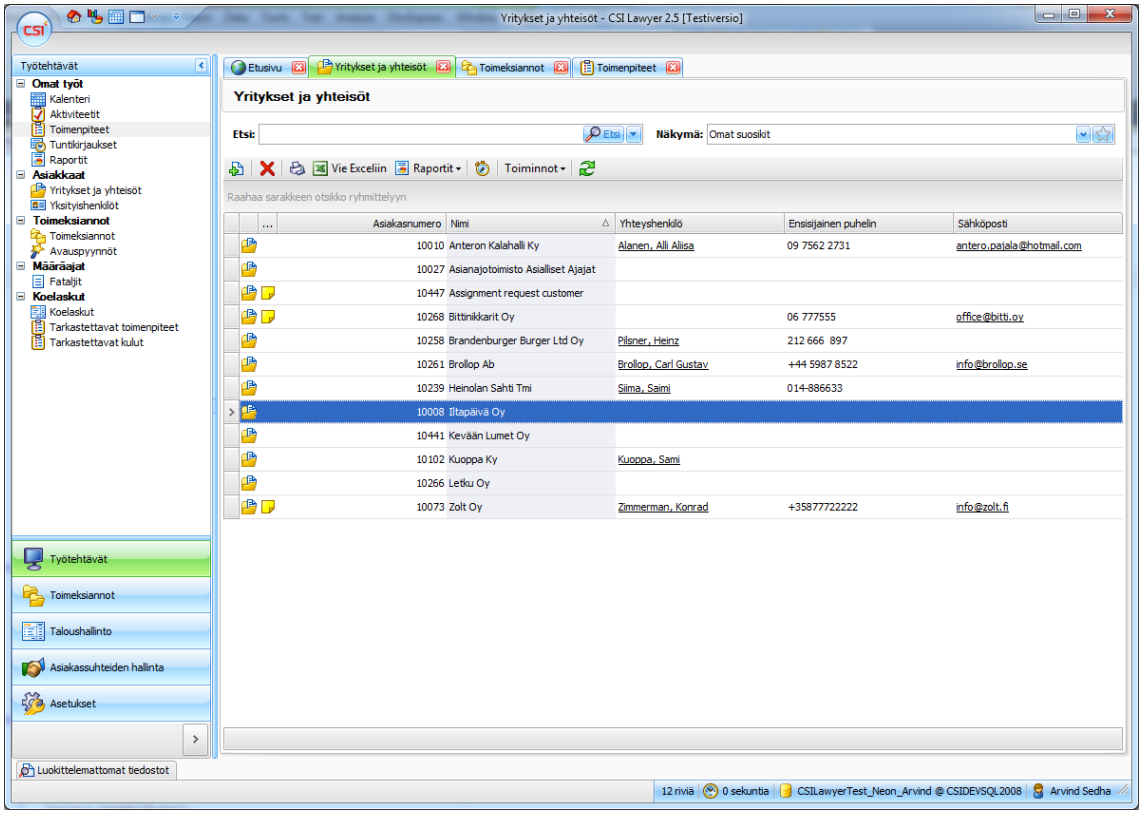


Figure 1: CSI Lawyer® User Interface

CSI Lawyer® is a complete practice and billing management solution which integrates all the core functions of the modern software required by law firms. The core functionality of the software includes: assignment management, registration of transactions and time, customer relationship management, resource management, document management, invoicing and finances and financial management. Versatile and flexible reporting, customer based customizations, continued development and integration with other third party systems such as M-Files, SharePoint, and Microsoft Dynamic CRM makes CSI Lawyer® a market player in the law sector in Finland. "Today, over 60% of the top 30 Finnish law firms use CSI Lawyer®, making it a clearly the market leader in Finland" [5]

Mobile phone has been an important part of human daily life during past 20 years. As the technology has been growing, mobile phones have changed to become multipurpose devices known as smartphones. Today's smartphones are widely used for a number of applications in business life due to wide a variety of business applications available for smartphones and repaid growth of mobile Internet connections worldwide. According to the latest research made by Gartner, smartphone sales have increased by 42%, which indicates that more and more people are interested in buying smartphones. [6]

CSI Helsinki Oy has developed in the past few years a Symbian ^3 client application as an extension to the CSI Lawyer® desktop application. It allows CSI Lawyer® users to manage their customers, transaction work time, and important appointments straight from the phone. Due to the resent changes in the mobile world, Nokia the biggest Symbian mobile phone manufacturer announced in February 2011 that they are going to take Microsoft's latest mobile operating system Windows Phone® as the primary operating system for their phones. According to International Data Corporation's (IDC) smartphone market share report, Symbian smartphone's share is going to shrink from 20% to 0.1% whereas Windows Phone® market share is expected to increase rapidly from 3.8% to over 20% by 2015. [6;7]

Due to the expected growth of Windows Phone® in the coming future, CSI Helsinki Oy has decided to make another mobile phone client for CSI Lawyer® users which should

allow them to create transactions, link and view the necessary information required to create transactions straight from their Windows Phone® device.

1.2 Initial requirements

The Mobile application world is different from the normal desktop application world due to the limitations in the types of resources that a mobile phone can offer. Due to the limited resources on the mobile phone, every mobile application developer needs to take care while trying to meet the functional requirements. The following are some of the limitations of mobile phone applications. [8,20]

- Limited amount of memory
- Limited processing power
- Small screen sizes, limited navigation space
- Different screen resolutions
- Limited network connectivity in some area/situations
- Battery consumption
- Simplicity of user interface
- Limited bandwidth
- Data security issues
- Touch, Keyboard or Hybrid keyboard interface
- Limitations of available sensors. [8,20]

Where mobile devices make it easier for users to communicate and exchange information on the move, many users still get frustrated due to a small screen size, a limited amount of information on the screen, or slow processing speed. These are some of the other challenges what developers need to take care of during any application design by keeping in mind that "An application without users is worthless".

The final version of the CSI Mobile® has been developed by keeping these general mobile application restrictions and user frustrations in mind while achieving the architectural and functional requirements.

1.2.1 Architecture-level requirements

CSI Lawyer® is a Windows client application which has been developed using .Net Framework and runs only on Windows computers. CSI Lawyer® communicates directly with the database and requires users to have direct access to the database. All the business logic and full processing of data is handled on the client computer. However in the past few years CSI Helsinki Oy has developed WCF service which runs on the server and is able to handle many common business operations handled by CSI Lawyer®. WCF service works as a bridge between CSI Lawyer® client applications and a physical database. Below we will consider three different architectural scenarios for CSI Lawyer®.

The first scenario illustrated in by figure 2 has both CSI Lawyer® Windows application and the CSI Lawyer® database is stored on the same computer. Only one user is using CSI Lawyer® all the time. No network or Internet connection is required because all the components required for CSI Lawyer® (database and client application) run on same computer.

Figure 2: One User, CSI Lawyer® application and database on the same computer

The second scenario illustrated in by figure 3 has CSI Lawyer® Windows application is installed on more than one computer and the CSI Lawyer® database is running on a separate computer (maybe on server) where everybody can access it any time over the local network or VPN network.

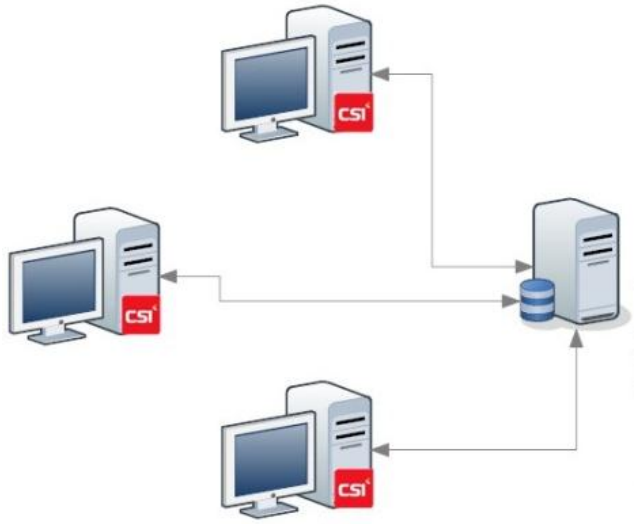


Figure 3: Many Users, CSI Lawyer® database on different computer

The second scenario illustrated in by figure 4 has CSI Lawyer® installed on one or more computers and the database is stored somewhere in a cloud which can be accessible any time over the Internet by all the CSI Lawyer® users.

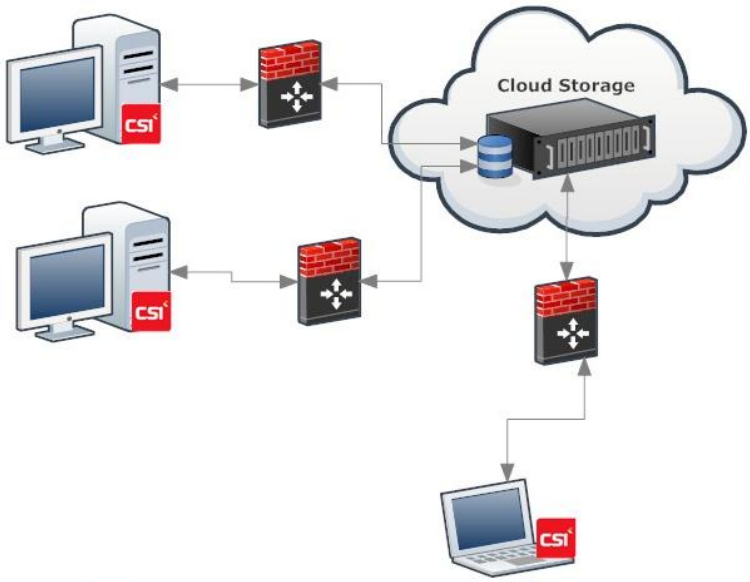


Figure 4: Many users, CSI Lawyer® database on cloud

In all these three scenarios, users are required to have direct access to the database. One part of the project is to find the right architecture solution which minimizes the configuration, reliability, security and connectivity problems so that CSI Lawyer® customers have to put minimum expenses for installation and setup of the backend

solution to put the mobile client application up and running for their employees. The backend architecture solution must meet following requirements.

- Existing Frameworks and technologies for data synchronization
- Fits to all scenarios mentioned above regardless of database location and number of users.
- Only data required by mobile client application should be exposed over Internet
- CSI Mobile® application must require only user credentials to work
- CSI Mobile® application should work in offline mode when the Internet connection is not available
- CSI Mobile® application meant to be thin client which implements minimum business logic
- Utilize CSI business service while saving and retrieving data from database.

Based on the mobile client application's backend and frontend requirements, the second objective of the project is to find the solution to how the data is getting synchronized between CSI Lawyer® and Windows Phone® database. The figure 5 illustrates the missing architecture solution to synchronize the data between CSI Lawyer® and CSI Mobile®.

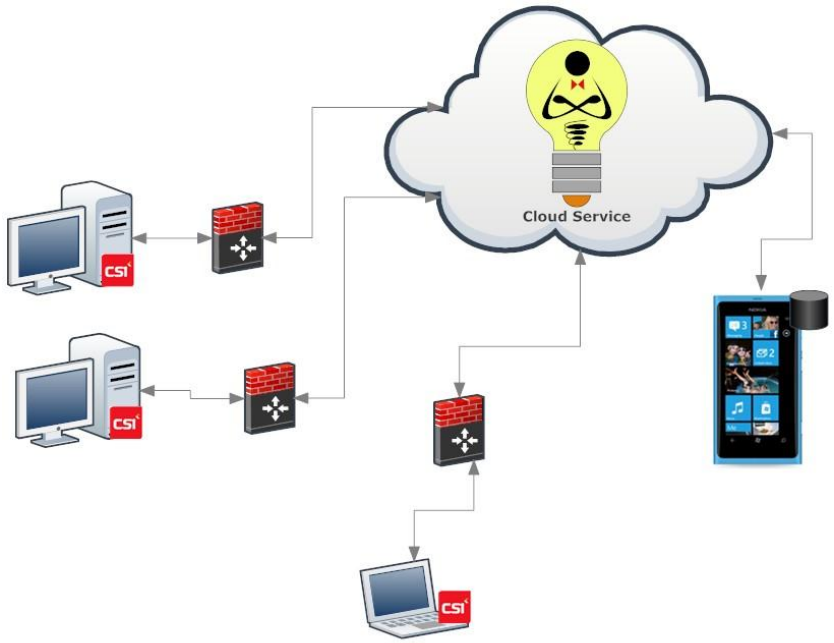


Figure 5: Missing architecture solution

1.2.2 Functional requirements

The CSI Mobile® application is intended for small to large organizations where the backend architecture can vary from a single computer to a cloud. The Windows Phone® client application should be easy to install and set up. It should only require an end user to have a Windows Phone® with an Internet connection and user credentials configured on CSI Lawyer®. Windows Phone® client application for CSI Lawyer® must meet following minimum functional requirements.

- Secure authentication with backend system
- Only favorite assignments and related data for specific user must be must be accessible on Windows Phone®.
- Allows user to create transaction with following information; assignment, customer subject, date, worked hours, billable hours Transaction type
- List and view transactions created by user
- Synchronizes required linked information for creating transaction; contacts and company Information, assignments, transaction types
- Minimum setup procedure (possibly username and password only)
- Client application should works temporary without Internet connection as well
- Settings page which allows user to save; user credentials, enable/disable backend data synchronization, switch application language
- Localization support for multiple languages
- Extendable, flexible, easy to use, responsive sleek design with live tiles.

The Mobile client application must utilize stable, reliable and appropriate technologies and frameworks to synchronize the data between the Windows Phone® and CSI Lawyer® database.

1.2.3 Technical requirements

The following technical requirements must be taken in consideration while finding and implementing the backend and frontend solution for CSI Lawyer® Windows Phone® mobile client application.

- Client application should support Windows Phone® 7.5 or above.
- Solution should be secure, flexible and scalable
- Backend solution must be developed using .Net and other Microsoft technologies
- Installation and usage of CSI Lawyer® Windows Phone® client application should not require organizations to configure their firewalls or open ports to make it work.

1.3 Structure of the study

The study consists of 7 chapters. Chapter 1 focuses on the introduction and the objectives of the project. It includes understanding and analyzing the business and architectural requirements for developing the CSI Lawyer® Windows Phone® application.

Chapter 2 focuses on the introduction of Windows Phone® development platform, tools and other connection technologies used for developing and Windows Phone® applications.

Chapter 3 focuses on the research of various communication protocols, data synchronization technologies and frameworks which can be used to develop the expected CSI Lawyer® Windows Phone® client application including backend and frontend business logic.

Chapter 4 focuses on finding possible solution based on protocols, technologies and framework research and implementing the backend data synchronization solution with all necessary components required for Windows Phone® client application to work.

Chapter 5 focuses on Windows Phone® application user interface design and prototype Windows Phone® client application development for CSI Lawyer®.

Chapter 6-7 focuses on identified problems during this project, client feedback analyses, future development suggestions of the client application and conclusion of the project.

2 Windows Phone® Platform

2.1 History

Microsoft has been trying to make its place in mobile computing since 1996 with its first version of Windows CE also known as Windows Embedded Compact. Windows CE was originally developed for mobile devices with low memory and low processing speed and it was mainly used for embedded systems. [9]

The Windows CE operating system created the basis for later mobile operating systems such as Pocket PC and Windows Mobile developed by Microsoft. Windows Mobile development started in 2000 when Microsoft first introduced Pocket PC 2000 operating system. Pocket PC was based on Windows CE 3.0 and had backward compatibility with the previous version of Windows CE. Pocket PC 2000 was not supporting normal mobile phone functionality. The figure 6 illustrates the Windows CE timeline. [9]

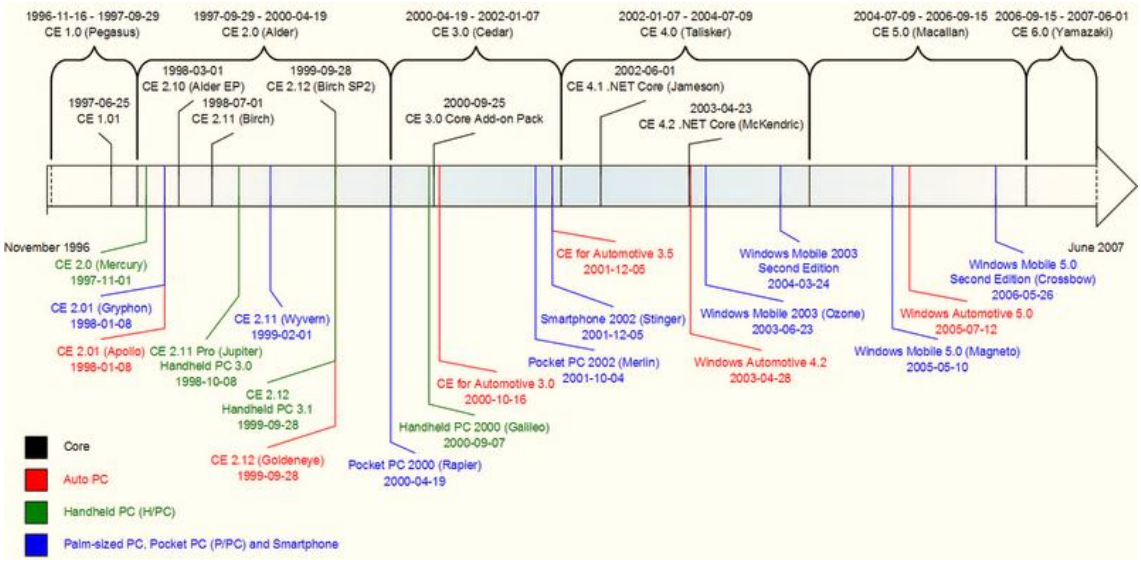


Figure 6: Windows CE Timeline. Reprinted from Windows Mobile [10]

In 2002 Microsoft released the Pocket PC 2002 which supported the basic functionality of mobile phones. After the release of Pocket PC 2002 Microsoft released the first fully functional mobile operating system which was known as Windows Mobile 2003. The Windows Mobile operating system was based on Windows CE 4.2 code.

In 2005 Microsoft released another mobile operating system family known as Windows Mobile 5 which has been supporting .Net Compact Framework 1.0. Later three more versions 6, 6.1 and 6.5 of Windows Mobile were released during next 5 years. The figure 7 illustrates the Microsoft's path to Windows Phone®. [10]

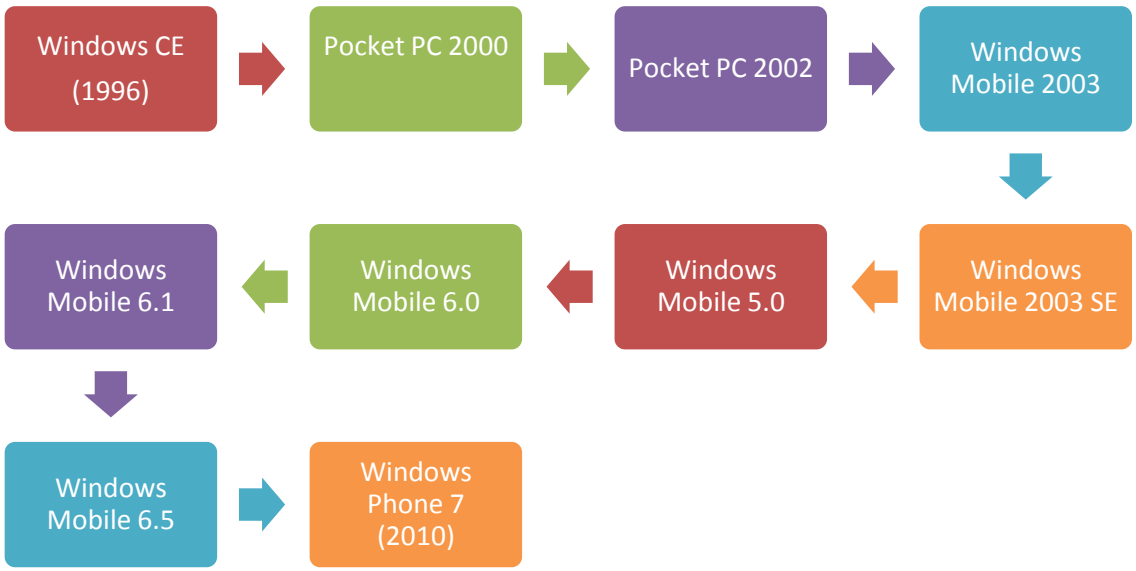


Figure 7: Microsoft's path to Windows Phone® [10]

The Windows Mobile operating system has not been successful especially on touch screen smartphones due to numerous problems such as slowness, in-responsiveness as it was not originally designed and optimized for touch screens, as well as due to other new smart phone operating systems such as Symbian, Android, iOs in the market, Windows Mobile continued to lose its market share during the past few years between 2005-2010.[8] In February 2011 Microsoft announced a completely new mobile platform currently known as Windows Phone®. It has been completely re-designed from scratch and specially optimized for touch screen mobile devices. Windows Phone® operating system is open to all manufactures which means Windows Phone® devices are made by several manufactures and available with a variety of network providers. [1]

2.2 Platform overview

In the past few years, many different mobile operating systems have been launched by different manufactures. These mobile operating systems are easy to use by end users and easy to develop new applications by developers. Apple's iOs and Google Android have made success in the mobile phone market due to their user interfaces and the vast number of applications available for their platforms. The vast number of rich applications plays a major role in the success of any mobile operating system. In order to make vast number of applications available to end users, mobile platform manufactures encourage the developers to create applications for their mobile platform by providing rich development tools and technologies which require minimum effort to learn and develop applications.

It is confusing and difficult for developer to choose, learn and work with different mobile platforms and programming languages which can be a problem in the success of any mobile platform if it fails to attract developers. For example Nokia's Symbian operating system made its way to the smart phone market much earlier than Apple's iOs, but still iOs leads the market with total number of rich application available for their mobile operating system. The reason why Symbian failed to attract developers was coding and tool complexity, which made it take more time to develop similar applications for Android or iOs. [11]

Microsoft has a long history in the software market for good developer friendly technologies and tools. It was a challenge for Microsoft to come up with something which would attract their existing developers to develop applications for their new mobile operating system.Microsoft utilized their existing technologies to create a stunning application platform for Windows Phone®. All programs for Windows Phone® are written in .NET using existing Microsoft® tools and technologies such as Visual Studio, Expression Blend®, Silverlight®, and the XNA Framework. Microsoft is targeting Windows Phone® not as an another mobile phone operating system but as an rich ecosystem where existing Microsoft tools, technologies and services can be used by various Windows Phone® manufactures and developers as illustrated by figure 8.

Figure 8: Windows Phone® - Ecosystem

Microsoft does not manufacture Windows Phone® hardware. Microsoft requires mobile phone vendors to implement at least a minimum set of hardware and software features for Windows Phone®. The figure 9 illustrates various Windows Phone® devices manufactured by different mobile phone vendors.

Figure 9: Windows Phone® - Devices

Windows Phone® platform allows developers to develop and run their applications across multiple devices from different manufacturers without worrying about whether there is sufficient memory, orientation support, camera, common sensors such as GPS and accelerometer. Below are the current Windows Phone® minimum hardware requirements [12]

- Common set of hardware controls and buttons such as Start, Search, and Back buttons.
- A large WVGA (800 x 480) or (320 x 480) format display capable of rendering most web content in full-page width and displaying movies in widescreen.
- Capacitive 4-point multi-touch screens for quick, simple control of the phone and its features.
- Support for data connectivity using cellular networks and Wi-Fi.
- 256 MB (or more) of RAM and 8 GB (or more) of flash storage.
- A-GPS
- Accelerometer.

All the applications created for Windows Phone® are delivered to end user phones using Windows Phone® Marketplace®. Applications submitted to marketplace must meet the minimum standards of reliability, efficiency, and good behavior set by Microsoft. Developers can manage all the applications submitted to Marketplace and follow sales and other user trends from online service called App Hub®.

2.3 Architecture

Windows Phone® applications are developed in managed code. Each application on Windows Phone® run isolated from each other and from the operating system in their own sandbox. Developers can use the Windows Phone® built-in library features, sensors and built-in cloud services to write our own applications using one of the existing application frameworks such as XNA or Silverlight. It is also possible to write interactive applications using web programming languages such as HTML/JavaScript/CSS which will be technically a Silverlight application with web browser control.

All the libraries available for desktop version of Silverlight and .Net framework are not supported whereas many phone specific libraries are supported which helps the developer to access phone data, sensor data, camera, push notifications, radio and built-in user Windows Phone® user interface controls. Windows Phone® Application Platform architecture is illustrated by figure 10. [13]

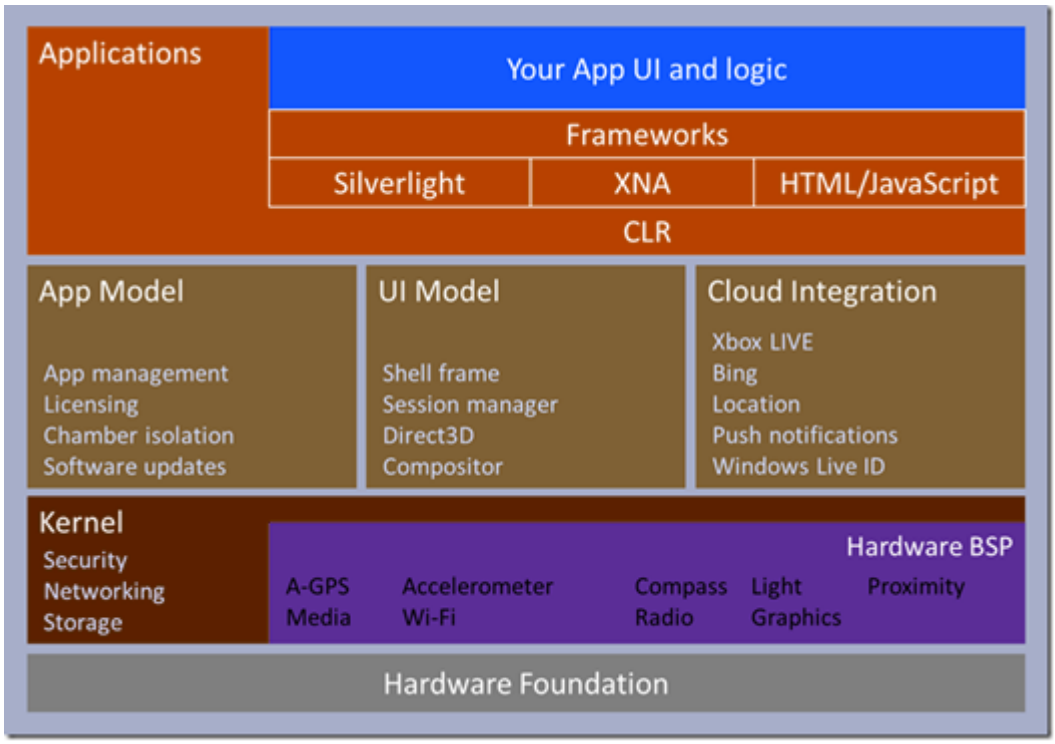


Figure 10: Windows Phone® - Application Platform architecture

Many third-party commercial and non-commercial control libraries are available on the Internet which helps developers to save time from developing commonly used user interface controls and communication related mechanisms.

2.4 User interface features and concepts

Rather than filling the small desktop with icons and other graphics, Microsoft engineers decided to design simple, sleek, quick, light and modern user interface which is responsive on touch screens. Microsoft calls the user interface of Windows Phone®, Metro UI. The design language used for designing the user interface is called Metro design language. Metro design language features hard edges, full bleed pages, sharp typography and fresh UI. Compared to its competitors Symbian, iOs, Blackberry and

Android, Metro user interface is a completely different experience on mobile devices which simply works and does not need much learning. Metro design language is based on following main principles. [8,24]

- Clean, Light, Open, Fast
- Feels Fast and Responsive
- Content not chrome
- Fierce Reduction of Unnecessary Elements
- Delightful Use of Whitespace
- Full Bleed Canvas
- Type is Beautiful, Not Just Legible
- Clear, Straightforward Information Design
- Uncompromising Sensitivity to Weight, Balance and Scale.

All the main information such as phone numbers, status updates, emails, pictures, videos, music, documents and applications are organized in their own special places called Hubs. Each Hub contains multiple views combined in panoramic view which can be accessed by sliding left and right by a finger as illustrated in figure 11. Windows Phone® provides six out of box Hubs: People, Pictures, Music + Video, Games, Office and Marketplace. [12,10]

Figure 11: Windows Phone® - Hubs

Inspired by a typography design start screen, the Windows Phone® is populated with something similar to either metro or airport banners known as live tiles. Live tiles can be placed on the start screen for each Hub and applications which support live tiles as illustrated by figure 12. Live tiles provide links to applications and update themselves

with live information received from a Hub, application or Internet to inform the user of any changes since the last run.

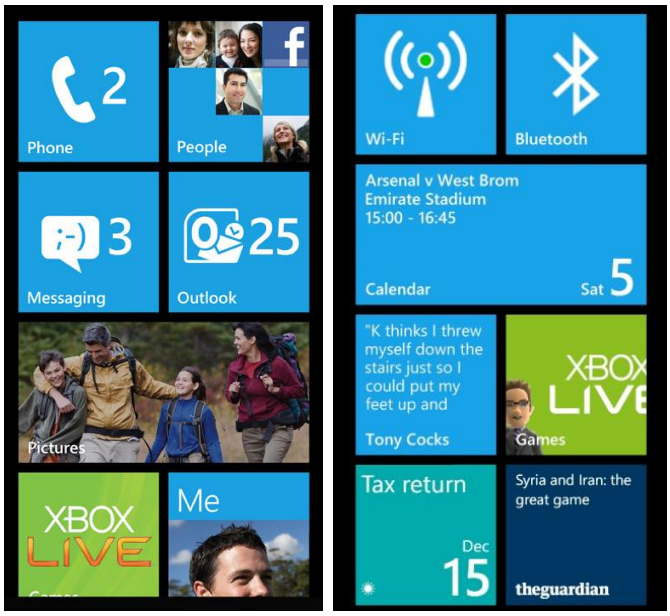


Figure 12: Windows Phone® - Live Tiles

Application tiles can consist of three components. [12,151-153]

- **Background Image:** a 173 x 173 image used to make up application tile background.
- **Title:** the text that will appear on the tile.
- **Count:** the number of updates or changes relating to the application.

2.5 Development environment and tools

The Windows Phone® Application Platform offers a large set of tools, a cohesive and well-designed managed API set, runtime services on devices that can be used to access local phone data, sensors data and cloud data such as Xbox LIVE®, Windows Azure, location, and notification services.

Microsoft Visual Studio® with Windows Phone® SDK is the main tool used for developing Windows Phone® applications as illustrated by figure 13. Microsoft Visual Studio® is an integrated development environment (IDE) which can be used for

debugging, coding and testing. Like any other .Net applications, Windows Phone® applications can be written using native .Net languages such as C# or VB.Net.

Windows Phone® emulator provided by Windows Phone® SDK simulates a copy of the actual phone operating system and is similar to real phone runtime in all respects except for performance and sensors. However all the developers are recommended to test the final application before distribution on a real Windows Phone® device to eliminate any device specific capability issues. [12, 24-26]

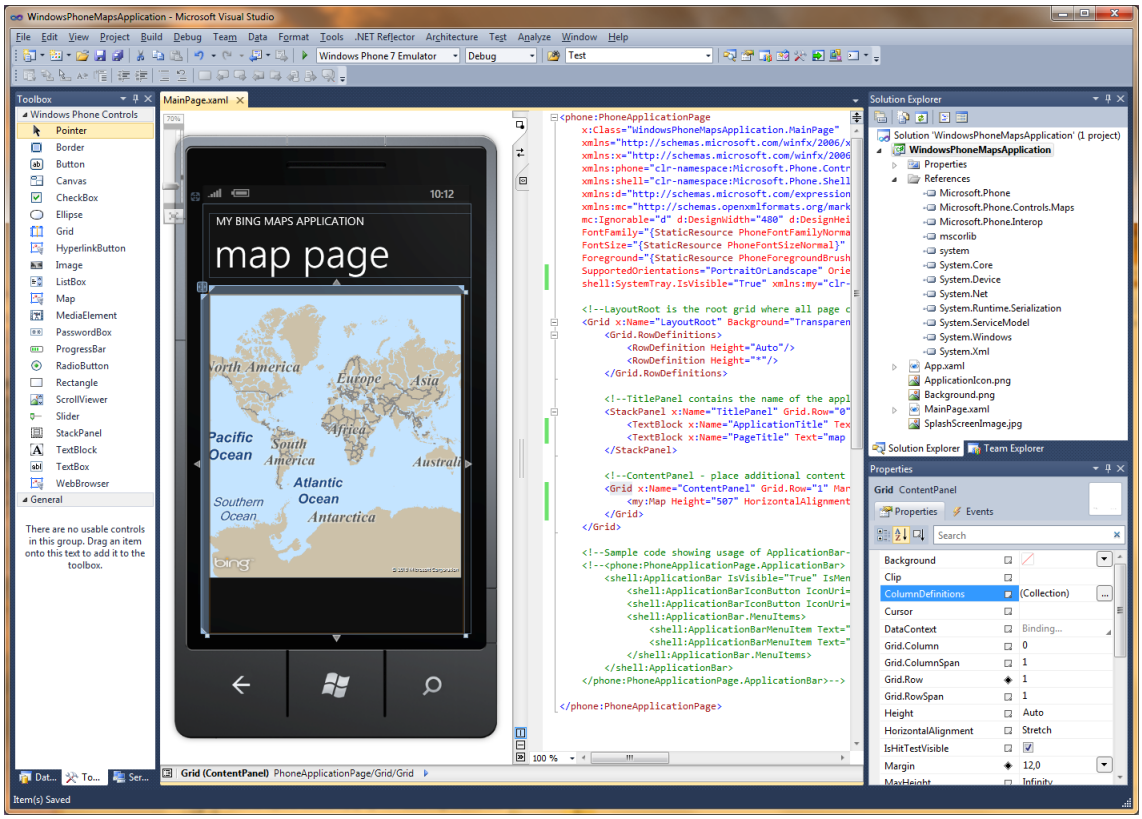


Figure 13: Windows Phone® - Development environment on Visual Studio® 2010

All the Windows Phone® applications are packed as an XAP file which can be used to deploy applications on the end user phone directly or through Windows Marketplace. Windows Phone® applications can only be deployed directly on developer unlocked phones. Windows Phone® Platform has been designed to make programming applications reusable, flexible and easier to code. In addition to .Net Framework as a base of Windows Phone®, user interface of Windows Phone® application is designed by two popular and modern programming platforms: Silverlight and XNA.

Silverlight is web application platform derived from Windows Presentation Foundation (WPF) technology. It is used to create stunning, compelling, attractive, sophisticated and interactive user interfaces with a mix of traditional controls, high-quality text, vector graphics, media, animation, and data binding that run on multiple platforms and browsers. Silverlight application's user interface is written using Extensible Application Markup Language (XAML). On Windows Phone® Platform, Silverlight 3.5 uses extended set of phone-specific API libraries and controls specially designed for Windows Phone® Platform. Microsoft Expression Blend can be used to develop visuals and animations for Silverlight applications.

XNA is Microsoft's game platform supporting both 2D sprite-based and 3D graphics with traditional game-loop architecture. Although XNA is mostly used for writing games for the Xbox 360 console, developers can also use XNA to write rich graphic applications for Desktop PC and Windows Phone®. XNA gives developers the capability to directly access features of the device such as video and sound systems, where this is necessary to provide the performance required for highly interactive gaming and associated types of applications. XNA Game Studio can be used to develop applications with XNA framework.

Windows Phone® 7.1 API allows developers to combine both frameworks to create graphically rich business applications by mixing GUI elements with lot of motion or particle effects.

2.6 Windows Marketplace and App Hub

Such as many other mobile operating ecosystems, Windows Phone® has its own application distribution channel known as Windows Marketplace. All the applications distributed through Windows Marketplace must meet a set of acceptable minimum criteria for quality and compatibility required by Microsoft to give a consistent and secure user experience. Windows Marketplace can be accessible through Windows Phone® built-in Marketplace Hub to search and install applications on the device. It is also possible to search and deploy Windows Phone® applications using Zune desktop software provided by Microsoft or through Windows Marketplace website after

registering a developer account with Microsoft. All these three Windows Phone® application distribution channels are illustrated by figure 14.

Figure 14: Windows Phone® - Application distribution channels

All the Windows Phone® applications are submitted by developers to the marketplace using a back end web based service called App Hub®. App Hub® provides complete details on submitting Windows Phone® applications to marketplace with all necessary certification requirements. All applications submitted to Windows Phone® Marketplace must pass certification testing before being published to end users. The figure 15 illustrates Windows Phone® application certification and distribution process. [14]

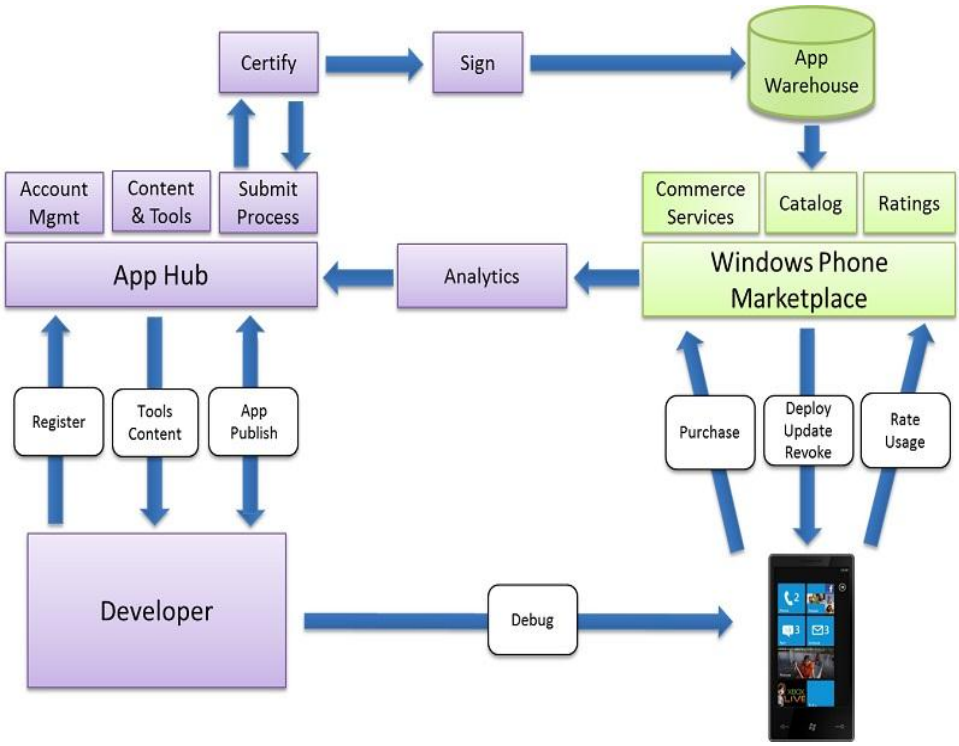


Figure 15: Windows Phone® - Application distribution process

Though the App Hub®, developers can follow the popularity, downloads and sales of their applications as well as make them available or unavailable to specific regions or sets of people. All the information related to published and sold applications is well organized on App Hub® dashboard as illustrated by figure 16.

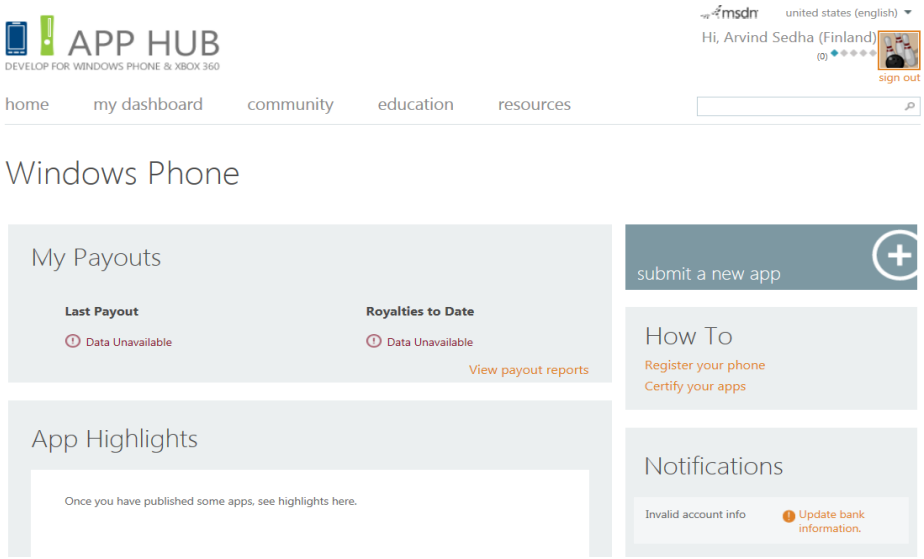


Figure 16: Windows Phone® - App Hub® Dashboard

At the time or writing this project Windows Marketplace supported 35 different countries. [15]

3 Communication protocol and frameworks

3.1 Overview

As defined in the requirements, the CSI Lawyer® client application must be able to synchronize data with the backend system when possible and make use of data on the Windows Phone® in offline mode. In Windows Phone® 7.1, Microsoft introduces support for SQL CE 3.5 databases which allows Windows Phone® applications to store local data in relational databases. At the time of writing this project, Windows Phone® only supported a database which existed locally on the phone. [16]. Windows phone isolated storage is another place, which can be used by applications to store and retrieve persisted data on Windows Phone®.
In the chapters 3.2 and 3.3, we will discover some of the possible data exchange message formats, available technologies and frameworks to achieve our task of performing backend data synchronization between CSI Lawyer® and CSI Mobile® application.

3.2 Communication protocols and data-interchange formats

There are a wide variety of message exchange protocols and data-interchange formats available for exchanging the data between various systems efficiently and securely over the Internet. While choosing the right communication protocol for exchanging information for mobile devices, it is important to consider compatibility, performance, expandability and security. Below are some of the possible communication protocols and data-interchange formats which can be used for synchronizing the data between CSI Lawyer® and CSI Mobile® application.

JSON stands for JavaScript Object Notation which is based on a subset of the JavaScript Programming Language. JSON is lightweight text-based data-interchange format which is completely language independent and easy for humans to read and write as listing 1 illustrates.

```
{
      "customerName"   : "Arvind Sedha",
        "customerNumber": "59129299",
      "address"  :
      {
          "streetAddress": "Mannerheimintie 1",
          "postalcode"   : "00100",
          "city"         : "Helsinki",
          "country"      : "Finland",
      },
      "phoneNumber":
      [
          {
            "type"  : "business",
            "number": "050-3248288"
          },
      ]
 }
```

Listing 1: JSON response

It is used for representing simple objects such as data structures and associative arrays. A large number of web applications use JSON objects for consuming web services through JavaScript. For certain type of data, JSON tends to be smaller than XML thus requires less amount of data transfer over the network. Windows Phone® SDK also supports built-in JSON data parser. Such as XML Schema for XML, JSON Schema can be used for JSON to provide a contract for what JSON data is required for a given application and how it can be modified.

Web services are a special piece of the software system designed to support interoperable machine-to-machine interaction over a network by making resources available in the XML format. Web services use standard protocols such as TCP, HTTP, or SMTP to transfer the data over the Internet. It is possible to consume web services in Windows Phone® application easily just by referencing the web application in the Windows Phone® project. [16,383-387]

Web Services Description Language (WSDL) language is used to describes the web service interface for communication, operations and message exchange. WSDL specification can be divided into six major elements as illustrated by figure 17.

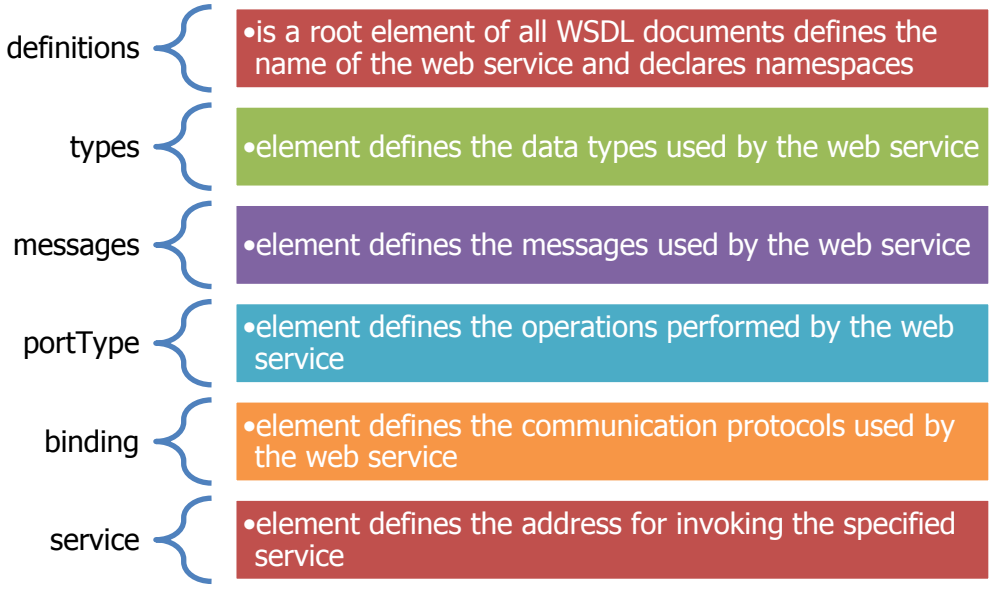


Figure 17: WSDL specification six major elements

As WSDL is a plain XML and can be consumed by any kind of system. Developers can make use of the web service and invoke it by having the complete WSDL provided by web services. Nowadays tools such as Visual Studio are able to generate fully working proxy classes to communicate with web service over the Internet by using WSDL. [16,383-387]

A SOAP once stood for Simple Object Access Protocol is a simple XML-based protocol developed by Microsoft to let applications exchange information over HTTP or HTTPS. Since the SOAP version 1.2 the acronym was dropped. SOAP describes envelope and message formats which provide a way to communicate between applications running on different operating systems, with different technologies and programming languages. The listing 2 illustrates simple example of a SOAP request for searching customer information using SOAP 1.1.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetCustomerInfo xmlns:m="http://namespaces.customer-info.com">
      <customerNumber>815155</ customerNumber >
    </m: GetCustomerInfo >
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 2: SOAP request

SOAP messages are sent over the Internet using http/https post messages. All the SOAP messages must have sender, message operation and message length information included in the soap message as illustrated in listing 3.

```
POST /CustomerInfo HTTP/1.1
Host: www.customer-info.com
Content-Type: text/xml; charset="utf-8"
Content-Length: 206
SOAPAction: http://www.customer-info.com/CustomerInfo
```

Listing 3: SOAP headers

As a response we receive the soap message as illustrated in listing 4.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetCustomerInfoResponse
      xmlns:m="http://namespaces.customer-info.com">
      <customerNumber>815155</ customerNumber >
      <name>Arvind Sedha</name>
      <birthYear>1983</birthYear>
    </m:GetCustomerInfoResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 4: SOAP response

OData stand for Open Data Protocol. It was originally created by Microsoft as web protocol for exposing relational data across the service layer over the network. OData has been built on the top other existing web technologies such as HTTP, Atom Publishing Protocol (AtomPub) and JSON as illustrated by figure 18. OData uses REST-based Uri syntax to expose and access information from a variety of sources including relational databases, file systems, content management systems and traditional Web sites over the network. OData supports two different data formats: JSON and AtomPub to communicate in both directions. [17]

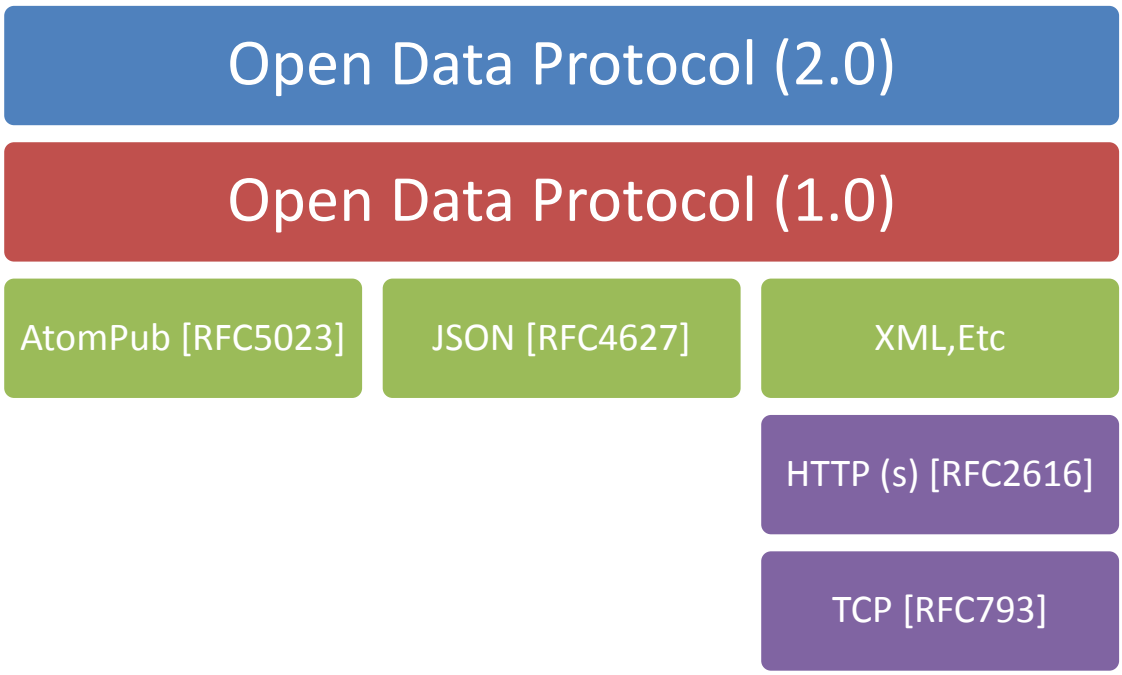


Figure 18: OData Architecture Stack

OData can be used for querying, shaping, filtering, ordering, paging and updating data across the Internet by using typical data operations over the Internet. OData leans on HTTP stack to allow for different HTTP verbs to mean different operations. OData maps HTTP verbs to these CRUD operations are shown in the table 1. [16]

HTTP Verb	Data Operation
GET	Read
POST	Update
PUT	Insert
DELETE	Delete

Table 1: OData HTTP Verb Mappings

OData has become popular over the past few years and is now natively supported and implemented in products by many big software vendors such as Microsoft, IBM, and SAP. Windows Phone® also provides built-in SDK libraries to consume OData web services.

3.3 Platform, Framework and Technologies

3.3.1 Windows Phone® Local databases

Since the release of Windows Phone® 7.1 SDK, applications are allowed to store the relational data in the local database. The application local database resides in a special container known as application Isolated Storage which means the database can only be accessed from the application that created it and cannot be shared between several applications.

The Local database in Windows Phone® 7.1 is an implementation of Microsoft SQL Server Compact (SQL CE) for Mango. SQL CE database on Windows Phone® is stored in a single ***.sdf*** file which resides in Isolated Storage of an application. The SQL CE database size can be up to 4 GB. Windows Phone® applications can access databases only through data context which acts as a proxy between the database and the application. No direct access to database using transact-SQL is supported by Windows Phone® application.

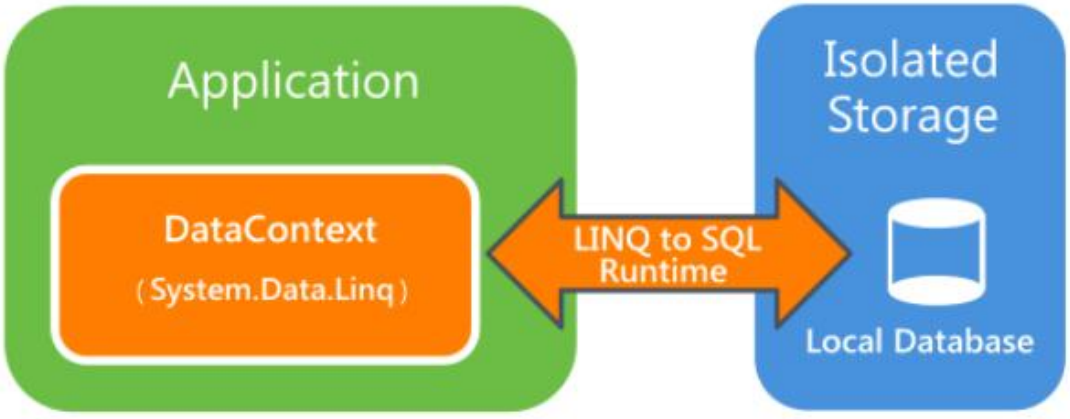


Figure 19: Accessing local data on Windows Phone®

All the database operations are carried out using LINQ to SQL object oriented approach as illustrated by figure 19. [18]

3.3.2 Windows Azure Platform

Windows Azure Platform is an application platform developed by Microsoft which runs on the Cloud. Windows Azure Platform provides different kind of services which allow users to quickly build, deploy and manage easily scalable applications across a global network of Microsoft-managed datacenters. All Azure services and application developed for Windows Azure Platform run on top of an operating system called Windows Azure as illustrated by figure 20. [19]

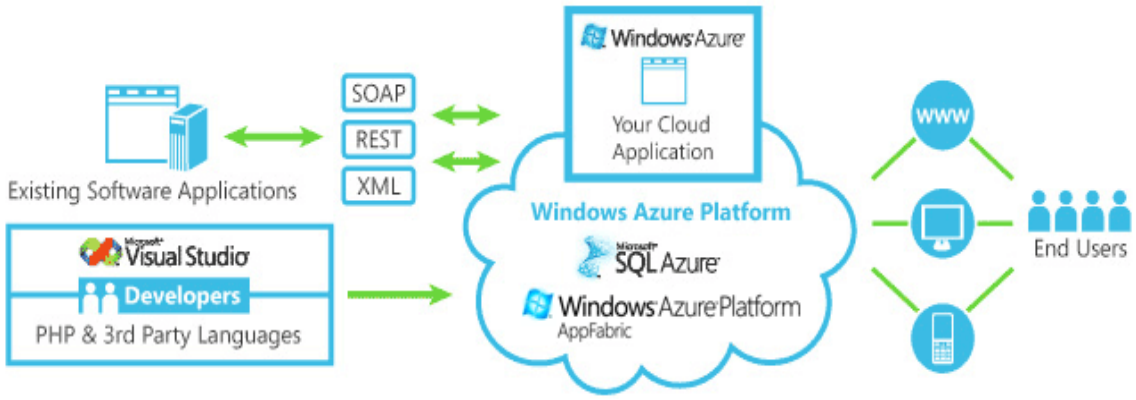


Figure 20: Windows Azure Platform [19]

Windows Azure enables the developer to use any development language, Framework, and tools to build applications. Windows Azure Platform features and services are exposed using API built on REST, HTTP, and XML. Microsoft also provides azure client libraries released under an open source license. These libraries and hosted on GitHub for various existing technologies and existing tools such as .Net, PHP and Java, Netbeans, Eclipse and Visual studio. [19]

Applications and services running on Windows Azure Platform are easily scalable. Windows Azure allows users to grow or shrink their application and service resources based on their needs. Users only pay for the resources used by their application in real time. Windows Azure is available in multiple datacenters around the world which allows users to deploy their applications or services close to the location best suited for their customers. Windows Azure Platform consists of following three core components. [19]

Windows Azure is an operating system which provides various kinds of services and tools to develop and run applications on the Cloud for different programming languages and tools already available in the market. The Compute service of Windows Azure platform provides different roles suitable for running different kinds of Azure applications. Some of the informant roles provided by Windows Azure are listed below. [19]

- Worker Role – to run background processes without any user interface
- Web Role – to run web applications with full support of .Net, PHP, Java and many other existing technologies.
- VM Role – to run virtual services in cloud for applications which when your application which requires a large number of server OS customizations and cannot be automated
- Data storage – to store virtually all types of data, from structured to unstructured data, NoSQL databases, blobs, tables and queues. [19]

SQL Azure service provides the full implementation of SQL server in the Cloud which is easily scalable based on user needs. Databases hosted in the SQL Azure can be easily accessed by any application same such as the on-premises SQL database. It provides PHP support, native ODBC and managed ADO .NET access. With the use of SQL Azure, there is no need of installing and configuring SQL Server for developers. The data stored on SQL Azure is backed up regularly on various Microsoft data centers automatically thus leaving little work for database administrators. [19]

AppFabric allows the users to build hybrid applications by connecting applications located on several servers in the cloud or on-premises. It includes applications running on various platforms such as Windows Azure, Windows Server, Java, Ruby, PHP. AppFabric consists of two big parts Service Bus and Access control. Service bus allows connecting of various applications and services running on different platforms and services in the cloud or on-premises environment. Access Control service provides various types of authentication and authorization. Access Control service supports various open and built on industry standards such as OAuth 2.0, WS-Trust, WS-Federation, SAML 1.1, SAML 2.0, and Simple Web Token (SWT) token formats. [19]

3.3.3 Microsoft Sync Framework

The growing numbers of mobile devices have created a great demand for data to be available simultaneously on various kind devices such as laptops, office desktops, Smartphones, or PDAs. Generally this kind of issue are solved in companies by providing access for employees to the corporate networks through VPN connections, Web servers, or some other connectivity method. This kind of solution can have many disadvantages such as network requirements, data access speeds or single point of failure etc.

One way to solve these kinds of problems is to build an application for mobile devices which occasionally connects to the network and only synchronizes the data required by the mobile device. These kinds of applications are called Occasionally Connected Application (OCA). Occasionally connected applications often suffer from problems such as how, what and when to synchronize the data among different devices. Generally this is solved by developers by creating some kind of custom synchronization business logic which defines what, when and how to synchronize the data amount various devices. The most common problems with custom synchronization solution are functionality limitations, time consuming development, non-flexible and difficult to maintain it.

Microsoft Sync Framework has been developed by Microsoft especially to provide a single solution for synchronization of various kinds of data including relational databases, file systems, lists, devices, PIM, music, video, RSS feeds etc. among different kind of devices. It adds synchronization, roaming, and offline capabilities to applications and provides full support for correct multi-master synchronization. By using the Microsoft Sync Framework, developers can build synchronization ecosystems that integrate any application, any data from any store using any protocol over any network. Microsoft Sync Framework comes with many out-of-the-box providers such as database synchronization providers, file synchronization provider and web synchronization components. If out-of-the-box providers are not enough to satisfy the need, developers can create their own custom providers to exchange information between devices and applications. [20]

The heart of the Microsoft Sync Framework is a metadata store which stores the tracking information of each data row to be synchronized as illustrated by figure 21. Metadata can be stored in a file, within a database or within the data source being synchronized.

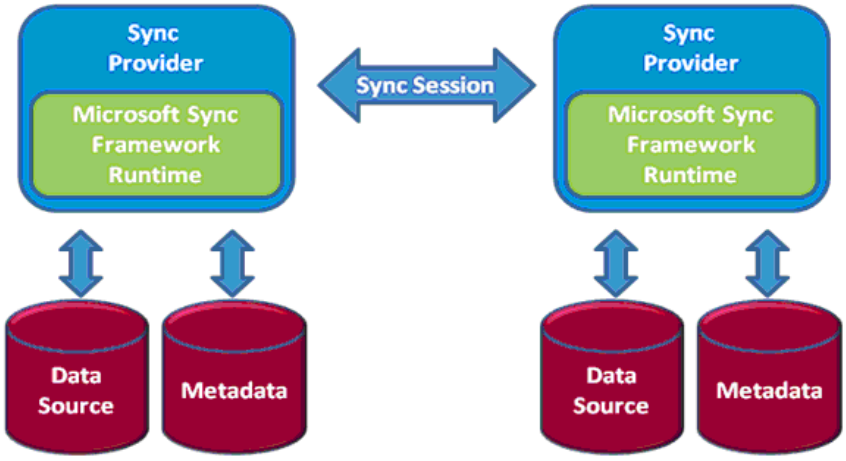


Figure 21: Microsoft Synchronization Framework core components

The Microsoft Sync Framework 4.0 October 2010 CTP is built on top of Microsoft Sync Framework 2.1. It makes use of the OData protocol. OData is an open standard, and thus allows all kinds of platforms to sync data with the Microsoft Sync Framework and build offline applications on any client platform capable of caching data. Microsoft Sync Framework already provides out-of-box database provides to sync SQL Server, SQL Azure, SQL CE databases. [21]

3.3.4 SQL Server Replication

SQL server replication is a set of technologies for copying and distributing data and database objects from one database to another. SQL replication performs synchronization between databases for many reasons such as for load balancing, offline processing and data redundancy. Using replication, system can distribute data to different locations and to remote or mobile users over local and wide area networks, dial-up connections, wireless connections, and the Internet. [22]

4 Backend development

4.1 Overview

In order to proceed with the development of Windows Phone® client application for CSI Lawyer®, CSI Helsinki Oy need to first develop the backend solution which allows users to synchronize the required data from CSI Lawyer® database to Windows Phone® device.

CSI Lawyer® Windows Phone® application must use isolated storage or compact database to store data locally on the phone. The CSI Lawyer® database might be stored on the customer's local environment or in the cloud environment. Due to the missing support for accessing databases directly by the Windows Phone®, there is no way for synchronizing the data from CSI Lawyer® database directly.

The technical requirements also define that the backend solution must be developed in a manner that there should not be any need for the customer to open ports or configure firewalls to make the data synchronization work between Windows Phone® and the CSI Lawyer® database. If the CSI Lawyer® database is located in highly restricted local area network, opening the ports in the firewall can be major security risk as whole financial data stored in CSI Lawyer® database would be exposed over the Internet. In order to solve this problem CSI Helsinki Oy should have another intermediate database which can be accessed through Internet by Windows Phone® or any other application. The one option for this intermediate database would be on Azure where it can be scaled seamlessly based on customer requirements.

Microsoft SQL Azure is fully scalable cloud based service built on the SQL server technologies which provides a full featured relational database. For our backend solution CSI Helsinki Oy is going to use SQL Azure for storing our intermediate relation database to ensure security, availability, reliability and scalability of the data. This intermediate database will allow CSI Lawyer® customers to synchronize required data between CSI Lawyer® database and Windows Phone® device.

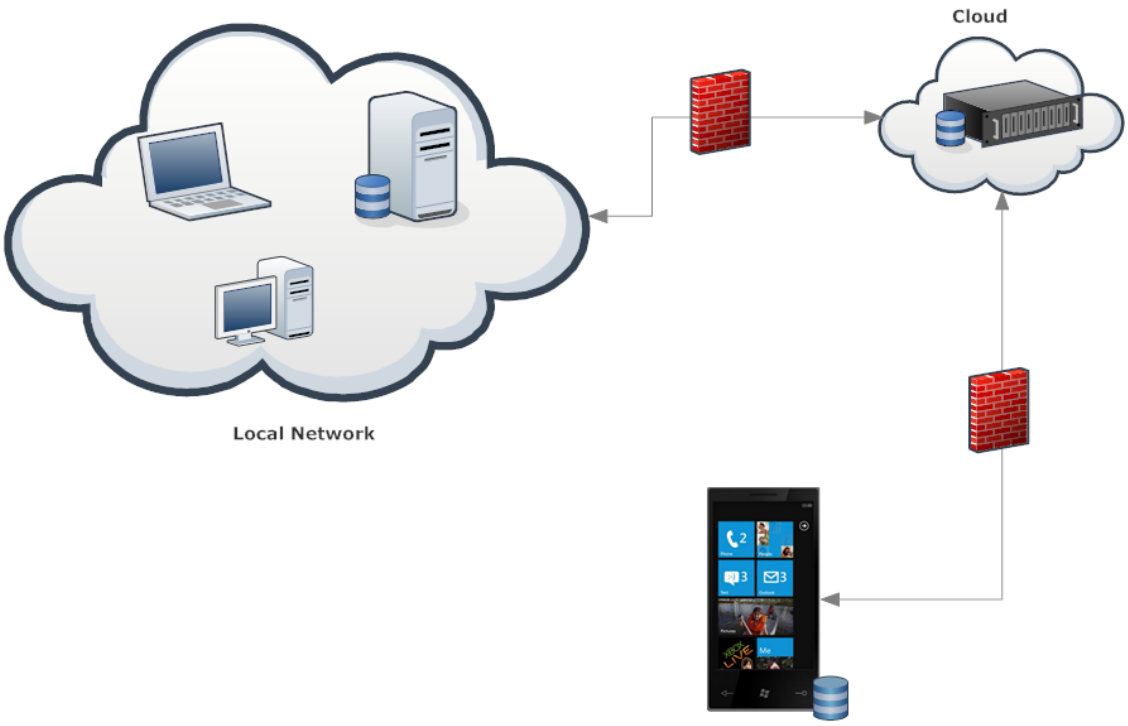


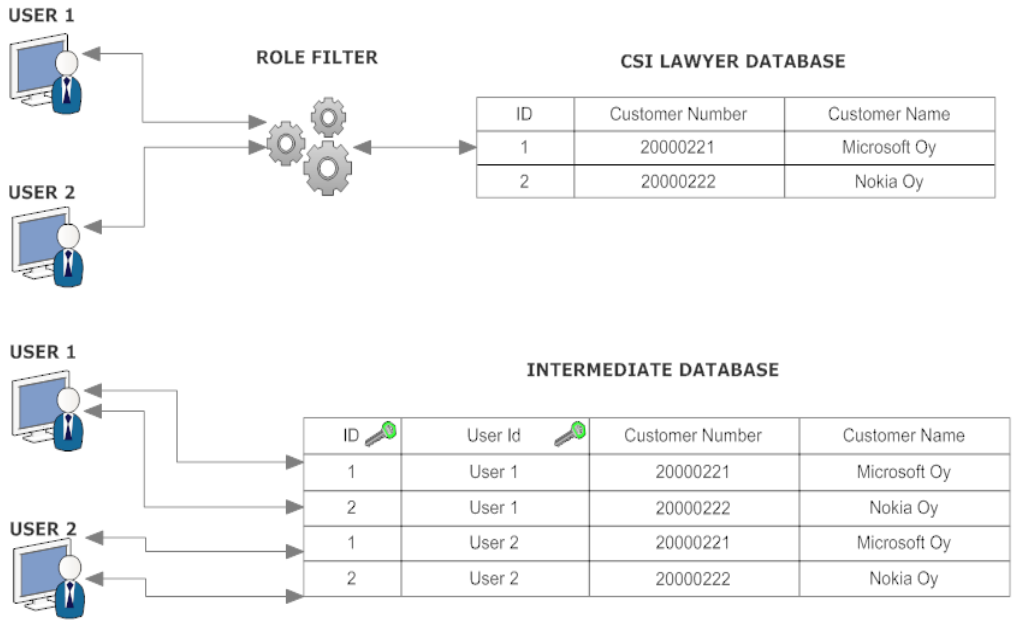
Figure 22: Intermediate database architecture

The solution illustrated by figure 22 has three different databases where each one resides in their own environment. In this architecture, Windows Phone® application database and CSI Lawyer® database never synchronize data directly. The intermediate database is used as a proxy between CSI Lawyer® database and Windows Phone® database for synchronizing data. The next problem is to find the solution on how the data is exchanged between these three databases.

CSI Lawyer® uses role based security which means none of the data rows are owned by single a user instead shared amount different users based on their roles and access rights. The intermediate database is going to be user based means every single data row in the intermediate database is owned by only a single user to ensure the integrity of the data illustrated by figure 23.

The backend solution for synchronizing the data between CSI Lawyer® and intermediate data consists of Windows client application which resides on the user machine where CSI Lawyer® is installed. The Windows based client application runs in background with minimalistic user interface to synchronize the data between CSI

Lawyer® and intermediate database. It only synchronizes the user specific data required by Windows Phone® application.



ID	Customer Number	Customer Name
1	20000221	Microsoft Oy
2	20000222	Nokia Oy

ID	User Id	Customer Number	Customer Name
1	User 1	20000221	Microsoft Oy
2	User 1	20000222	Nokia Oy
1	User 2	20000221	Microsoft Oy
2	User 2	20000222	Nokia Oy

Figure 23: CSI Lawyer® database vs. intermediate Database

The Windows Phone® does not support any kind of direct database connectivity such as desktop based .Net applications. Currently Windows Phone® only supports access to Windows CE database located on device using LINQ to SQL data context. The only possible way to expose the data from the intermediate database running on cloud is by using on one of the web based protocols such as SOAP or O-Data which are well supported by Windows Phone® platform.

4.2 Possible solutions

Based on the analysis on available protocols, frameworks and technologies CSI Helsinki Oy can synchronize the data between Windows Phone®, intermediate and CSI Lawyer® databases using custom sync solution or Microsoft Sync Framework. However CSI Helsinki Oy is going to use Microsoft Sync Framework solution in this project to implement the backend data synchronization as it requires less work and fits perfectly out-of-box for our problem.

4.2.1 Custom data synchronization solution

In a custom sync solution CSI Helsinki Oy need to design, develop and implement fully hybrid solution across the multiple platforms for tracking, sending, receiving and securing data contents. Because the intermediate database and local CSI Lawyer® databases are Microsoft SQL server databases, CSI Helsinki Oy can use SQL data replication to synchronize the data between these databases with some sort of intermediate business logic. In intermediate business logic CSI Helsinki Oy need to verify and convert the data into the right form expected by each database.

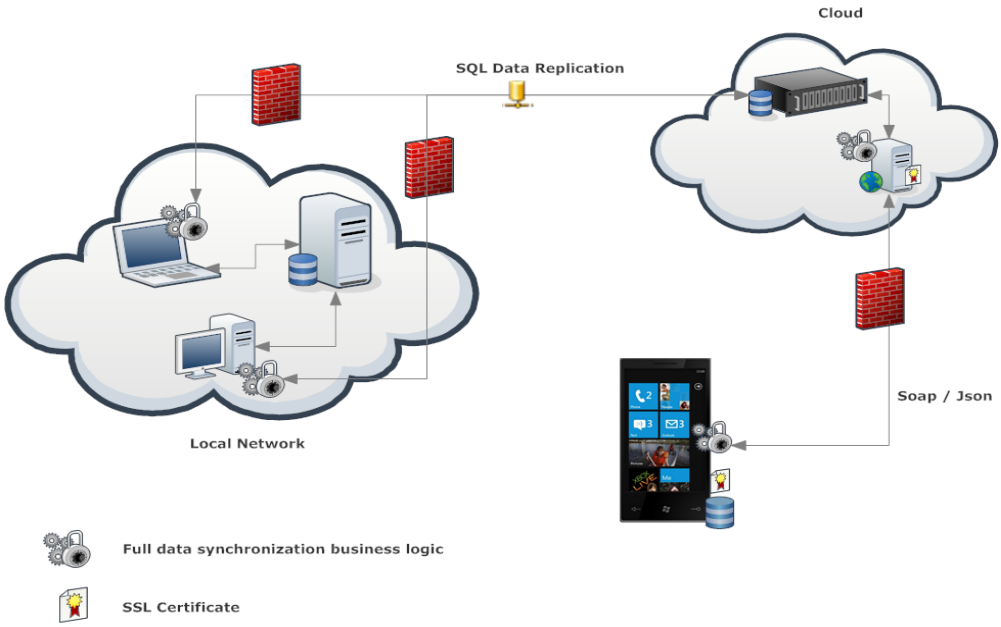


Figure 24: Custom data synchronization solution

Data synchronization between Windows Phone® and intermediate database can be achieved using web service running in the cloud which exposes the data from intermediate database in SOAP or JSON format as illustrated by figure 24. On the mobile device CSI Helsinki Oy need a fully custom solution to track and send changes securely to the intermediate database over the web.

This kind of solution needs a lot of effort from developer to implement a fluent, secure and flexible system. The Windows Phone® application can also become quite huge as the developer needs to implement a lot of business logic for tracking and synchronizing data between Windows Phone® and intermediate database.

4.2.2 Microsoft Sync Framework data synchronization solution

In the Microsoft Sync Framework solution CSI Helsinki Oy are going to use Microsoft Sync Framework as a backbone for data synchronization between different databases and Windows Phone® device as illustrated by figure 25.

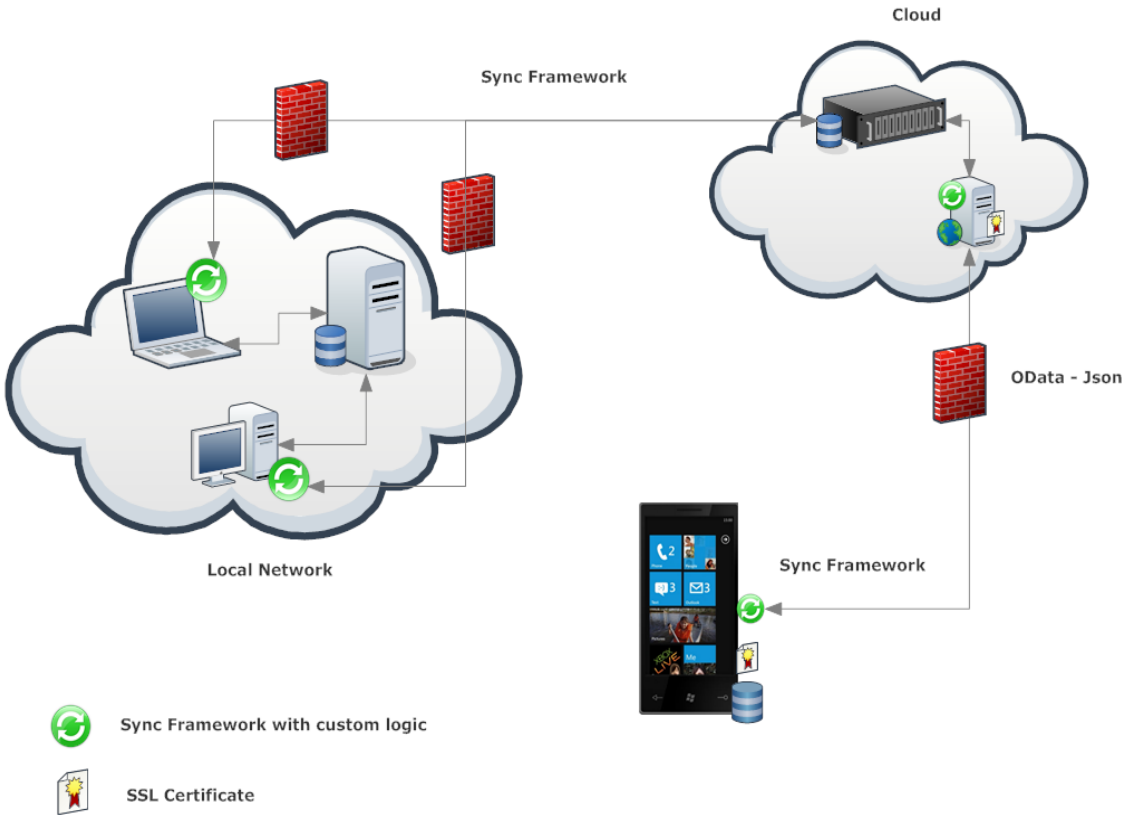


Figure 25: Microsoft Sync Framework data synchronization solution

The current release of Microsoft Sync Framework called Sync Fx which is based on Microsoft Sync Framework 4.0 CTP. Microsoft Sync Framework comes with a provider for SQL Server and SQL Azure out of the box. For Windows Phone®, Microsoft Sync Framework provides components for Isolated Storage so CSI Helsinki Oy can use it without writing a line of code.

CSI Helsinki Oy is going to use SQL server and SQL Azure providers to synchronization data between local CSI Lawyer® and intermediate database. Due to the role based security on CSI Lawyer® database and user based security on intermediate database, CSI Helsinki Oy need to implement custom data synchronization logic to modify the

data when it is transferred to and from CSI Lawyer® database. This is not the problem with Microsoft Sync Framework at all, as CSI Helsinki Oy can customize every piece of business logic used by Microsoft Sync Framework to perform data synchronization. Because CSI Helsinki Oy want to keep the intermediate database structure as close to Windows Phone® database structure, all the data modifications required by CSI Lawyer® database and intermediate database must be done when synchronization is performed between the CSI Lawyer® database and the intermediate database. In this way CSI Helsinki Oy can ensure that data synchronization between Windows Phone® and intermediate database does not need any special business logic and data from intermediate database will be save on Windows Phone® as it is.

The data synchronization between the Windows Phone® and intermediate database is not possible in the same manner as between the CSI Lawyer® and the intermediate database because Windows Phone® does not support SQL server or SQL Azure providers for accessing SQL database directly. CSI Helsinki Oy have to synchronize the data between intermediate database and Windows Phone® device over the Internet using web protocols. Microsoft Sync Framework already supports the data synchronization over the web using OData protocol which exposes the data to any device using REST requests in JSON format. Because data is synchronized between Windows Phone® and intermediate database as it is without any modifications, CSI Helsinki Oy did not need any custom data synchronization business logic on Windows Phone® or sync web service. Data exposed by Microsoft Sync Framework over the web can also be modified before committed to and from database.

4.3 Synchronization Scope

CSI Lawyer® database is large and contains over 50 tables. The Windows Phone® device does not need all of this data stored on the CSI Lawyer® database as CSI Helsinki Oy did not want to synchronize anything extra to the intermediate database than what is required by the phone. In the Microsoft Sync Framework, it is possible to define the synchronization scope to limit the database tables, columns and data CSI Helsinki Oy want to synchronize. Even the source and destination databases can have different amount of tables and columns. Only items that fit within the scope description are tracked and synchronized.

4.3.1 CSI Lawyer® database

CSI Lawyer® database sync scope is going to be bit different than the intermediate database sync scope because backend application needs to track at least one extra table and some extra columns than intermediate database. Based on the functional requirements, backend application is going to synchronize only a user's favorite assignments and related information such as customer, contact, address, transactions and transaction templates as illustrated in table 2.

Table Name	Description	Primary Key
Assignment	Assignment information	record_guid
Customer	Assignment's customer information Contact's parent customer information	record_guid
Contact	Assignment's contact information Customer's primary contact information	record_guid
TransactionTemplate	Transaction template information	record_guid
Transaction	Transaction information	record_guid
User	User information	record_guid
Address	Customer and contact's address information	record_guid
Favorite	User's favorite assignments	record_guid

Table 2: CSI Lawyer® database synchronization scope

Favorite is the special table which CSI Helsinki Oy are not going to synchronize with intermediate database, but CSI Helsinki Oy need to include it in the sync scope to track when the user adds to or removes an assignment from favorite list. The figure 26 illustrates the local database model of CSI Layer® database with all tables participating in synchronization scope.

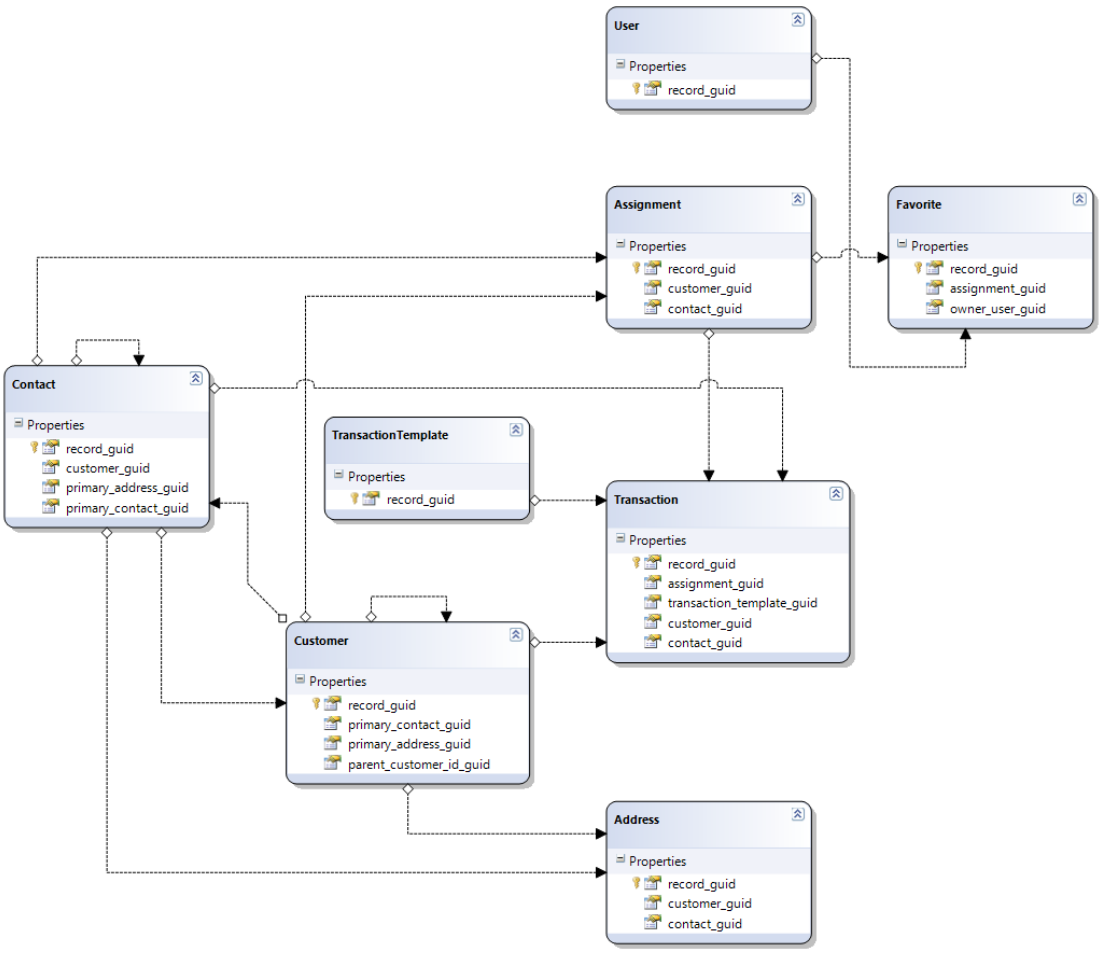


Figure 26: CSI Lawyer® database logical model

There is no possibility to filter the data from CSI Lawyer® database with simple Microsoft Sync Framework data filters. When an assignment is added to favorite, assignment and its related data must be handled by Microsoft Sync Framework as new records. When assignment is removed from favorite, assignment and its related data must be handled by Microsoft Sync Framework as deleted records. Only the Favorite table knows which assignment is added as favorite and to which the user but it does not know anything about assignment's related information such as customer, contact and transactions etc. To achieve the required functionality, CSI Helsinki Oy need to modify the SQL stored procedures used by Microsoft Sync Framework to select the changes during each sync operation.

Intermediate database is not completely similar to CSI Lawyer® database. Custom business logic is required before data is submitted to CSI Lawyer® and intermediate

database during sync operation. For example, before each row is synchronized from CSI Lawyer® database to intermediate database, CSI Helsinki Oy change the modified_by column to user_guid column and assign it the current user id.

4.3.2 Intermediate database

Intermediate database is simple in structure just as required by the Windows Phone®. Data synchronized between Windows Phone® and intermediate database is not modified by any kind of custom business logic thus keeping synchronization as simple as possible. All the rows in the tables are unique for the each user. In every table except the user table, the foreign key consists of record_guid and user_guid. CSI Helsinki Oy is going to create two identical synchronization scope templates; one for CSI Lawyer® database and other for Windows Phone®. Two different scope templates will ensure that data tracking works as expected for both Windows Phone® and CSI Lawyer® database. A table 3 illustrates tables included in the synchronization scope for the intermediate database.

Table Name	Description	Primary Key
Assignment	Assignment information	record_guid user_guid
Customer	Assignment's customer information Contact's parent customer information	record_guid user_guid
Contact	Assignment's contact information Customer's primary contact information	record_guid user_guid
TransactionTemplate	Transaction template information	record_guid user_guid
Transaction	Transaction information	record_guid user_guid
User	User information	record_guid
Address	Customer and contact's address information	record_guid user_guid

Table 3: Intermediate database synchronization scope

Filtering the data from intermediate database is simple because each row from the table belongs to only one user. CSI Helsinki Oy can user user_guid column to filter data from all the tables on intermediate database.

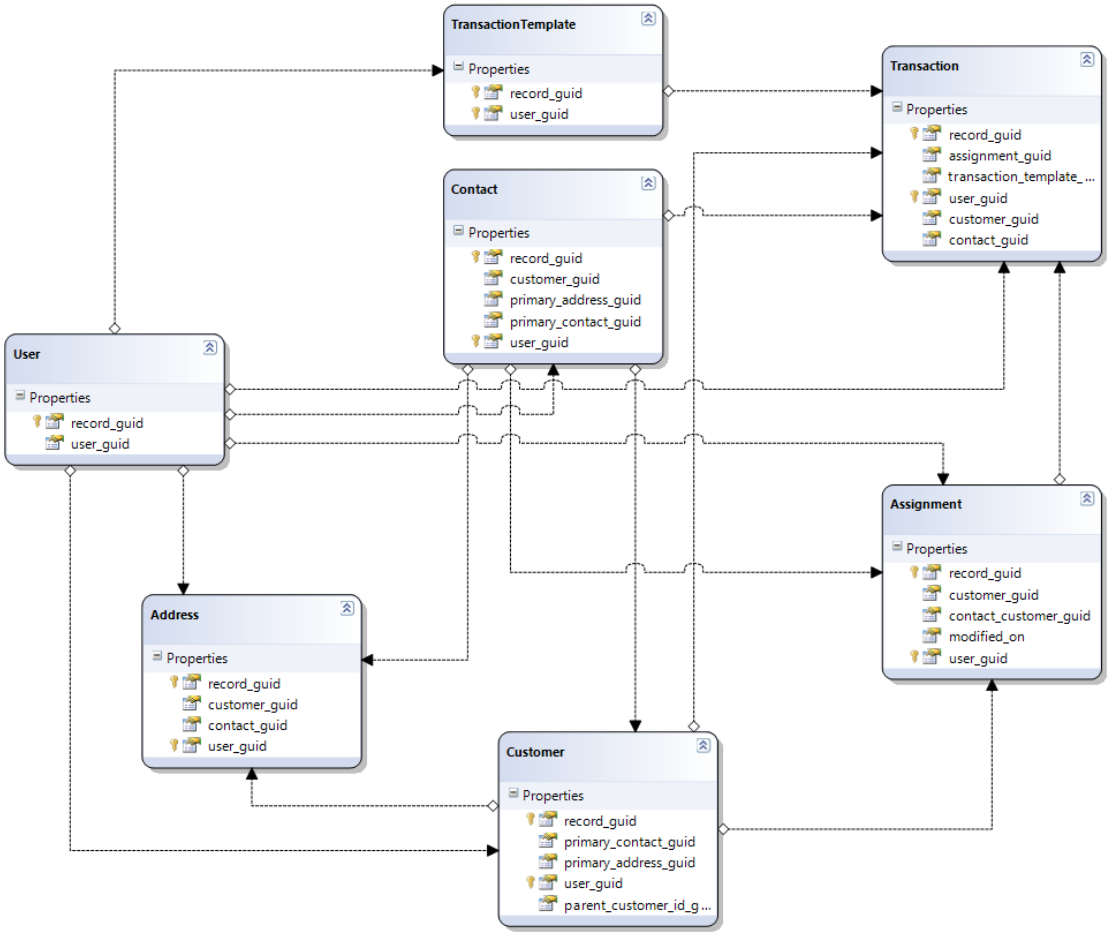


Figure 27: Intermediate database logical model

The figure 27 illustrates the logical database model for intermediate database with primary and foreign keys.

4.4 Synchronization configuration

The sync configuration describes what data CSI Helsinki Oy want to synchronize. Sync configuration is defined as sync scope and sync scope templates. In a sync scope CSI Helsinki Oy define scope name, database name and location, tables and columns CSI Helsinki Oy want to synchronize. It also includes filters CSI Helsinki Oy want to apply

when synchronizing data. Sync scope is a static scope with fixed filter values. Sync scope can also be defined as a scope template which can be used to create sync scopes dynamically at run time. The generated configuration file is used to provision or de-provision the database.

The Sync configuration file is a pure XML file which can be created by hand using any text editor. Creating a sync configuration file manually by hand for each table with filters needs great care and time thus making this job time-consuming. Microsoft Sync Framework provides a good graphical tool called 'Sync Service Utility' which can be used to do this job with the wizard such as interface as illustrated by figure 28. This tool is limited in what it can do, so if developer needs more control over the configuration file, developer can always tweak it afterwards using our favorite text editor.

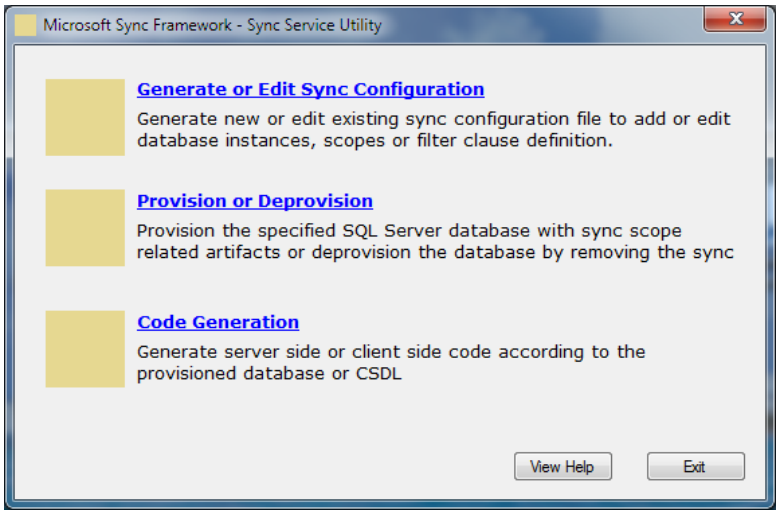


Figure 28: Microsoft Sync Framework - Sync Service Utility

Once the sync configuration file has been created, CSI Helsinki Oy can use same graphical tool or command line utility 'SyncSvcUtil' to provision or de-provision the database. It is also possible to create the sync configuration for provisioning and de-provisioning using the managed code written using .Net Framework.

4.5 Database provisioning

Database provisioning is the process of configuring database for synchronization by using synchronization scope and/or synchronization scope template defined in XML file or in the code. After CSI Helsinki Oy applys the sync scope to the database, Microsoft Sync Framework creates change-tracking and metadata management infrastructure in the database to enable change-tracking. These objects consist of metadata tables, triggers and stored procedures for synced tables to track changes in the table. Once the database has been provision, it can be synchronized with other databases and data stores. Database provisioning is generally done only once unless sync scope has been changed.

Deprovisioning is process of removing all the all the Microsoft Sync Framework management infrastructure created during the provisioning process. Both provisioning and deprovisioning can be performed using the 'Sync Service Utility', command line utility or in the managed .Net code. [23]

In our solution, CSI Helsinki Oy is going to create the sync configuration and provision the CSI Lawyer® database in the .Net managed code. It is necessary because Windows sync client application installed on the user's computer to perform the synchronization must do CSI Lawyer® database provisioning on first sync. CSI Helsinki Oy is going to use parameter-based filter to filter the data thus it is compulsory to create scope template with filters and then filtered scopes based for each user based on the scope template.

As explained in the chapter 4.3, synchronization scope contains information of all the tables and columns CSI Helsinki Oy want to synchronize along with data filters, it would be clumsy to define and manage the whole schema in the code. In this solution, I created a LINQ-to-SQL data context where I defined all the tables and columns I wanted to include in the synchronization configuration. Then in the code, I read the contents of this data context dynamically and created the sync scope with tables, columns and filters on-fly as illustrated in listing 5.

```
// Create a scope description object
var scopeDesc = new DbSyncScopeDescription(ProvissioningTemplateName)
     ;

// Read the tables from data models
var model = new AttributeMappingSource().GetModel(typeof(DataContext)
     );
foreach (var mt in model.GetTables())
{
    var columnsToInclude = new Collection<string>();

    // Get columns from data model table
    foreach (var dm in mt.RowType.DataMembers)
    {
        // Skip associations from data model
        if (!dm.IsAssociation)
            columnsToInclude.Add(dm.MappedName);
    }

    // Cerate scope for table
    var tableScope = SqlSyncDescriptionBuilder.GetDescriptionForTable
     (
        GetStrippedTableName(mt.TableName), columnsToInclude, databas
     e);

    scopeDesc.Tables.Add(tableScope);
}
```

Listing 5: Creating sync configuration on-fly using data context

Once I had scope description ready, I created the database provisioning template in the code by using scope description created in the earlier step and then provision the CSI Lawyer® database as illustrated in listing 6.

```
// Create the provisioning scope template
SqlSyncScopeProvisioning serverTemplate = new SqlSyncScopeProvisionin
     g(database, scopeDesc, SqlSyncScopeProvisioningType.Template);

// Tables already exists in database, we skip the table creation
serverTemplate.SetCreateTableDefault(DbSyncCreationOption.Skip);

// Create or upadte the stored procedures used by Microsoft Sync
     Framework in the dataabse
serverTemplate.SetCreateProceduresForAdditionalScopeDefault(DbSyncCre
     ationOption.CreateOrUseExisting);
// Add table filters
foreach (var mt in model.GetTables())
{
    var tableName = GetStrippedTableName(mt.TableName);

    // Favorite table - no needs to filter anything
    if (tableName.Equals("Favorite", StringComparison.CurrentCultureI
     gnoreCase))
        return;
```

```
    // If user use record_guid to filter the data
    if (tableName.Equals("User", StringComparison.CurrentCultureIgnor
     eCase))
    {
        serverTemplate.Tables[tableName].AddFilterColumn("record_guid
     ");
        serverTemplate.Tables[tableName].FilterClause = "[side].recor
     d_guid

     = @userGuid";
        serverTemplate.Tables[tableName].FilterParameters.Add(
                new SqlParameter("@userId
     ", SqlDbType.UniqueIdentifier));
    }
    // for other records, use SQL method to filter,
    // we have defined more complex business logic
    else
    {
        serverTemplate.Tables[tableName].AddFilterColumn("record_guid
     ");
        serverTemplate.Tables[tableName].FilterClause = string.Format
     (

     "dbo.FilterData([side].record_guid,@userGuid,'{0}') = 1", tableNa
     me);
        serverTemplate.Tables[tableName].FilterParameters.Add(

     new SqlParameter("@userId", SqlDbType.UniqueIdentifier));
    }
}
// Finally check weather template already exists in the database, if
     no create it
if (!serverTemplate.TemplateExists(ProvissioningTemplateName))
    serverTemplate.Apply();

database.Dispose();
```

Listing 6: Creating sync scope template

Once I have provision the CSI Lawyer® database with sync template using the code above, I see additional database tables created by the Microsoft Sync Framework as illustrated by figure 29.

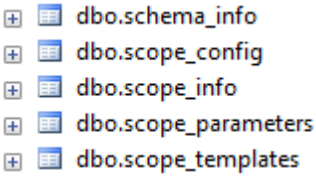


Figure 29: Microsoft Sync Framework - Management tables

Each management table created by Microsoft Sync Framework is described in table 4. These tables are used to store the metadata information needed by Microsoft Sync Framework.

Table Name	Description
schema_info	Contains the version information
scope_config	Contains configuration parameters for each scope. e.g. tables, columns, filters
scope_info	scope name and knowledge information for each scope
scope_parameters	filter parameter values for each scope
scope_templates	scope template name and description for each scope

Table 4: Microsoft Sync Framework - Management table descriptions

Microsoft Sync Framework also creates the tracking table and three triggers for each table included in scope as illustrated by figure 30.

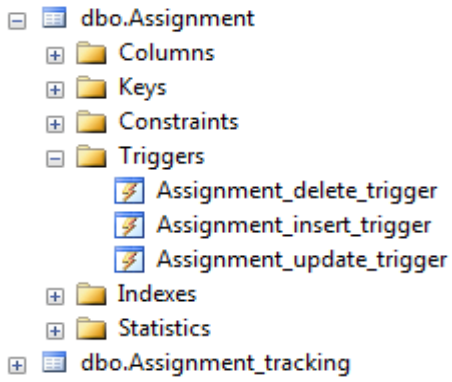


Figure 30: Microsoft Sync Framework - Triggers and Tracking tables

These three Insert, Update and Delete triggers are responsible for keeping the corresponding table's tracking table information up to date for every insert, update and delete operation. The tracking table contains the change tracking information for the rows in the tables that are part of the sync scope. The tracking table contains also all the columns defined in the data filters in the sync scope.

Microsoft Sync Framework also creates 10 stored procedures for each table to enumerate and apply changes to the table. It creates also stored procedures for each table that is a part of sync scope as illustrated by figure 31.

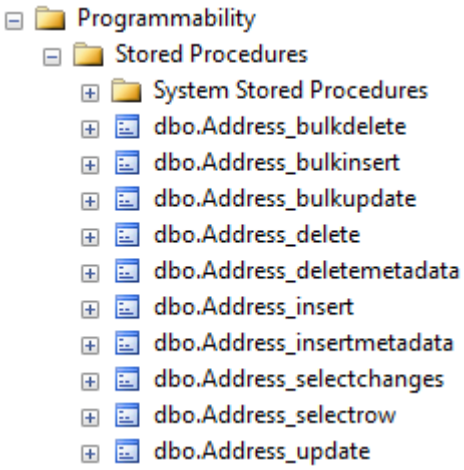


Figure 31: Microsoft Sync Framework - Stored procedures

For SQL 2008 or SQL Azure databases, Microsoft Sync Framework creates bulk procedures with table-valued parameters that contain all changes to be applied to the table. This significantly reduces the return trip required to apply each row to the target table.

Microsoft Sync Framework also creates the user-defined table data type for each table defined in sync scope during the provisioning process as illustrated by figure 32. These user-defined table types are required by the bulk procedures table-valued parameter.

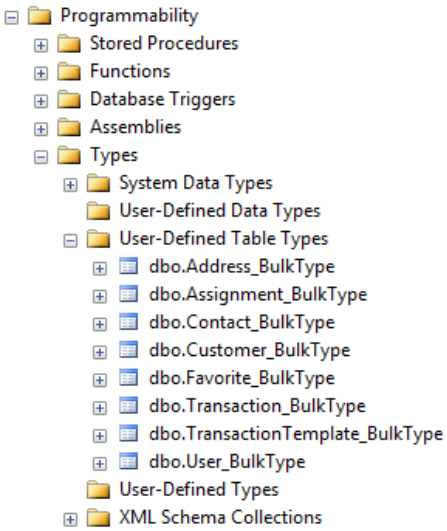


Figure 32: Microsoft Sync Framework – User-defined table types

The standard 'selectchanges' stored procedure was not enough for us to achieve the expected functionality required by favorite assignment as this only selects the changes for each record as added or removed when it is physically added or removed from the database. In project scenario, I wanted the Microsoft Sync Framework to treat all assignments and its related data as deleted when user removes it from favorite and as created when the user adds it to favorite. To achieve the expected functionality, I modified the 'selectchanges' stored procedure for each tracked table with custom business logic. For example the original 'selectchanges' stored procedure uses 'sync_row_is_tombstone' column from tracking table to identify whether the specific row has been deleted from database. In the modified stored procedure CSI HELSINKI OY call the SQL method to determine whether the row should be treated as deleted or not when it is not physically deleted from the database. A listing 7 illustrates a small piece of modified 'selectedchanges' stored procedure.

```
(CASE WHEN [side].[sync_row_is_tombstone] = 1 THEN 1 ELSE CASE WHEN d
     bo.FilterData([side].record_guid,@userGuid,'{0}') = 0 THEN 1 ELSE
      0 END END) as [sync_row_is_tombstone],
```

Listing 7: Microsoft Sync Framework - Modified 'selectedchanges' stored procedure

The Intermediate database needs to be provisioned only once thus there is no need of any code. Because I am going to use user id to filter the data from the database, sync template is required. I will use 'Sync Service Utility' to create the sync template and provision the database.

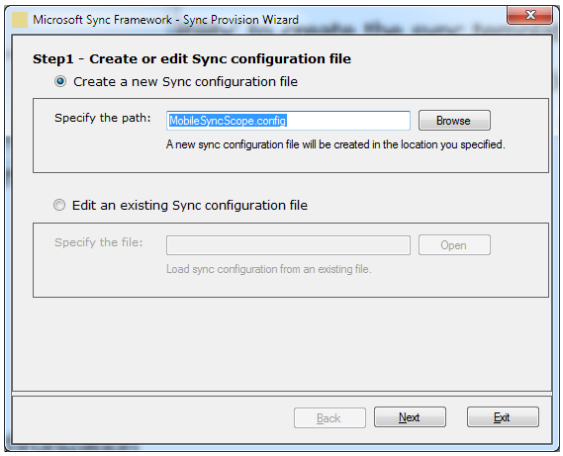


Figure 33: Sync Service Utility - Creating sync configuration file tool step 1

First I needed to create the sync template configuration by opening the 'Generate or Edit sync configuration' wizard from the 'Sync Service Utility' and defining the configuration file name as illustrated by figure 33.

In the following next two screens, I defined the database and sync scope information as illustrated by figure 34.

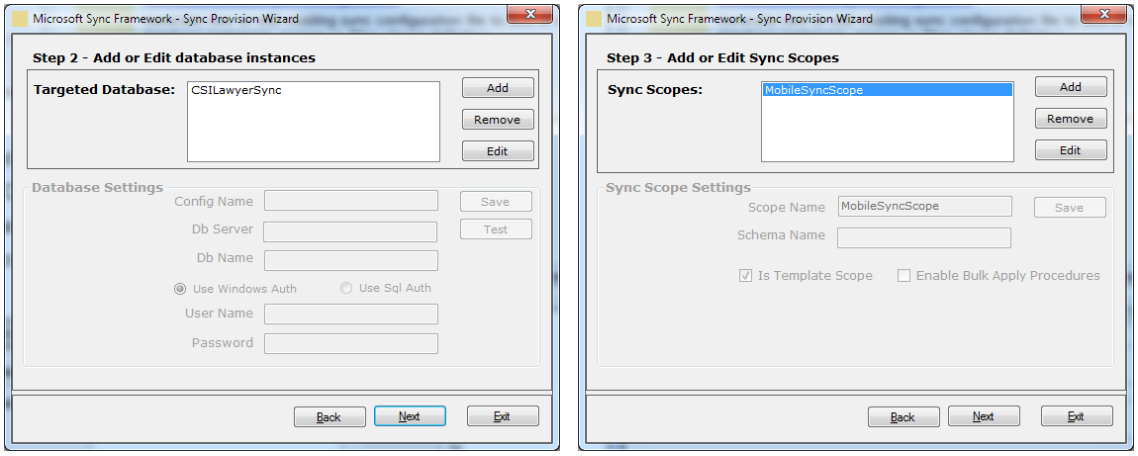


Figure 34: Sync Service Utility - Creating sync configuration file tool step 2-3

In the next screen, I choose the tables, columns and filters I want to include in the synchronization scope template as illustrated by figure 35.

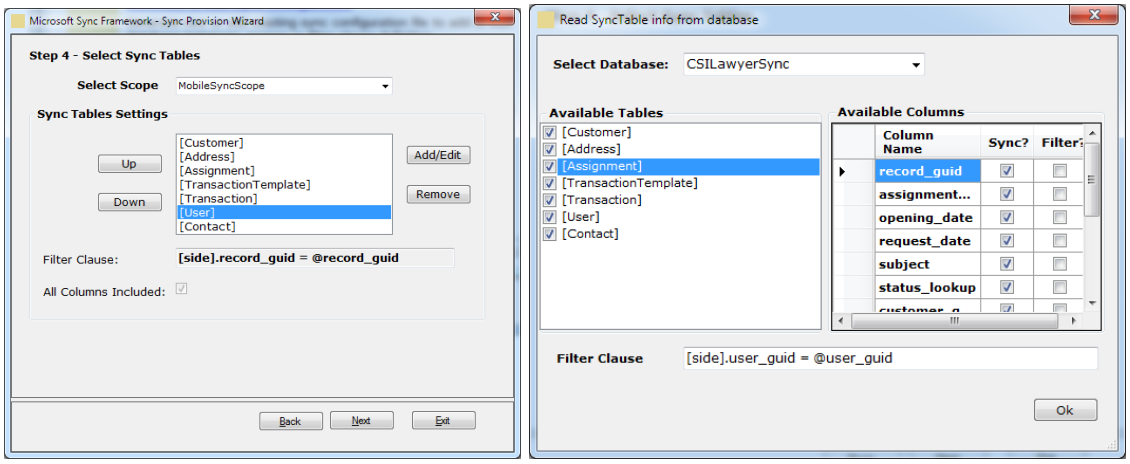


Figure 35: Sync Service Utility - Creating sync configuration file tool step 4

The next screen shows the XML contents of configuration file as illustrated by figure 36. Now I have the configuration file ready.

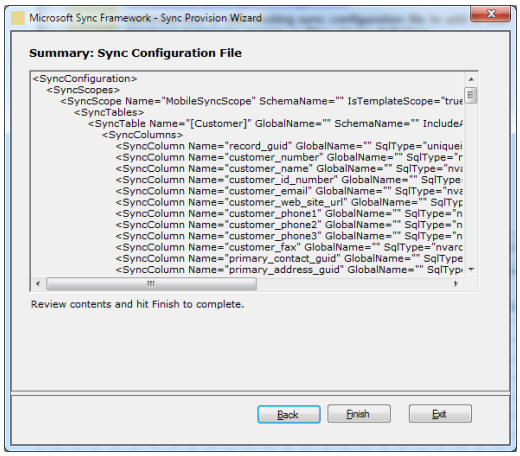


Figure 36: Sync Service Utility - Creating sync configuration file tool step 5

To provision the intermediate database, use the 'Sync Provision Wizard' from 'Sync Service Utility' with configuration file generated during the previous steps as illustrated by figure 37.

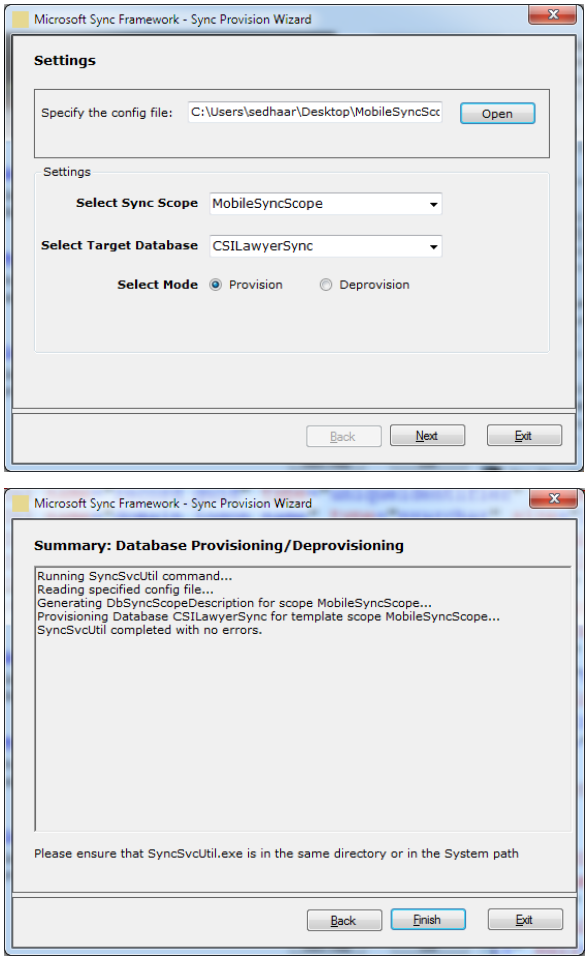


Figure 37: Sync Service Utility - Provisioning the database tool

4.6 CSI Windows sync client

CSI Windows sync client is a Windows based application developed using Windows Presentation Foundation and .Net 4 Framework. It runs in the background in the system tray and performs the actual data synchronization between CSI Lawyer® and the intermediate database. CSI Windows sync client application implements the business logic to modify the data before submitting it to CSI Lawyer® and intermediate database. The figure 38 illustrates the Microsoft Sync Framework architecture CSI Windows sync client application. Green components are the ones provided by Microsoft Sync Framework and the orange component is a custom code which has been developed by CSI Helsinki Oy.

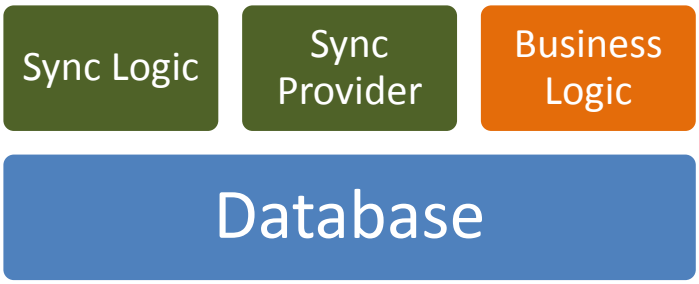


Figure 38: CSI Windows sync client - Architecture

After installing the CSI Windows sync client application on the computer, the user needs to set the email, password and sync interval information in the settings of the application as illustrated by figure 39. The application verifies the validity of username and password; if correct, the sync client gets activated and starts synchronizing the data in user-defined sync time intervals.

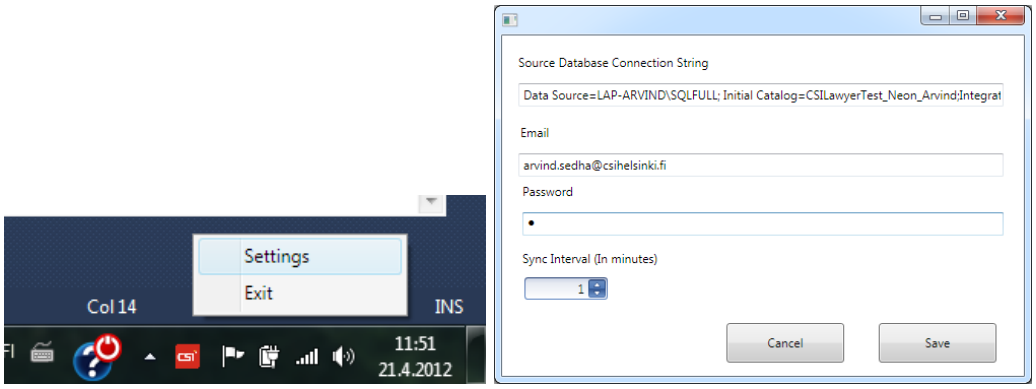


Figure 39: CSI Windows sync client - Settings

Every run of synchronization as well as the synchronization results are informed to user using the system tray pop-up as illustrated by figure 40.

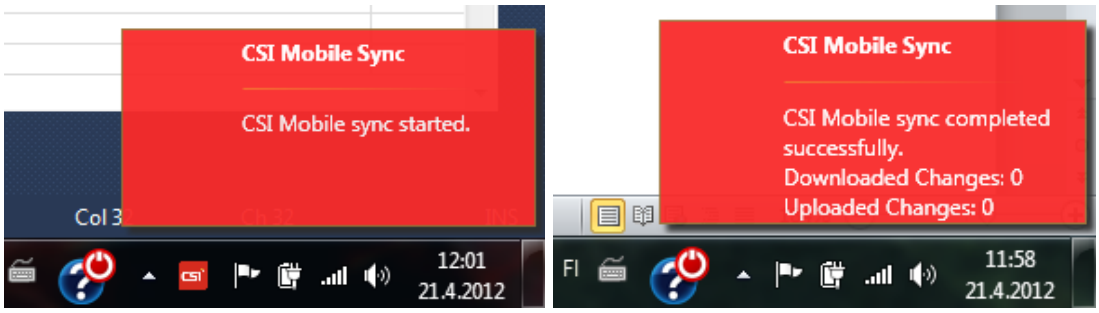


Figure 40: CSI Windows sync client - Sync notifications

The process of initiating and performing actual synchronization after both databases have been provisioned is simple. Microsoft Sync Framework API contains a class named SyncOrchestrator where is defined the local and remote sync providers participating in synchronization as well as the direction of data synchronization as illustrated by listing 8. It starts and controls the synchronization session, and dispatches progress events to the application. [24]

```
// Initiate the SQL connections
SqlConnection sourceDatabase = new SqlConnection(SourceSqlDatabaseCon
     nectinString);
SqlConnection destinationDatabase = new SqlConnection(DestinationSqlD
     atabaseConnectinString);

// Initiate local database provider
var localProvider = new SqlSyncProvider(ProvissioningScopeName, sourc
     eDatabase);
localProvider.ChangesSelected += localProvider_ChangesSelected;

// Initiate remote database provider
var remoteProvider = new SqlSyncProvider(ProvissioningScopeName, dest
     inationDatabase);
remoteProvider.ChangesSelected += remoteProvider_ChangesSelected;

// Initiate the sync orchestrator
SyncOrchestrator syncOrchestrator = new SyncOrchestrator()
{
    LocalProvider = localProvider,
    RemoteProvider = remoteProvider,
    Direction = SyncDirectionOrder.UploadAndDownload // Direction of
     synchronization
};
```

```
// Perform the synchronization
var stats = syncOrchestrator.Synchronize();

// Dispose the connections to databases
sourceDatabase.Dispose();
destinationDatabase.Dispose();
```

Listing 8: Performing data synchronization

The code illustrated in listing 8 uses SqlSyncProvider class which is part of Microsoft Sync Framework. It exposes many different events which can be listened to perform business logic and error handling during the synchronization operation. Currently I am interested in listening ChangesSelected event for both databases sync providers. I used ChangesSelected event to modify data and data structure before it is submitted to CSI Lawyer® or intermediate database as illustrated in listing 8.

```
void localProvider_ChangesSelected(object sender, DbChangesSelectedEv
     entArgs e)
{
    // Remove the favorite table from changes as it's not required
    // by intermediate database
    e.Context.DataSet.Tables.Remove("Favorite");

    // Get's the database model used for synchronizatio scope
    var model = new AttributeMappingSource().GetModel(typeof(DataCont
     ext));
    foreach (var mt in model.GetTables())
    {
        var dataTable = e.Context.DataSet.Tables

     [ProvisionHelper.GetStrippedTableName(mt.TableName)];
        if (dataTable.Columns.Contains("modified_by"))
        {
            // Replace the modified_by column name to user_guid
            dataTable.Columns["modified by"].ColumnName = "user guid"
     ;
            foreach (DataRow row in dataTable.Rows)
            {
                // Assign current user id to each row for user_guid c
     olumn
                if (row.RowState != DataRowState.Deleted)
                    row["user_guid"] = UserGuid;
            }
        }
    }
}
```

Listing 9: Business logic for 'SelectedChanges' event of CSI Lawyer® database

As the intermediate database contains the column user_guid for each table which does not exists on CSI Lawyer® database, CSI Helsinki Oy use ChangesSelected event of

local provider to modify the tables' structure and data in a way that is acceptable by intermediate database. CSI Helsinki Oy also drop "Favorite" table from the selected changes data set as it is not required by intermediate database.

```
void remoteProvider_ChangesSelected(object sender, DbChangesSelectedE
     ventArgs e)
{
    // Get's the database model used for synchronization scope
    var model = new AttributeMappingSource().GetModel(typeof(tDataCon
     text));
    foreach (var mt in model.GetTables())
    {
        // Replace the user_guid column name to modified_by for each
     table
        var dataTable = e.Context.DataSet.Tables

     [ProvisionHelper.GetStrippedTableName(mt.TableName)];
        if (dataTable.Columns.Contains("user_guid"))
            dataTable.Columns["user_guid"].ColumnName = "modified_by"
     ;

        // Check if there is any transactions
        var tableName = ProvisionHelper.GetStrippedTableName(mt.Table
     Name);
        if (tableName.Equals("Transaction", StringComparison.CurrentC
     ultureIgnoreCase))
        {
            // Copy the created and modified transactions data rows t
     o list
            List<DataRow> newTransactions = new List<DataRow>();
            foreach (DataRow row in dataTable.Rows)
            {
                if (row.RowState == DataRowState.Added
                     || row.RowState == DataRowState.Modified)
                    newTransactions.Add(row);
            }

            // Create the transaction using business logic
            // and remove transaction from sync table
            TransactionHandler tHandler = new TransactionHandler(m_Cs
     iDataContext);
            foreach (var dr in newTransactions)
            {
                Transaction transaction = CreateTransactionObject(dr)
     ;
                if (dr.RowState == DataRowState.Added)
                    tHandler.Create(transaction);
                else if (dr.RowState == DataRowState.Modified)
                        tHandler.Update(transaction);
                dataTable.Rows.Remove(dr);
            }
        }
    }
}
```

Listing 10: Business logic for SelectedChanges event of intermediate database

The table' structure and data received from intermediate database before committing it to CSI Lawyer® database. The code illustrated in listing 9 shows the procedure to modify and process the data using business logic after changes have been selected from intermediate database for committing to CSI Lawyer® database.

4.7 CSI Azure sync service

Direct synchronization using the SQL providers is an appropriate solution for synchronizing data between CSI Lawyer® and intermediate database but it cannot be used to synchronize the data between intermediate database and Windows Phone®. Windows Phone® API does not support direct access to any kind of database, thus leaving only few choices to expose the data from intermediate database to Windows Phone® such as sockets, web technologies and protocols.

Microsoft Sync Framework provides out-of-box libraries to expose the data to any kind of devices using OData protocol. Since OData is an open standard, different kinds of platforms such as PHO, .Net, Java are able to sync data using the Microsoft Sync Framework. CSI Helsinki Oy will use these libraries to expose the data from intermediate database to Windows Phone® in CSI Azure sync service. To make the communication between Windows Phone® and CSI Azure sync service more secure, I used custom authorization business logic as well as transport level data encryption using SSL certificates. The figure 41 illustrates the CSI Azure sync service architecture.

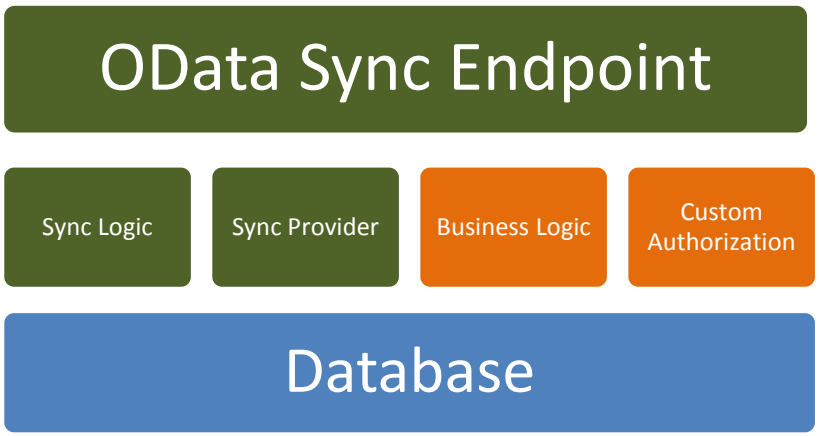


Figure 41: CSI Azure sync service – Architecture

The CSI Azure sync service consists of Azure web role, OData end point and proxy classes created manually or generated automatically by using the 'Sync Service Utility' tool. I have used the 'Sync Service Utility' tool to generate the necessary classes which consist of proxy classes for each table and OData service endpoint as illustrated by figure 42.

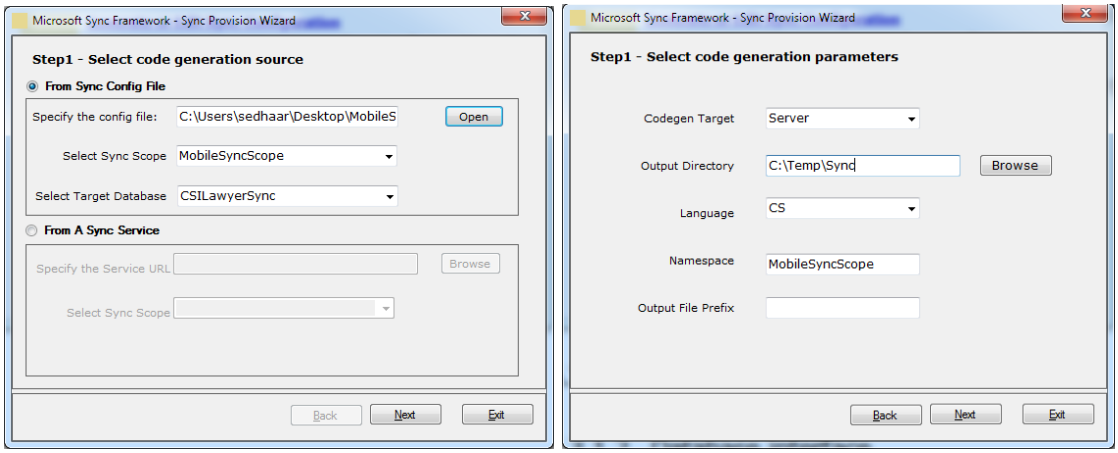


Figure 42: CSI Azure Sync Service - Endpoint code generation

The code generation wizard generates many classes organized in three different files. These generated files will be used in our Azure web role project. These classes are required by the Microsoft Sync Framework to expose data from database using OData protocol over the web. The table 5 describes what kind of classes each of these generated files includes.

File Name	Description
ScopeNameEntities.cs	Contains the classes for each table with every column included in the sync scope. All these classes implement the IOfflineEntity interface.
ScopeNameSyncService.svc	OData endpoint markup description
ScopeNameSyncService.svc.cs	OData endpoint code file.

Table 5: Microsoft Sync Framework - OData endpoint generated files

If I use these filters in a sync scope, I need to define them in the InitializeService static method in the ScopeNameSyncService.svc.cs file. In our scenario CSI Helsinki Oy need to define the filters for each table in InitializeService as shown in the code below.

```
public static void InitializeService(ISyncServiceConfiguration config
     )
{
    // Define connection string
    config.ServerConnectionString = ConfigurationManager.ConnectionSt
     rings

     ["IntermediateDatabaseConnectionString"].ToString();

    // Set the scope used by sync service
    config.SetEnableScope("MobileSyncScope");

    // Initiate the filters for each table
    config.AddFilterParameterConfiguration("userId", "User",

     "@record_guid", typeof(Guid));
    config.AddFilterParameterConfiguration("userId", "Assignment",

     "@user_guid", typeof(Guid));
    config.AddFilterParameterConfiguration("userId ", "Contact",

     "@user_guid", typeof(Guid));
    config.AddFilterParameterConfiguration("userId", "Customer",

     "@user_guid", typeof(Guid));
    config.AddFilterParameterConfiguration("userId", "Address",

     "@user_guid", typeof(Guid));
    config.AddFilterParameterConfiguration("userId", "TransactionTemp
     late",

     "@user_guid", typeof(Guid));
    config.AddFilterParameterConfiguration("userId", "Transaction",

     "@user_guid", typeof(Guid));

    // Set data format and conflict resolution
    config.SetDefaultSyncSerializationFormat(SyncSerializationFormat.
     ODataJson);
    config.SetConflictResolutionPolicy(ConflictResolutionPolicy.Serve
     rWins);

    // Testing related code
    config.EnableDiagnosticPage = true;
}
```

Listing 11: Initializing CSI Sync Service

At this point OData sync service endpoint is up and running with default configuration, but still it cannot be accessed by the Windows Phone® Silverlight application. OData sync service is called using REST requests and Windows Phone® Silverlight application cannot access it due to cross domain access issue. In order to make our OData endpoint available to Windows Phone® Silverlight application, CSI Helsinki Oy need to

add the clientaccesspolicy.xml or crossdomain.xml with valid configuration at the root of the application. I used clientaccesspolicy.xml for CSI Azure sync service with support for both HTTP and HTTPS protocols as illustrated in listing 12. [25]

```
<?xml version="1.0" encoding="utf-8" ?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="http://*"/>
        <domain uri="https://*"/>
      </allow-from>
      <grant-to>
        <resource include-subpaths="true" path="/"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Listing 12: Crossdomain.xml for CSI sync service

Microsoft Sync Framework sync service OData endpoint only supports communication over HTTP and does not have any authorization mechanism out of the box. To add HTTPS support, I defined the binding configuration in the web.xml file of web application for the sync service's OData endpoint. webHttpBinding type is the right binding type for CSI Azure sync service as it allows communication using REST calls. The endpoint contact used in the endpoint must be "Microsoft.Synchronization .Services.IRequestHandler". The final binding configuration in the web.xml file is illustrated in listing 13. The important parts are highlighted in bold.

```
 <system.serviceModel>
    <serviceHostingEnvironment multipleSiteBindingsEnabled="true" />
    <services>
      <service name="Csi.Web.SyncService.MobileSyncService">
        <endpoint address=""  binding="webHttpBinding"

     bindingConfiguration="higherMessageSize"
                  contract="Microsoft.Synchronization.Services.IReque
     stHandler"
                  behaviorConfiguration="web"/>
      </service>
    </services>
    <bindings>
      <webHttpBinding>
        <binding name="higherMessageSize" maxReceivedMessageSize="900
     00000" >
          <security mode="Transport">
            <transport clientCredentialType="None" />
```

```
        </security>
      </binding>
     </webHttpBinding>
    </bindings>
    <behaviors>
      <endpointBehaviors>
        <behavior name="web">
          <webHttp />
        </behavior>
      </endpointBehaviors>
      <serviceBehaviors>
        <behavior name="">
          <serviceMetadata httpsGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
```

Listing 13: CSI sync service endpoint configuration

To handle the user authorization during each request made to sync service based on username and password, the CSI Azure sync service uses the custom Http module. The custom authentication module inherits from IHttpModule interface. It verifies the received user credentials using the business logic for each sync request. If the user authorization fails, the data synchronization is cancelled immediately. To enable the custom HTTP module, you need to do it in the modules section of web.config as illustrated in listing 14.

```
<system.webServer>
    <modules>
      <add name="CustomAuthenticationModule"
     type="Csi.Web.SyncService.BL.CustomAuthenticationModule,Csi.Web.S
     yncService"/>
    </modules>
</system.webServer>
```

Listing 14: Enables custom authentication module

5 Windows Phone® client development

5.1 Overview

The CSI Lawyer® Windows Phone® application will be called CSI Mobile®. CSI Mobile® is a Windows Phone® occasionally connected client application for CSI Lawyer® designed for people who spend most of their time working out of the office. It allows users to create and edit transactions as well as view customer, contact and assignment information using their Windows Phone®. The application makes use of Microsoft Sync Framework to synchronize the data between local data store and intermediate database using sync service's OData endpoint over the Internet. The application has been designed keeping usability and flexibility in mind, so that the application remains responsive, pages and controls are consistent, as well as follows the design guidelines recommended by Microsoft.

5.2 Application architecture

CSI Mobile® application data is stored on Windows Phone® in the isolated storage. Currently there is no Microsoft Sync Framework compact database storage provider available for Windows Phone® and CSI Helsinki Oy either did not want to spend time implementing it. Instead CSI Helsinki Oy will use the isolated storage provider currently included in Microsoft Sync Framework to store the data on the Windows Phone® isolated storage. The Isolated storage provider supports exchanging of synchronization messages using the OData Sync protocol.

CSI Mobile® uses data context to retrieve and stored data in the isolated storage. The data context needed by Microsoft Sync Framework to interact with the isolated storage and OData protocol must be inherited from IsolatedStorageOfflineContext class**.** The IsolatedStorageOfflineContext class exposes properties, methods, and events required for synchronization process. This includes methods to initiate a session, add and remove items, and initiate synchronization; properties that expose collections of errors and conflicts; and events that indicate the progress and result of the process. [26]

All the entities exposed by isolated storage offline context must inherit from IsolatedStorageOfflineEntity base class. This base class exposes properties used by the Microsoft Sync Framework provider to define and monitor changes to the local data. It also exposes events that indicate when the entity value changes and a method to reject the changes and restore the original version. The figure 43 illustrates the CSI Mobile® application architecture. [26]

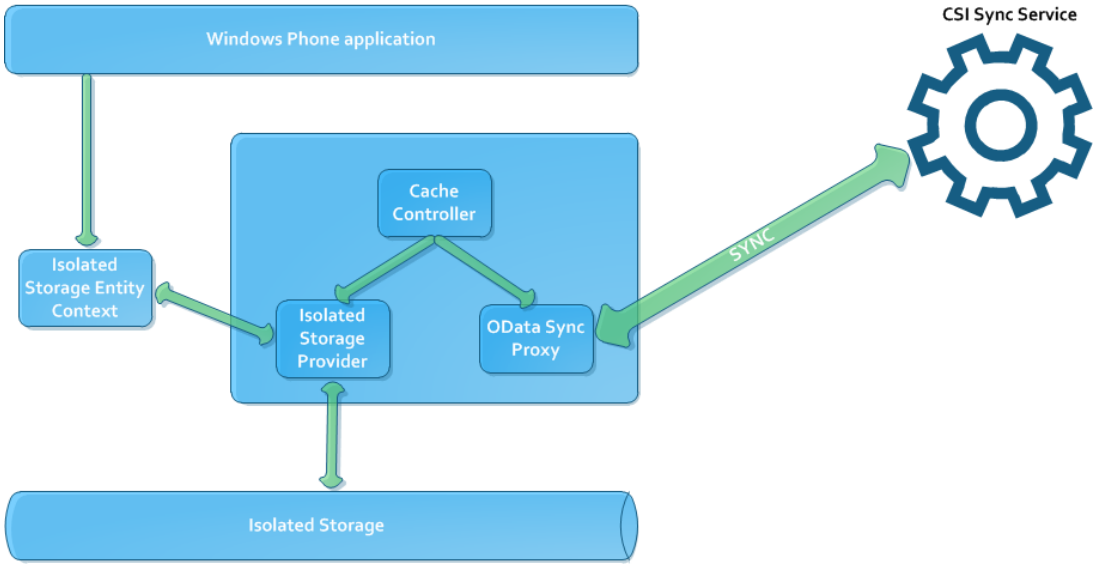


Figure 43: CSI Mobile® - Application architecture

CSI Helsinki Oy can write this entity classes and data context ourselves as long as they fulfill the requirements necessary by Microsoft Sync Framework isolated storage provider. The easiest way to create these classes is by using 'Sync Service Utility' provided with Microsoft Sync Framework. It is the same utility used in chapter 4 and sections 4.3 and 4.4 to generate the code for sync service code.

5.3 User interface design

The user interface of the CSI Mobile® was designed completely in harmony with Windows Phone® design guidelines. I had used Microsoft Blend to design and customize the user interface of CSI Mobile® application. Microsoft Blend allows creating and using the design time data source for every component supporting data binding which has been used in designing the CSI Mobile® application user interface in Microsoft Blend.

5.3.1 User interface components

Windows Phone® SDK comes with many standard controls which are used in most of the user interface of CSI Mobile® applications. However, Windows Phone® SDK is missing many of the important controls. Microsoft has released an open source library of controls called Silverlight for Windows Phone® Toolkit which includes most of the missing controls from standard Windows Phone® SDK.

There are also many other commercial and open source Windows Phone® control libraries which are available on the Internet which might be useful in many scenarios. Apart from standard SDK and Silverlight Toolkit controls, I have also used Coding4Fun library controls in CSI Lawyer® development. For example I have used 'Toast Prompt' from Coding4Fun library to inform the user when background agents are enabled or disabled. Despite having the standard components provided by these libraries, I created three custom controls. These three custom controls are ProgressPopup control, Login Control and DefaultText Control.

Login control is one of the three custom controls I created for this project as illustrated by figure 44.

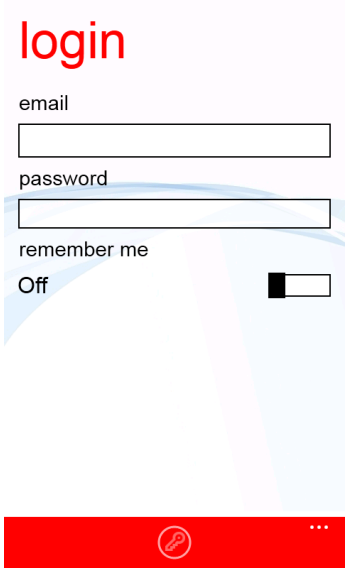


Figure 44: CSI Mobile® - Login control

CSI Lawyer® is an enterprise application and thus it requires that the data is protected by some kind of authentication. Login controls validates the credentials provided by users against the intermediate database by sending credential information to sync service.

The second control is the progress popup control as illustrated by figure 45. It shows the progress bar with text as popup over any other control thus disallowing user interaction with the application while something critical is in progress. The progress popup control is used on pages where some action might take longer time. See appendix 1 for code.

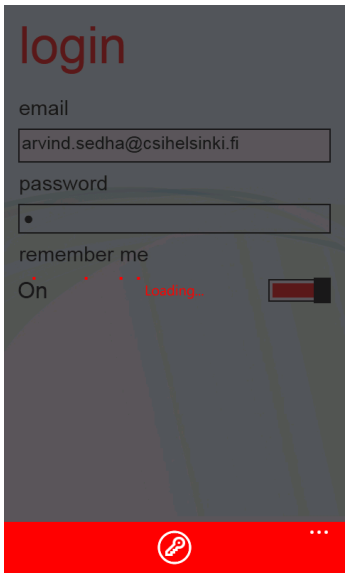


Figure 45: CSI Mobile® - Loading popup control

The third control is DefaultTextbox control. The DefaultTextbox control inherits from TextBox control. It shows the default text in the textbox predefined in the code. The default test is cleared as soon as it gets the focus. The DefaultTextbox control is illustrated by figure 46. See appendix 2 for code.

Figure 46: CSI Mobile® - Default textbox control

CSI Mobile® reflects the colors and styles of CSI Helsinki Oy logo. All the pages are using a custom background image with a white background. Windows Phone® default theming is does not affect any of the CSI Mobile® application pages and components except live tiles. Due to the white background, all the controls placed everywhere use custom styles including panorama, pivot item, textbox, text block, lists and application bar as illustrated by figure 47. All the user-defined styles for components used in Windows Phone® application are stored globally in app.config file. The styles defined globally in app.config can be used anywhere on any page.

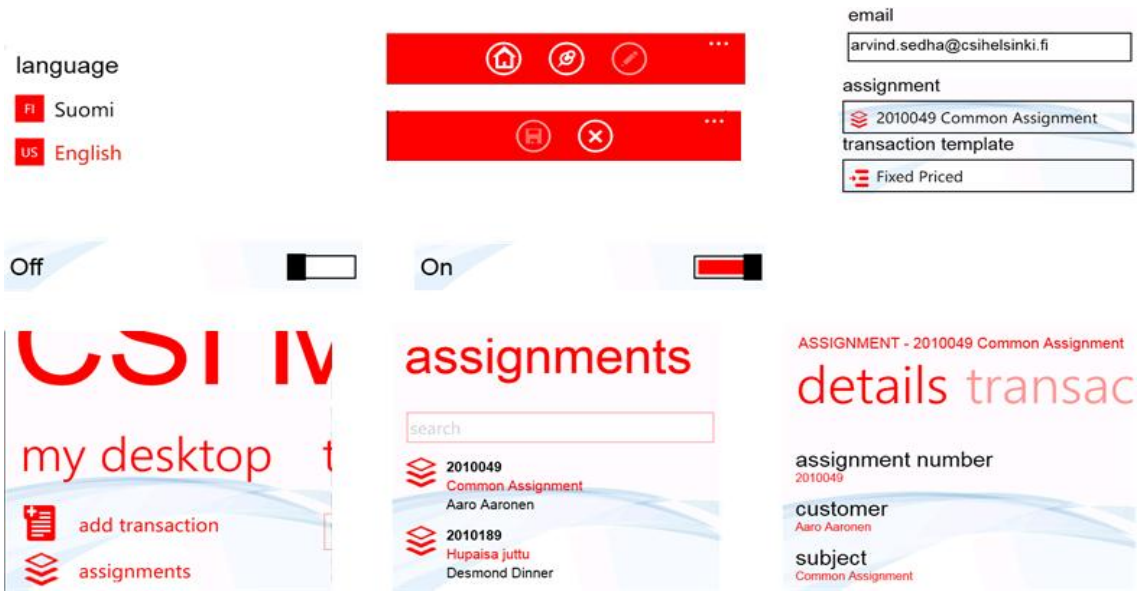


Figure 47: CSI Mobile® - User interface control styles

Aside from the user defined styles to keep consistency between different pages and controls, I also use custom tilt transitions for better user response. I used Peter Torr's tilt effect classes to activate the tilt effect on every clickable item on every page automatically. [27]

5.3.2 Page navigation

CSI Mobile® uses only a small number of pages, with a limited number of navigation routes between those pages as illustrated by figure 48. However, it also supports deep linked pinning Tiles to Start. A Tile is a link to an application displayed in Start. An Application Tile can be pinned to Start, which when tapped by the user will launch the application. Users can also pin a specific entity as a secondary Tile to Start.

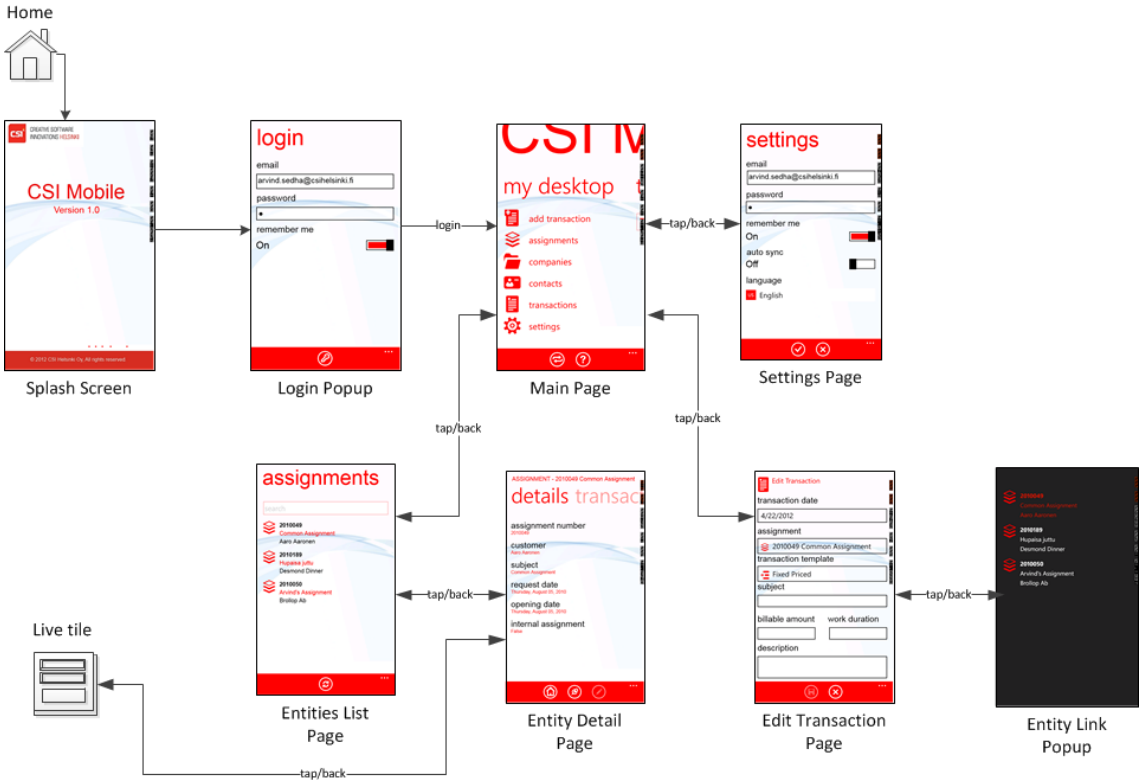


Figure 48: CSI Mobile® - Application page navigation

When a secondary Tile is tapped by the user, the application is launched and the entity detail page is displayed. Secondary tiles offer the user quick and easy access to different entities of the application. Splash screen and Login are special kinds of pages

which did not stay on the application back stack. It means when the user presses the back button on the application from the main page, the user wants to exist the application rather than navigating to the login or splash screen page. To keep away the splash screen and login page, I used the Popup control to display these pages contents.

5.4 Application pages and components

5.4.1 Splash screen

The Splash screen was implemented as a user control, which is displayed immediately as a popup when main page instance is created. In the code I used the background worker to show the splash screen for a specific amount of time. When the main page instance is created, application activates the splash screen popup and then run background worker asynchronously. This creates the secondary thread where we can perform some lengthy operation and main thread still stays responsive as illustrated by figure 49.

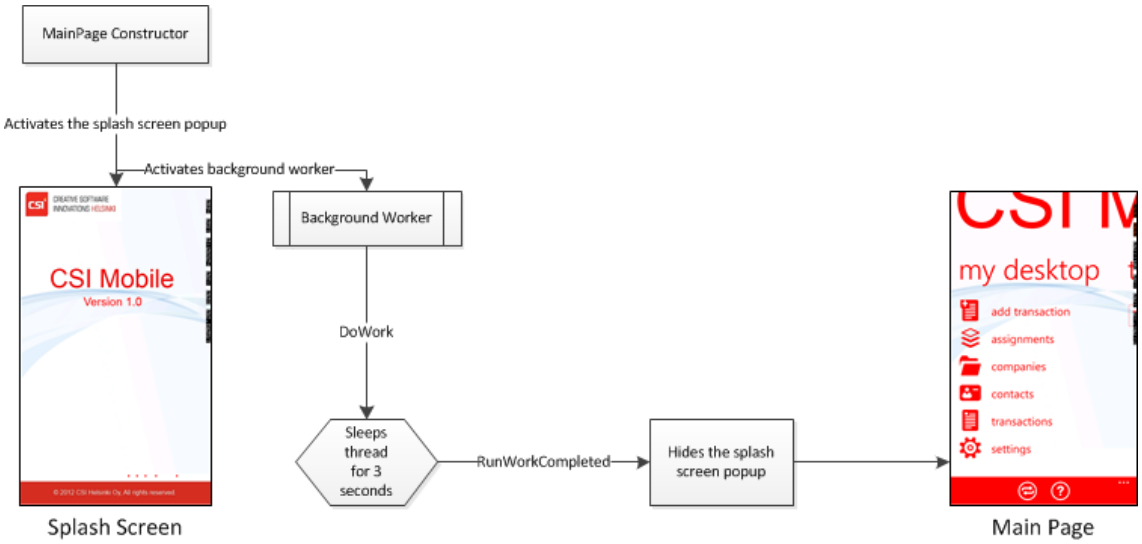


Figure 49: CSI Mobile® - Splash screen displaying process

In the background worker's **dowork** event we make the thread.sleep() method to make it sleep for the time we want splash screen to be visible. Finally when

background worker completes the work, it calls the runWorkerCompletedEvent where we hide the splash screen popup. See appendix 3 for the code.

5.4.2 Login dialog

The logic dialog uses a custom login control. It is displayed immediately after the splash screen as popup dialog on first run of application or if the user has "remember me" option disabled in the settings of the application. Login control validates the credential information entered by the user locally by validating email and password length while the user is filling the credential details. When control accepts the entered credential information, login button gets activated as illustrated by figure 50.

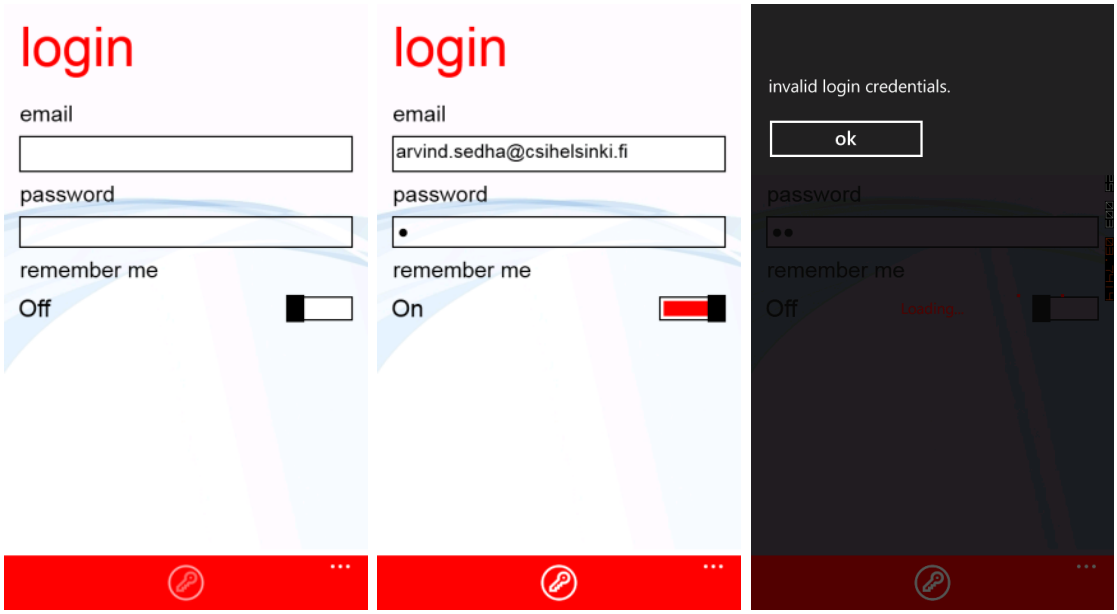


Figure 50: CSI Mobile® - Login dialog

When the user clicks the login button, the login control encrypts the user credentials using Advanced Encryption Standard (AES) symmetric algorithm and sends it to the sync service login module for authentication. WebClient class is used to send the GET request with the user credentials. If the user credentials are valid, the sync service's login module returns authorization token which can be used in data synchronization process. After receiving a valid authorization token, the login control saves the user

credentials and authorization token to an isolated storage and navigates the user to main page. If credentials are invalid, an appropriate message is displayed to the user.

5.4.3 Settings page

The Settings page allows the user to change application related settings such as the user credentials, activation or deactivation of background agents, and application language. After changing settings, the user can accept or reject the changes using buttons from the application bar as illustrated by figure 51.

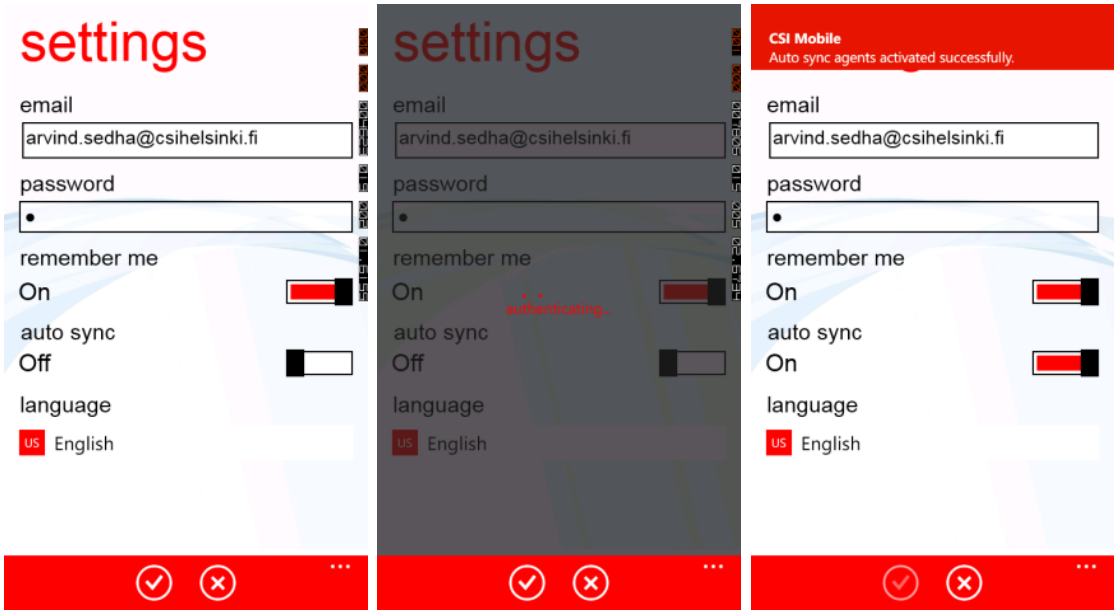


Figure 51: CSI Mobile® - Settings page

For language setting I used special mechanism to reflect the language change immediately as illustrated by listing 15 and listing 16. All the language resources are bound using a special proxy class called LocalizedResources which implements the INotifyPropertyChanged interface.

```
<TextBlock Margin="21,0,24,0" TextWrapping="Wrap"
Text="{Binding LocalizedResources.email, Source={StaticResource Local
     izedStrings}}"
VerticalAlignment="Stretch" Style="{StaticResource CsiLabel}"/>
```

Listing 15: Language resource binding

When a user changes the language and saves the settings, we change the culture information of the thread immediately and then call ResetResources() method from LocalizedStrings object to update the user interface. Please see the appendix 4 for implementation of LocalizedStrings class.

```
// Change the language
CultureInfo newCulture = new CultureInfo(AppSettings.Instance.Languag
     eSetting);
Thread.CurrentThread.CurrentCulture = newCulture;
Thread.CurrentThread.CurrentUICulture = newCulture;

// Reset resources
((LocalizedStrings)App.Current.Resources["LocalizedStrings"]).ResetRe
     sources();
```

Listing 16: Switching UI language

The Settings page also displays the toast message for successful activation or deactivation of background agents if the user has changes the auto sync setting.

5.4.4 Main page

The Main page is the homepage of the application which uses the panorama control. The panorama control of the main page has two panorama items. The first panorama item is called 'my desktop' where we have placed links to common actions required by the user to quickly view or change information in the application as illustrated by figure 52. The actions list uses the listbox control to display the items which are initiated dynamically in the code.

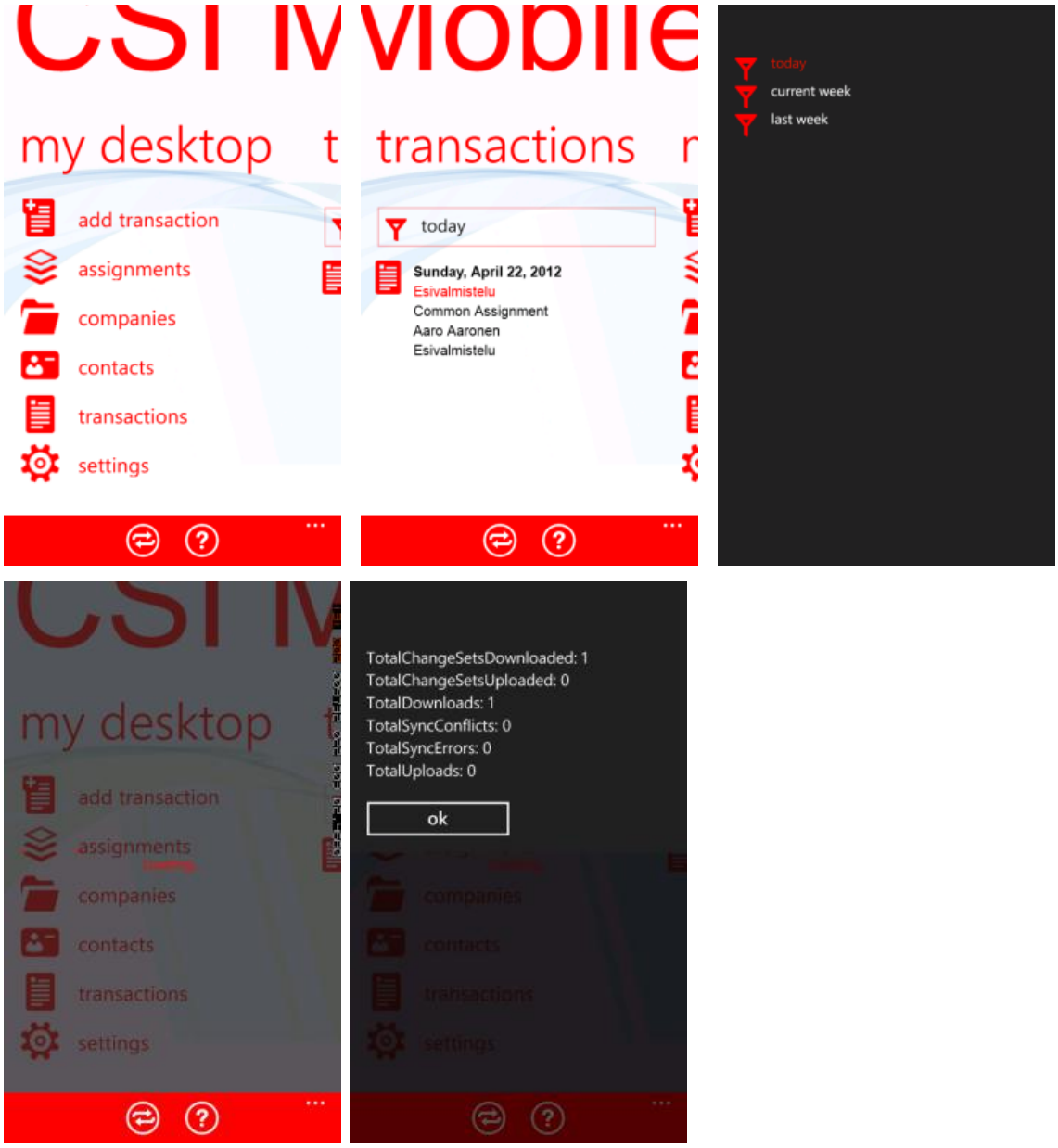


Figure 52: CSI Mobile® - Main page

The second panorama item displays a filter with a list of transactions, so a user can quickly navigate between today, the current week or last week transactions without going to the full list of transactions as illustrated by figure 52. The transactions filter uses the ListPicker control. The popup displayed by this list picker control is created dynamically on-fly in the code-behind. The transactions list also uses a similar template on all the lists in the application which is created dynamically on-fly behind the code using entity handlers. All the columns displayed on the list view are controlled by the entity handler of each entity.

The application bar displayed on the main page is also initialized dynamically. The main page application bar shows only the login button when the login popup is displayed and sync and about button when main page is displayed. Sync button initializes the synchronization between the intermediate database and the local data store. Data synchronization procedure is defined in 'data synchronization' chapter.

5.4.5 Entity page

Entity page displays the list of entities. Simple and straight forward in look, entity page dynamically displays the contents of every possible entity supported by the application. The layout of the list displayed on the entity page is dynamically generated using the entity handlers which define the visible columns, column display order and column display style. All the entity handlers inherit from base handler class which holds most common business logic shared by each entity handler. The listing 17 illustrates the piece of code defined in the assignment handler defines the layout of the list of assignments.

```
public override List<EntityColumn> GetListColumns(string parentColumn
     )
{
    List<EntityColumn> columns = new List<EntityColumn>();
    columns.Add(new EntityColumn("assignment_number") { IsBold = true
      });
    columns.Add(new EntityColumn("subject")

     { TextStyle = TextStyle.CsiListTextBlockStyleRed });
    if (!parentColumn.Equals("customer_guid") && !parentColumn.Equals
     ("customer_name"))
        columns.Add(new EntityColumn("customer_name"));
    return columns;
}
```

Listing 17: Defining visible columns in entity handler

The base handler contains the actual business logic to create the list template for each entity type. The listing 18 illustrates the code to generate the list data template.

```
public virtual DataTemplate GetListItemTemplate(string parentColumn,
     bool forParentSelection = false)
{
     StringBuilder sbDataTemplateXml = new StringBuilder();
     sbDataTemplateXml.AppendLine(@"<DataTemplate x:Key=""EntityTempla
     te""
                  xmlns=""http://schemas.microsoft.com/winfx/2006/xam
     l/presentation""
                  xmlns:x=""http://schemas.microsoft.com/winfx/2006/x
     aml""
                  xmlns:toolkit=""clr-
     namespace:Microsoft.Phone.Controls;

     assembly=Microsoft.Phone.Controls.Toolkit"">");
     sbDataTemplateXml.AppendLine(@"<Grid Tag=""{Binding Mode=OneWay,
     Path=record_guid}"">");
     sbDataTemplateXml.Append(@"<Image HorizontalAlignment=""Left"" Ma
     rgin=""0,8,0,0"" Source=""/Resources/Images/List/");
     sbDataTemplateXml.Append(typeof(T).Name.ToLower());
     sbDataTemplateXml.AppendLine(@"_48.png"" Stretch=""Fill"" Width="
     "48"" Height=""48"" VerticalAlignment=""Top""/>");
     sbDataTemplateXml.AppendLine(@"<StackPanel Margin=""60,8,16,10""
     VerticalAlignment=""Top"">");

     // Columns
     foreach (var column in GetListColumns(parentColumn))
     {

         sbDataTemplateXml.Append(@"<TextBlock Text=""{Binding ");
         sbDataTemplateXml.Append(column.ColumnName);
         sbDataTemplateXml.Append(@"}"" HorizontalAlignment=""Left""

     Width=""Auto"" TextWrapping=""Wrap"" ");
         if (!forParentSelection)
         {
             sbDataTemplateXml.Append(@"Style=""{StaticResource ");
             sbDataTemplateXml.Append(column.TextStyle.ToString());
             sbDataTemplateXml.Append(@"}"" ");
         }
         sbDataTemplateXml.Append(@"Margin=""0,5,0,0""");
         if (column.IsBold)
             sbDataTemplateXml.Append(@" FontWeight=""Bold""");
         if (column.IsItalic)
             sbDataTemplateXml.Append(@" FontStyle=""Italic""");
         sbDataTemplateXml.AppendLine("/>");
     }

     sbDataTemplateXml.AppendLine("</StackPanel>");
     sbDataTemplateXml.AppendLine("</Grid>");
     sbDataTemplateXml.AppendLine("</DataTemplate>");

     DataTemplate dt = (DataTemplate)XamlReader.Load(sbDataTemplateXml
     .ToString());
     return dt;
}
```

Listing 18: Dynamic generation of list data template

In the code illustrated in listing 18, I created the string representation of XAML for the data template we want to use for the list. Then I used XamlReady.Load() method to load the XAML as DataTemplate object to memory.

The dynamic generation of lists removes the pain of creating entity pages for each entity supported by the application. I do not need to either worry of consistency, because every entity template is generated from the same code thus following the same design patterns. The final outputs of these dynamically generated lists are illustrated by figure 53.

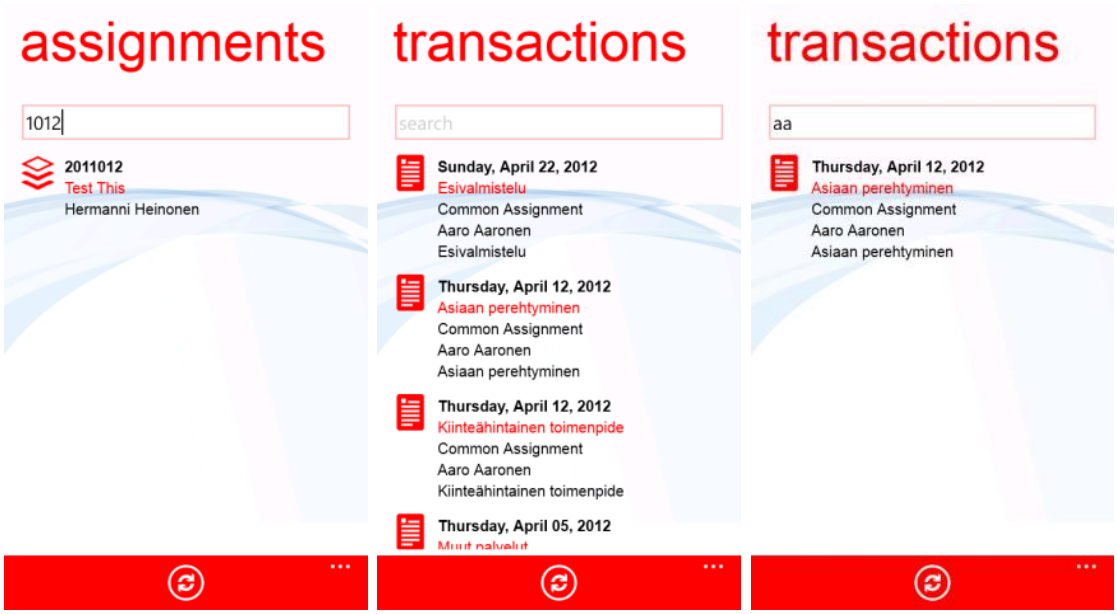


Figure 53: CSI Mobile® - Entity pages

Another important feature of the entity page is quick data filtering. The small textbox on top of the entity list highlighted in light red color can be used to filter the data from the list quickly. The business logic for filtering the data from the list when a user types something in the filter text box is defined in the entity handler of each entity as illustrated by listing 19. For example, we can see from the assignment handler that subject and assignment number columns are participating in filtering the data for assignment list.

```
public override ObservableCollection<IEntity> GetFilteredRecords(stri
     ng searchString, IEnumerable<IEntity> collectionToSearch = null)
{
     if (collectionToSearch == null)
         collectionToSearch = Entities;

     return new ObservableCollection<IEntity>(collectionToSearch.Cast<
      Assignment>()
                 .Where(r => r.subject.Contains(searchString)
                             || r.assignment_number.Contains(searchStr
      ing))
                 .Cast<IEntity>());

}
```

Listing 19: Business logic for list data filtering

The refresh icon placed in the application bar refreshes the list based on data stored on the isolated storage using isolated storage data context.

The Entity page also supports the context menu which is also dynamically generated for the list. Context menu of entity list exposes at least one action for each entity 'pin to start' as illustrated by figure 54. In some cases such as transaction, it also exposes the edit action for the entity.

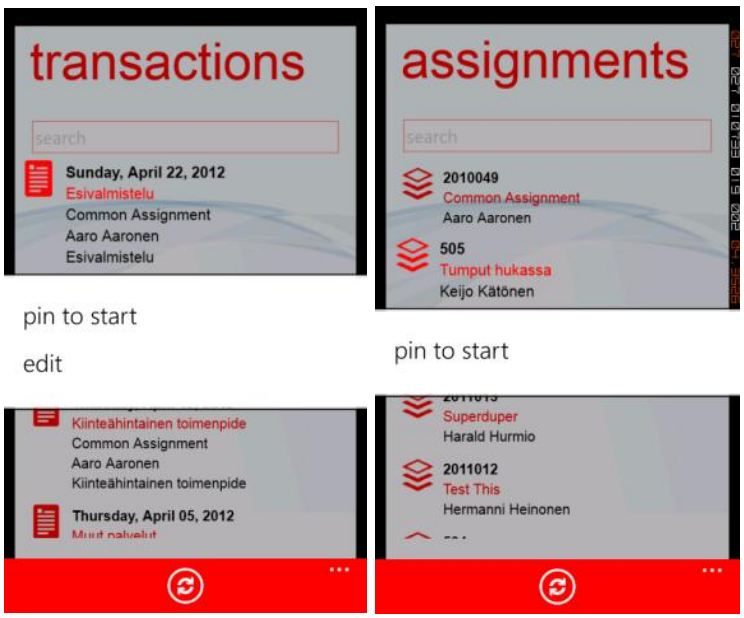


Figure 54: CSI Mobile® - Entity page context menu

Generation of list data context dynamically was one of the complex problems I faced during the development of this application. Dynamic generation of the context menu

for each item of list is not supported directly. Because I generate the list template dynamically in the code, XamlReader fails if we define the event handlers in the Xaml we want to load using XamlReader. For this application, it was important to have context menu for the list, because we want to allow the user to pin any entity to Start. In order to dynamically generate the context menu, CSI HELSINKI OY listened to the MouseLeftButtonDown event of list control. In the event listener, I tracked the clicked item and the group where it belongs. In the case of lists each list, item is grouped using a Grid. Once I had the full item group, creating and attaching of the context menu dynamically to the list was easy. For more details, see appendix 5.

5.4.6 Entity detail page

Entity detail page shows the details of the entity. It shows also the list of all the related entities of the specific entity. Entity detail page is also dynamically generated as many of the other pages of this application. The data displayed on the entity detail page is defined in the entity handler of each entity as illustrated in listing 20. The following code shows the data displayed for company entity.

```
public override List<EntityColumn> GetDetailsPageColumns()
{
    List<EntityColumn> columns = new List<EntityColumn>();
    columns.Add(new EntityColumn("customer_number"));
    columns.Add(new EntityColumn("customer_name"));
    columns.Add(new EntityColumn("customer_id_number"));
    columns.Add(new EntityColumn("primary_contact_name"));
    columns.Add(new EntityColumn("parent_customer_name"));
    columns.Add(new EntityColumn("customer_phone1"){ ColumnType = Col
     umnType.Phone});
    columns.Add(new EntityColumn("customer_phone2"){ ColumnType = Col
     umnType.Phone});
    columns.Add(new EntityColumn("customer_phone3"){ ColumnType = Col
     umnType.Phone});
    columns.Add(new EntityColumn("customer_email"){ ColumnType = Colu
     mnType.Email});
    columns.Add(new EntityColumn("customer_fax"));
    columns.Add(new EntityColumn("customer_web_site_url")
                                    {ColumnType = ColumnType.Web});
    columns.Add(new EntityColumn("primary_address_name")

     { ColumnType = ColumnType.Address });
    return columns;

}
```

Listing 20: Defining entity detail page columns in entity handler

The related entities are defined in the extended entity class with necessary captions and filters. The code illustrated in listing 21 shows how we define related records for each entity.

```
private void InitChilds()
{
    m_Childs.Add(new ChildDetail
    {
        DisplayName = AppResources.assignments,
        RecordType = typeof(Assignment),
        Records = OfflineEntityContextHelper.Context.AssignmentCollec
     tion

     .Where(r => r.customer_guid == record_guid).Cast<IEntity>(),
        FilteredColumn = "customer_guid"
    });
    m_Childs.Add(new ChildDetail
    {
        DisplayName = AppResources.transactions,
        RecordType = typeof(Transaction),
        Records = OfflineEntityContextHelper.Context

     .TransactionCollection.Where(r => r.customer_guid == record_guid)
        .OrderByDescending(r => r.transaction_date).Cast<IEntity>(),
        FilteredColumn = "customer_guid"
    });
    m_Childs.Add(new ChildDetail
    {
        DisplayName = AppResources.contacts,
        RecordType = typeof(Contact),
        Records = OfflineEntityContextHelper.Context.ContactCollectio
     n

     .Where(r => r.customer_guid == record_guid).Cast<IEntity>(),
        FilteredColumn = "customer_guid"
    });
    m_Childs.Add(new ChildDetail
    {
        DisplayName = AppResources.subsidiries,
        RecordType = typeof(Customer),
        Records = OfflineEntityContextHelper.Context.CustomerCollecti
     on

     .Where(r => r.parent_customer_id_guid == record_guid).Cast<IEntit
     y>(),
        FilteredColumn = "parent_customer_id_guid"
    });
}
```

Listing 21: Defining related records in entity class

The entity detail page utilizes many of the Windows Phone® capabilities to use the data for performing various simple tasks on the phone. All the phone numbers are automatically listed in two sections; call and SMS. It is possible to call, send SMS, view

address location on the map and send email using the data displayed on the entity details page as illustrated by figure 55. The functionality of data displayed on the entity details page is defined in the entity handler of each entity.

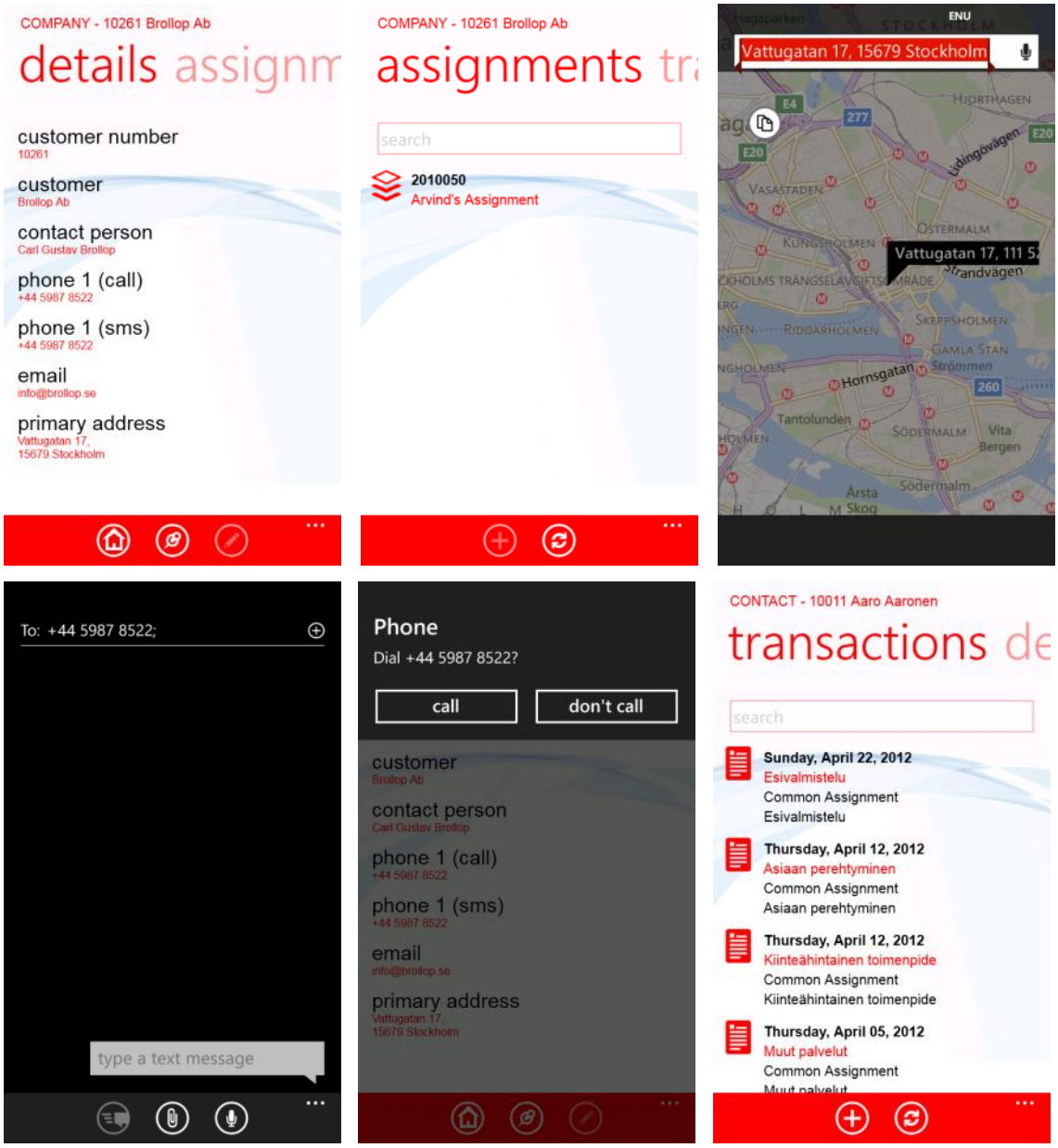


Figure 55: CSI Mobile® - Entity detail page

The application bar of the entity detail page is dynamically generated. As shown in the picture above, the application bar changes according to page views.

5.4.7 Edit transaction page

The edit transaction page allows the user to edit or create new transactions in the CSI Mobile® application. The edit transaction page only allows saving of transactions if data is valid. The real-time data validation is performed while a user completes the transaction details. The save icon on the application bar is activated when data on the edit transaction page is ready for saving as illustrated by figure 56.

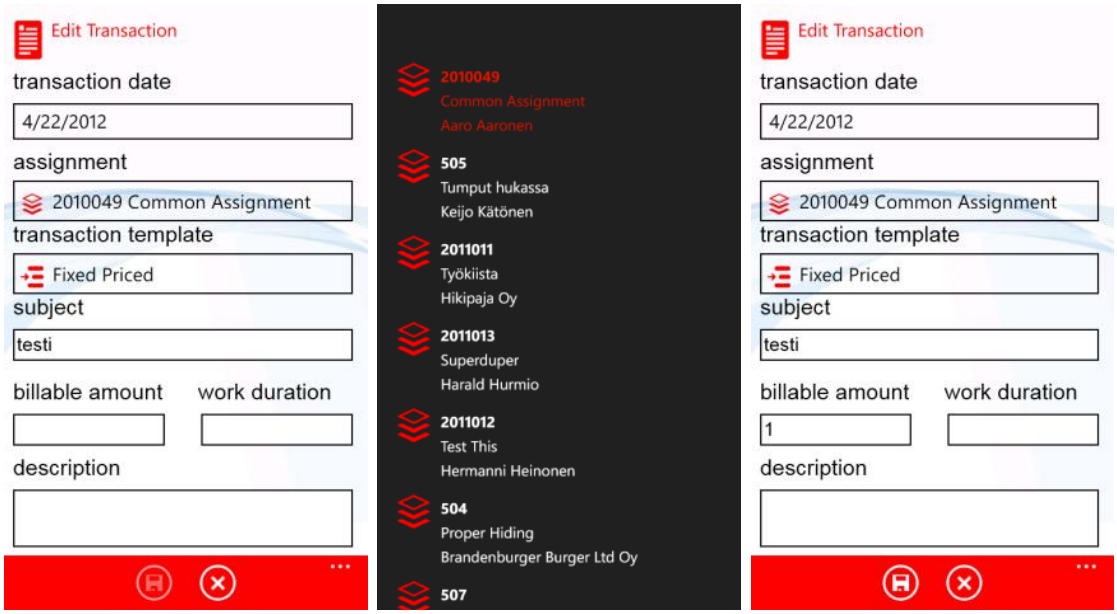


Figure 56: CSI Mobile® - Edit transaction page

Edit transaction uses entity lookup controls for linking the assignment and transaction templates for transaction. It's important to notice that entity lookup pop uses the same list template for assignment and transaction template as used for entity list.

When the user hits the save button, transaction object is send to transaction handler which finalize the data processing before committing it to the isolated stored using isolated storage data context.

5.4.8 Live tiles

CSI Lawyer® application allows pinning of any item from the list on the Start. CSI Mobile® live tiles reflect the Windows themes as illustrated by figure 57. CSI Mobile®

live tiles have front and back contents. Every entity in the CSI Mobile® inherits IEntity interface which implements the display name property. Display name property of the entity is used for live tile title.

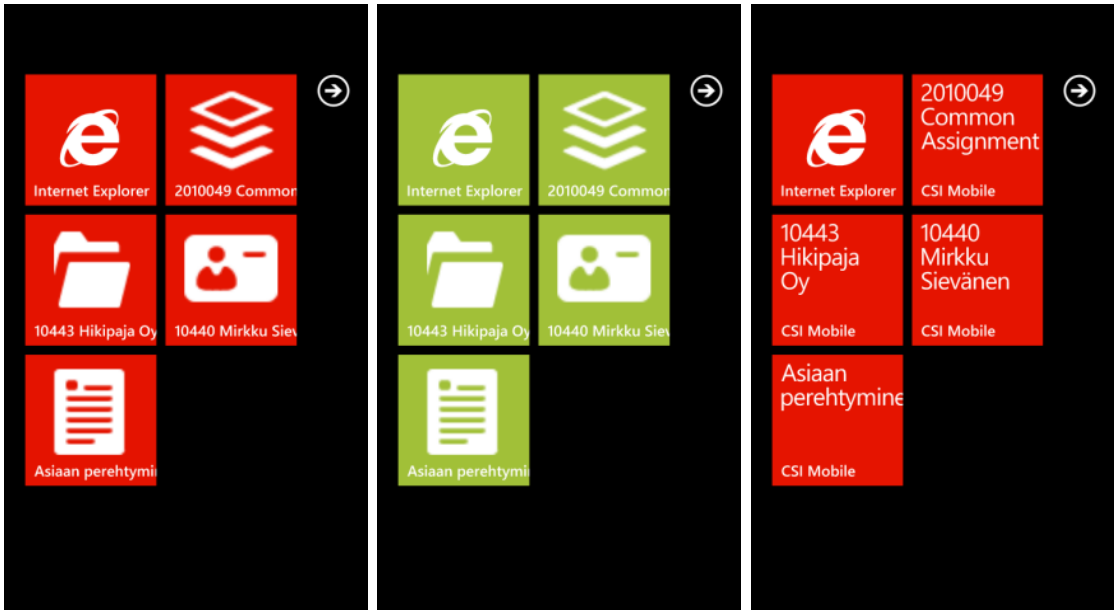


Figure 57: CSI Mobile® - Live tiles

Every secondary live tile in CSI Mobile® application is created using the TileHelper which contains only one method CreateTile(). Before creating new tile on the Start, it checks weather live tile for specific entity exists or not. If live tile already exists for a specific entity, no new live tile is created. Clicking on the secondary live tile from the Start takes the user to the entity detail page of that specific entity.

```
public static void CreateTile(IEntity  entity)
{
    var foundTile = ShellTile.ActiveTiles.FirstOrDefault(x => x.Navig
     ationUri.ToString().Contains(
        string.Format("record_guid={0}&entityType={1}", entity.record
     _guid, entity.GetType().Name.ToLower())));

    if (foundTile == null)
    {
        var secondaryTile = new StandardTileData
        {
            BackgroundImage = new Uri(String.Format("/Resources/Image
     s/Tile/{0}_background_tile.png",

     entity.GetType().Name.ToLower()), UriKind.Relative),
            Title = entity.display_name,
            BackTitle = AppResources.application_name,
```

```
                BackContent = entity.display_name,
            };

            ShellTile.Create(new Uri(String.Format("/EntityDetailPage.xam
        l?record_guid={0}&entityType={1}",
                entity.record_guid, entity.GetType().Name.ToLower()),
                UriKind.Relative), secondaryTile);
        }

    }
```

Listing 22: Creating live secondary tiles

The code illustrated in listing 22 shows the method of TileHelper which creates the live tile on the Start.

5.4.9 Background agents

CSI Mobile® supports both kinds of background agents for synchronizing the data with intermediate data supported by Windows Phone®. These background agents are periodic and resource intensive background agents. Due to the limitation of both background agent types, it is important to use both kinds of background agents for synchronizing data in the background. Background agents can be activated or deactivated for synchronizing data in background using the settings of CSI Mobile® application.

5.5 Data synchronization

CSI Mobile® uses background agents if active for synchronizing data with the backend supported by the Windows Phone®. It is also possible to perform quick synchronization by using 'Sync' button available on the application bar of the main page. Every data synchronization request is verified by the sync service before actual synchronization thus requiring the user credentials to be sent along. The credential information is added in the request headers in the right format as required by the sync service. For synchronization code, please see the appendix 6.

5.6 Testing

Due to the time shortage while writing this project, minimal functional testing for the backend and Windows Phone® application was performed. During the testing of the prototype application some minor user interface related bugs were found and fixed. A Critical system such as this needs a sufficient amount of testing before putting the backend and Windows Phone® application into production.

During the testing, overall application performs well, user interface was responsive and informative and data synchronization just works such as magic transparently behind the scenes without any problems.

6 User experience

6.1 Identified problems

The synchronization of data between the CSI Lawyer® database and intermediate database based on the favorite assignment was one of the most challenging tasks I faced during the whole project. In order to achieve the expected results in synchronization, I needed to understand the whole architecture of the Microsoft Sync Framework.

Some of the data manipulation and processing was easy and I performed it in the code by listening to various events provided by the Microsoft Sync Framework provider. However listening to the events in the code was not enough. After analyzing the stored procedures and SQL triggers generated and used by Microsoft Sync Framework to select changes, I was able to change them as required.

Another problem I faced was in exposing the Microsoft Sync Framework OData endpoint over the SLL and implementing custom authentication. All the examples and sample codes available on the Internet for implementing OData endpoint did not define how to expose the OData endpoint using SSL. Because I had never developed any WCF service which would accept the REST request, I did not know that a special binding type known as 'webHttpBinding' existed and could have been used. Another problem was the interface used in the binding. The default HTTP solution does not need any kind of binding to be defined in the web.config, but when I want to expose the data over HTTPS, binding must be configured in web.config. After searching for different custom solutions implemented by other people and checking their binding configuration, I found that Microsoft.Synchronization.Services.IRequestHandler had to be used in the binding.

After I successfully managed to expose the OData sync service over the http, Windows Phone® never managed to synchronize data. The only error I got was the 'Not Found' error. It took me days to solve the 'Not Found' error. Finally I did some more research

I found that this error occurred because of the Silverlight cross domain problem. Once I knew the problem, the fixing was quick.

During the development of the Windows Phone® application, I found it challenging to develop an architecture where almost the whole user interface was dynamically generated based on the business logic defined by handlers and entities. The biggest problem for me was to dynamically create list views for each entity list parsed form XAML generated in the code. Overall that problem was also solved quite quickly if compared to the problem of displaying the context menu for dynamically generated entity list data templates. I searched many days for the solution to the problem of creating the context menu for the dynamically loaded list template without any result. I was surprised that nobody else had faced the same problem where user interfaces are generated in the code for Windows Phone®. After researching and analyzing many days, I finally came up with my own solution for linking the context menu dynamically with dynamically loaded data template of the list control.

Another problem I faced was the data encryption algorithms supported by Windows Phone®. The sync service required passwords to be encrypted using the Rijndael encryption algorithm, whereas Windows Phone® does not support .Net libraries for Rijndael encryption. The workaround was to encrypt a password using Advanced Encryption Standard (AES) and then re-encrypt it using the Rijndael encryption on the server side.

Despite the major development problems, I had to spend much time to study how to generate a self-signed certificate for the IIS and make Windows Phone® trust the localhost authority used by my self-signed certificate. If the SSL certificate is not trusted by Windows Phone®, one will receive a 'Not Found' exception which does not tell what the actual problem is.

6.2 Enduser feedback

At the time of writing this project, CSI Helsinki Oy did not have enough time to put the CSI Mobile® application for testing to the real users. So the only feedback received

during this application development was based on company colleagues, instructors and testers.

During the testing, the test team found a few bugs related to data synchronization but overall the application performance was declared to be satisfactory. The users felt the application was easy to use, fast and consistent. The users were impressed with the simplicity and unique application's graphics design.

6.3 Future development

The first important future development suggestion would be to change the data synchronization between the CSI Lawyer® and the intermediate database using the Rest OData endpoint instead of direct access. This would solve some of the possible security holes created by communication of the Windows Sync Client application.

Another development suggestion would be to add tasks, work time, critical tasks and chart support on the CSI Mobile®. It would be crucial for the users spending most of the time outside the office to record his/her working hours for specific assignment or customer as well as to maintain critical and normal tasks straight from the CSI Mobile® application.

Throughout testing the CSI Mobile® application and backend data synchronization before production use is highly recommended due to the minimal testing performed during the CSI Mobile® application development.

7 Conclusions

The main target of the project was to research and implement a possible solution including the Windows Phone® application for allowing the CSI Lawyer® users to create transactions and access transaction related information such as assignments, customers and transaction templates from the Windows Phone®.

The major part of the project was to develop the right scalable backend infrastructure which needed minimum effort from the enduser to take the system into use. To minimize the workload, CSI Helsinki Oy decided to use existing technologies and framework, so the focus would stay mostly on the business logic rather than on low-level infrastructure.

Microsoft Sync Framework combined with Windows Azure was found to be a scalable and easy to implement solution, fully supported by Microsoft. There is a large amount of help, documentations, sample codes and API in a printed and web versions of these technologies and framework.

The backend solution was the most difficult and challenging part of the whole project whereas Windows Phone® application development was the most interesting part of the project. The number of tools, control libraries and Windows Phone® APIs provided by Microsoft and the third parties were easy to use. Some of the Windows Phone® SDK restrictions came up when developing CSI Mobile® such as Windows Phone® API does not allow to search and link the received calls in the application when any phone call is received or made by Windows Phone®.

The final prototype of CSI Mobile® came to be consistent in the user interface design, stable in working and easily scalable in architecture. CSI Mobile® followed the guidelines for the user interface design with some of the best practices suggested by the Windows Phone® development team. The CSI Mobile® application performs the main task of creating and editing transaction on the Windows Phone® and then synchronizing it to the CSI Lawyer® database using backend logic in an excellent way.

Overall the project was a challenge for me as most of the frameworks and technologies used in developing the backend and Windows Mobile application were unknown to me at the beginning of this project. The objectives defined in the functional and architectural requirements were achieved with a fully working prototype application for Windows Phone® and necessary components for backend data synchronization as the outcome of the project.

References

[1] T. Ricker. Microsoft announces ten Windows Phone 7 handsets for 30 countries: October 21 in Europe and Asia, 8 November in US (Update: Video!) [Online]. URL: http://www.engadget.com/2010/10/11/microsoft-announces-ten-windows-phone-7-handsets-for-30-countrie/. Accessed 9 September 2011.

[2] S. Miles. Windows Phone 7: Microsoft confirms launch partners [Online]. URL: http://www.pocket-lint.com/news/34441/windows-phone-7-launch-partners. Accessed 9 September 2011.

[3] Nokia. Nokia and Microsoft Announce Plans for a Broad Strategic Partnership to Build a New Global Mobile Ecosystem [Online]. URL: http://press.nokia.com/2011/02/11/nokia-and-microsoft-announce-plans-for-a-broad-strategic-partnership-to-build-a-new-global-ecosystem/. Accessed 9 September 2011.

[4] C. Davies. Windows Phone "Mango" official; Acer, Fujitsu and ZTE onboard," Microsoft News Center [Online]. URL: http://www.slashgear.com/windows-phone-mango-official-acer-fujitsu-and-zte-onboard-24153926/. Accessed 12 September 2011.

[5] CSI Helsinki Oy. Asiakkuudenhallinta ja toiminnanohjaus [Online]. URL: http://www.csihelsinki.fi/suomeksi/Yritys/CSIHelsinkiOy/tabid/189/Default.aspx. Accessed 25 November 2011.

[6] Egham. Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent [Online]. URL: http://www.gartner.com/it/page.jsp?id=1848514. Accessed 22 November 2011.

[7] "Worldwide Smartphone Market Expected to Grow 55% in 2011 and Approach Shipments of One Billion in 2015, According to IDC," IDC press release, 9 June 2011. [Online]. URL: http://www.idc.com/getdoc.jsp?containerId=prUS22871611. Accessed 20 November 2011.

[8] D. Betts, Federico Boerr, Scott Densmore, Jose Gallardo Salazar and Alex Homer, Windows® Phone 7 Developer Guide, New York: Microsoft Corporation, 2010.

[9] C. Tilley. The History of Windows CE [Online]. URL: http://www.hpcfactor.com/support/windowsce/. Accessed 13 November 2011.

[10] Wikipedia. Windows Mobile [Online]. URL: http://en.wikipedia.org/wiki/Windows_Mobile. Accessed 5 December 2011.

[11] A. Orlowski. Why Symbian failed: developers, developers, developers [Online]. URL: http://www.theregister.co.uk/2010/11/09/symbian_developers_mailbag/. Accessed 15 November 2011.

[12] C. F. Nick Randolph, Professional Windows® Phone 7 Application Development: Building Applications and Games Using Visual Studio, Silverlight®, and XNA®, Indianapolis: Wiley Publishing, Inc., 2011.

[13] M. Mercan. Windows Phone 7 (PH7) - 101 [Online]. URL: http://www.mrtmrcn.com/en/post/2011/09/15/Windows-Phone-7-%28-PH7-%29-Introduction.aspx. Accessed 12 December 2011.

[14] Microsoft. Chapter 7 - Interacting with Windows Marketplace [Online]. URL: http://msdn.microsoft.com/en-us/library/gg490776.aspx. Accessed 22 December 2011.

[15] T. Brix. Windows Phone Developers Get New App Hub Features: Mango app submission just one month away [Online]. URL: http://windowsteamblog.com/windows_phone/b/wpdev/archive/2011/07/20/windows-phone-developers-get-new-app-hub-features-mango-app-submission-just-one-month-away.aspx. Accessed 25 December 2011.

[16] S. Wildermuth, Essential Windows Phone 7.5, Addison-Wesley Professional, 2011.

[17] Open Data Protocol [Online]. URL: http://www.odata.org/. Accessed 4 Feburary 2012.

[18] Microsoft. Local Database Overview for Windows Phone [Online]. URL: http://msdn.microsoft.com/en-us/library/hh202860%28v=vs.92%29.aspx. Accessed 5 Feburary 2012.

[19] Microsoft. Windows Azure Platform [Online]. URL: http://www.windowsazure.com/en-us/develop/overview/. Accessed 15 April 2012.

[20] Microsoft. Introduction to Microsoft Sync Framework [Online]. URL: http://msdn.microsoft.com/en-us/sync/bb821992. Accessed 5 Feburary 2012.

[21] Microsoft. Appendix E - Microsoft Sync Framework and Windows Phone 7 [Online]. URL: http://msdn.microsoft.com/en-us/library/gg507824.aspx. Accessed 5 Feburary 2012.

[22] Microsoft. SQL Server Replication [Online]. URL: http://msdn.microsoft.com/en-us/library/ms151198.aspx. Accessed 15 April 2012.

[23] Microsoft. How to: Provision and Deprovision Synchronization Scopes and Templates (SQL Server) [Online]. URL: http://msdn.microsoft.com/en-us/library/ff928603.aspx. Accessed 20 April 2012.

[24] Microsoft. SyncOrchestrator Class [Online]. URL: http://msdn.microsoft.com/en-us/library/cc306211.aspx. Accessed 20 April 2012.

[25] Microsoft. Making a Service Available Across Domain Boundaries [Online]. URL: msdn.microsoft.com/en-us/library/cc197955(v=vs.95).aspx. Accessed 21 April 2012.

[26] Microsoft. Appendix E - Microsoft Sync Framework and Windows Phone 7 [Online]. URL: http://msdn.microsoft.com/en-us/library/gg507824.aspx. Accessed 21 April 2012.

[27] P. Torr. Tilt Effect [Online]. URL: http://blogs.msdn.com/b/ptorr/archive/2010/08/11/updated-tilt-effect.aspx. Accessed 2 April 2012.

ProgressPopup.Xml

```
<UserControl x:Class="Csi.Wp7.Client.Controls.ProgressPopup"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    xmlns:toolkit="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone.Controls.Toolkit"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    d:DesignHeight="480" d:DesignWidth="480">

    <Grid x:Name="LayoutRoot" Background="Transparent" Width="480" Height="800">
        <Rectangle x:Name="backgroundRect" Grid.Row="0" Fill="{StaticResource PhoneChrom
eBrush}" Opacity="0.75"/>
        <StackPanel x:Name="stackPanel" Orientation="Vertical" VerticalAlignment="Center
">
            <toolkit:PerformanceProgressBar Name="pbTwitter"
                            IsIndeterminate="True" Foreground="Red" />
            <TextBlock Opacity="1" Height="30"
                       HorizontalAlignment="Center"
                       VerticalAlignment="Center"
                       Foreground="Red"
                 Name="textBlockStatus" Text="{Binding LocalizedResources.loading,
                                    Source={StaticResource LocalizedStrings}}" />
        </StackPanel>
    </Grid>
</UserControl>
```

ProgressPopup.Xml.cs

```
namespace Csi.Wp7.Client.Controls
{
    public partial class ProgressPopup : UserControl
    {
        internal Popup ChildWindowPopup
        {
            get;
            private set;
        }
        public string LoadingText
        {
            get { return textBlockStatus.Text; }
            set { textBlockStatus.Text = value; }
        }

        public ProgressPopup()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Shows this progress bar popup.
        /// </summary>
        public void Show()
        {
            if (ChildWindowPopup == null)
            {
                ChildWindowPopup = new Popup();

                try
                {
                    ChildWindowPopup.Child = this;
                }
                catch (ArgumentException)
```

```
                {
                    throw new InvalidOperationException("The control is already shown.")
;
                }
            }

            if (ChildWindowPopup != null && Application.Current.RootVisual != null)
                ChildWindowPopup.IsOpen = true;
        }

        /// <summary>
        /// Hide this progress bar popup.
        /// </summary>
        public void Hide()
        {
            ChildWindowPopup.IsOpen = false;
        }
    }
}
```

DefaultTextbox.cs

```
namespace Csi.Wp7.Client
{
    public class DefaultTextbox : TextBox
    {
        private string _defaultText = string.Empty;
        public string DefaultText
        {
            get
            {
                return _defaultText;
            }
            set
            {
                 defaultText = value;
                SetDefaultText();
            }
        }

        public DefaultTextbox()
        {

            this.GotFocus += (sender, e) =>
            {
                Foreground = new SolidColorBrush(Colors.Black);

                if (this.Text.Equals(DefaultText)) { this.Text = string.Empty; }
            };
            this.LostFocus += (sender, e) => { SetDefaultText(); };
        }

        private void SetDefaultText()
        {
            if (this.Text.Trim().Length == 0)
            {
                this.Text = DefaultText;
                Foreground = new SolidColorBrush(Colors.LightGray);
            }
        }
    }
}
```

```
private BackgroundWorker backroungWorker;
public MainPage()
        {
            InitializeComponent();
            if(AppSettings.Instance.AutoSync)
                AgentHandler.Activate();

            ShowSplashScreen();
        }

private void ShowSplashScreen()
        {
            m_Popup = new Popup();
            m_Popup.Closed += popup_Closed;
            m_Popup.Opened += popup_Opened;
            m_Popup.Child = new SplashScreen();
            m_Popup.IsOpen = true;
            StartLoadingData();
        }

        void popup_Opened(object sender, EventArgs e)
        {
             Dispatcher.BeginInvoke(() =>
                {
                   if (m_Popup.Child is LoginControl)
                       InitilaizeApplicationBarForLoginPage();
                   else
                       ApplicationBar.IsVisible = false;
                });
        }

        void popup_Closed(object sender, EventArgs e)
        {
            Dispatcher.BeginInvoke(() =>
                {
                   InitilaizeApplicationBarForMainPage();
                });
        }

        private void StartLoadingData()
        {
            backroungWorker = new BackgroundWorker();
            backroungWorker.DoWork += backroungWorker_DoWork;
            backroungWorker.RunWorkerCompleted += backroungWorker_RunWorkerCompleted;
            backroungWorker.RunWorkerAsync();
        }

        void backroungWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventAr
gs e)
        {
            Dispatcher.BeginInvoke(() =>
                {
                    m_Popup.IsOpen = false;
                }
            );
        }

        void backroungWorker_DoWork(object sender, DoWorkEventArgs e)
        {
            Thread.Sleep(5000);
        }
```

```
public class LocalizedStrings : INotifyPropertyChanged
{
    private readonly static AppResources localizedResources = new AppResources();
    public AppResources LocalizedResources { get { return localizedResources; }
    }

    public void ResetResources()
    {
        AppResources.Culture = Thread.CurrentThread.CurrentUICulture;
        OnPropertyChanged(() => LocalizedResources);
    }

    #region INotifyPropertyChanged region
    public event PropertyChangedEventHandler PropertyChanged;

    public void OnPropertyChanged<T>(Expression<Func<T>> selector)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(GetPropertyNameFromExpres
sion(selector)));
        }
    }

    public static string GetPropertyNameFromExpression<T>(Expression<Func<T>> property)
    {
        var lambda = (LambdaExpression)property;
        MemberExpression memberExpression;

        if (lambda.Body is UnaryExpression)
        {
            var unaryExpression = (UnaryExpression)lambda.Body;
            memberExpression = (MemberExpression)unaryExpression.Operand;
        }
        else
        {
            memberExpression = (MemberExpression)lambda.Body;
        }

        return memberExpression.Member.Name;
    }
    #endregion
}
```

```
m_lb_Entities.MouseLeftButtonDown += m_lb_Entities_MouseLeftButtonDown;

void m lb Entities MouseLeftButtonDown(object sender, System.Windows.Input.MouseButtonEv
entArgs e)
{

    if (e.OriginalSource is ListBox)
        return;

    var element = VisualTreeElementHelper.FindFirstElementInVisualTreeParent<Grid>((UIEl
ement)e.OriginalSource);
    if (element == null)
        return;

    // Selected entity
    Guid entityGuid = element.Tag != null ? new Guid(element.Tag.ToString()) : Guid.Empt
y;
    var source = m lb Entities.ItemsSource as IEnumerable<IEntity>;
    if (entityGuid != Guid.Empty && source != null)
        m SelectedEntity = source.First(r => r.record guid == entityGuid);

    if (m_SelectedEntity == null)
        return;

    // Initiate context menu
    ContextMenu cm = new ContextMenu();
    var pinToStartMenuItem = new Microsoft.Phone.Controls.MenuItem() { Header = AppResou
rces.pintostart.ToLower() };
    pinToStartMenuItem.Click += pintToStartMenuItem_Click;
    cm.Items.Add(pinToStartMenuItem);


    if (m_IEntityHandler.CanEditEntity(m_SelectedEntity))
    {
        var editMenuItem = new Microsoft.Phone.Controls.MenuItem() { Header = AppResourc
es.edit.ToLower() };
        editMenuItem.Click += editMenuItem Click;
        cm.Items.Add(editMenuItem);
    }
    ContextMenuService.SetContextMenu(element, cm);


}
```

```
private void ApplicationBarSyncButton_Click(object sender, EventArgs e)
{
    ProgressBar.Show();
    OfflineEntityContextHelper.Context.LoadCompleted += Context LoadCompleted;
    OfflineEntityContextHelper.Context.LoadAsync();
}


void Context LoadCompleted(object sender, Microsoft.Synchronization.ClientServices.Isola
tedStorage.LoadCompletedEventArgs e)
{
    try
    {
        if (e.Exception != null)
        {
            Dispatcher.BeginInvoke(() =>
            {
                ProgressBar.Hide();
                MessageBox.Show(e.Exception.Message);
            });
        }
        else
        {
            Dispatcher.BeginInvoke(() =>
            {
                OfflineEntityContextHelper.Context.CacheController.RefreshCompleted += C
acheController_RefreshCompleted;
                OfflineEntityContextHelper.Context.CacheController.RefreshAsync();
            });
        }
    }
    finally
    {
        OfflineEntityContextHelper.Context.LoadCompleted -= Context_LoadCompleted;
    }
}

void CacheController RefreshCompleted(object sender, Microsoft.Synchronization.ClientSer
vices.RefreshCompletedEventArgs e)
{
    ProgressBar.Show();
    if (e.Error != null)
    {
        Dispatcher.BeginInvoke(() =>
        {
            MessageBox.Show(e.Error.Message);
        });
    }
    else
    {
        OfflineEntityContextHelper.Context.CacheController.RefreshCompleted -
= CacheController RefreshCompleted;
        OfflineEntityContextHelper.AddStats(e.Statistics, e.Error);

        StringBuilder sb = new StringBuilder();
        sb.AppendLine("TotalChangeSetsDownloaded: " + e.Statistics.TotalChangeSetsDownlo
aded);
        sb.AppendLine("TotalChangeSetsUploaded: " + e.Statistics.TotalChangeSetsUploaded
);
        sb.AppendLine("TotalDownloads: " + e.Statistics.TotalDownloads);
        sb.AppendLine("TotalSyncConflicts: " + e.Statistics.TotalSyncConflicts);
        sb.AppendLine("TotalSyncErrors: " + e.Statistics.TotalSyncErrors);
        sb.AppendLine("TotalUploads: " + e.Statistics.TotalUploads);
        MessageBox.Show(sb.ToString());

    }
    ReLoadTransactionList();
    ProgressBar.Hide();
}

static void InitContext()
{
```

```
    if (AppSettings.Instance.UserIdSetting != Guid.Empty)
    {
        context = new OfflineEntityContext("CSILawyer " + AppSettings.Instance.UserIdSet
ting.ToString(), new Uri(CommunicationHelper.SyncServiceUrl));
        context.CacheController.ControllerBehavior.BeforeSendingRequest += BeforeSending
Request;
        context.CacheController.ControllerBehavior.AddScopeParameters("userGuid", AppSet
tings.Instance.UserIdSetting.ToString());
        context.CacheController.ControllerBehavior.SerializationFormat = SerializationFo
rmat.ODataJSON;
    }
}

public static void BeforeSendingRequest(HttpWebRequest req, Action<HttpWebRequest> resum
ption)
{
    req.Headers[HttpRequestHeader.Authorization] = CommunicationHelper.GetAuthorizationH
eader(AppSettings.Instance.EmailSetting, AppSettings.Instance.PasswordSetting);
    // Must resume
    resumption(req);
}
```