HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

coresystems

Mobile Enterprise Resource Planning with a Windows 8 Store App

Bachelor Thesis

Department of Computer Science
University of Applied Science Rapperswil

Autumn Term 2012

Author(s):	Timon Brüllmann Amic Celebic Thomas Geiser
Advisor:	Prof. Dr. Luc Bläser
Project Partner:	coresystems AG, Windisch
External Co-Examiner:	Thomas Briner, Dipl. Inf.-Ing. ETH
Internal Co-Examiner:	Prof. Dr. Joseph Joller

ABSTRACT

Coresystems Inc. is a leading Swiss software development company of ERP solutions for international customers. While coresystems already offers their mobile ERP application on iOS, Android and Silverlight frontends, there is currently no support for Microsoft's Windows 8 platforms. The goal of this project was therefore to develop a new Windows 8 ERP frontend for coresystems, applying an innovative UI design and software architecture.

In this project, we have designed and implemented the new mobile ERP application for Windows 8 Store, which supports the major data model of the coresystems ERP and also integrates to their existing data cloud. Compared to the existing coresystems solutions, this application features the following highlights: (1) A cutting-edge user interface design which follows the guidelines for Windows 8 Store Apps and is touch-optimized both on tablets and PCs**.** (2) A flexible offline modus which allows the user to work on a local database that performs automatic background synchronization to cloud services when connected to the internet. (3) A flexible data model that can be easily extended towards a larger set of ERP data on the coresystems cloud.

Our developed ERP mobile application is fully functional, enables intuitive usage, effective offline and online work. It will serve as a demonstrator for coresystems in the Microsoft's mobile platform area and provides a solid base for further development and extensions. It is planned that the new application will be presented at the CeBit 2013 tech fair.

1 Management Summary

1.1 Current Situation

Coresystems maintains a cloud service, which integrates multiple enterprise resource planning (ERP) systems, like SAP or Microsoft Dynamics, into a single cloud storage. For this cloud, coresystems developed various applications, such that sales and service representatives have all the relevant company data on mobile devices like smartphones, tablets or laptops.

At the beginning of this project, these ERP apps were available for Apple and Android devices, but not for Windows 8 Store apps. As the sales department of coresystems is however often approached by customers asking for a Windows Store app, coresystems decided to develop a prototype, which can be shown to potential customers. The development of this app was then assigned as the topic for this thesis.

1.2 Task and Approach

The Windows 8 based mobile ERP app should not only be a simplistic prototype, but should include complex features, such as integration of the existing coresystems cloud, local data persistence for offline work support, and a modern state-of-the-art Windows 8 user interface.

To reach this goal, the quality must be ensured through testing and good project management. It is a typical software engineering project, not just a proof-of-concept that can be thrown away afterwards.

In a first phase requirements have been collected and formally defined. To ensure viability, pre-studies have been done in most aspects of the application. Coresystems was involved in designing the user interface (UI) by usability testing a paper prototype and giving feedback.

The second phase consisted of creating a read-only version of the app, which already included persistence. After showing it to coresystems and getting feedback, we knew exactly in which direction to head and added features in every iteration.

Quality was reached through a lot of testing with the cloud, unit-testing and a robust architecture.

1.3 Conclusion

The final product has met all demanded requirements. The UI follows closely the guidelines for Windows 8 Store apps, it is modern, has an intuitive look-and-feel and is touch-optimized. Thanks to the offline mode and synchronization design, the user never has to care doing a synchronization with the cloud – it all happens in the background.

Unfortunately many new features in .NET 4.5 and the Windows 8 Store App SDK are quite complex and are hard to grasp through small tutorials. That is why a lot of time has been lost in creating the UI and also in designing the synchronization. This was a reason why Windows Phone 8 App could not be included in the project.

All in all, the project was a success and delivered a robust application that constitutes a solid basis for coresystems to extend with additional features and modules.

1.4 Outlook

Now that the application supports a specific set of data, i.e. customer-related ERP data, it is a good basis to be broadened to further areas. The extension of functionality by modules, like reports or sales orders, is rather straightforward. If the desired business entity already exists in the project, no conversions or mappings have to be added. Otherwise the business model has to be expanded. If this is ensured, UI masks can be provided as needed.

To have the same level of functionality as coresystems' iPad or Silverlight app, we estimate an effort of about two to three months in a team of three developers, which is already familiar with the business logic and the desired technologies.

2 THANKS

We would like to thank Prof. Dr. Luc Bläser for supervising this project and helping us in difficult stages of the project. We would also like to thank the coresystems team for supporting us with the development of this application.

Another big thanks goes to our families and friends who were missed out a little bit during our bachelor thesis.

Table of Contents

3 INTRODUCTION

Coresystems has become the innovation leader in their industry by adapting an intelligent cloud and creating revolutionary products for accessing enterprise resource planning systems. They have developed mobile services for iOS and Android devices and also provide a web based interface to communicate with the ERP-system.

To extend their portfolio and to answer customers' demands, they would like frontend support for Windows 8 devices. The goal of such an app is to access the ERP-System (Customer Relationship, Sales and Services) through the existing cloud-backend, to intuitively view and manipulate the provided data.

This new mobile ERP application should have distinct qualities, to differentiate itself from existing solutions on the Windows 8 RT platform. Following aspects should be considered through innovative software-design:

- **Cutting-Edge UI:** The user interface is easy-to-use and can handle large amount of data, without breaking the flow of the application. It follows the Windows 8 style guides and has a fresh and modern look-and-feel.
- **Offline Support:** Since there is not always an internet connection available, the app supports an offline-mode. Data is cached and persisted locally and is synchronized with the cloud dynamically, when there is a connection available.
- **Robust architecture**: The application is both, a show-case for marketing and a base for further development. As a result, the app includes the core-functionality and is easy to expand.

The first chapter of this document is the analysis chapter, containing the requirements of the app, the UI design with the paper prototype and finally an overview of the technologies used in this project.

The next chapter is about the application's architecture. In it we discuss our decisions and show the core-functions of the app in-depth.

And finally, there is an overview of the testing we made throughout the development and a chapter with the conclusion and outlook of the project.

4 ANALYSIS

4.1 REQUIREMENT SPECIFICATION

This chapter describes the motivation of this software solution called MobileERP, as well as the requirements of the software. Due to time limitation, only a subset of all possibilities, which are generally done in an ERP-system, are realized.

4.1.1 Motivation

With the release of Windows 8 by the end of October 2012 there opened a new market for mobile business applications. This is a great opportunity for coresystems to add a new product to their mobile ERP solutions portfolio.

The goal of this software project is to develop a first version of a mobile client for Windows 8 Store app.

4.1.2 Basic Information

Windows 8 comes with a total new user interface, which is designed to use with fingers or a mouse and keyboard. Due to this major change in the interaction principles, it was one of the tasks to design a UI based on the guidelines from Microsoft.

4.1.3 Non-Functional Requirements

4.1.3.1 Reliability

The mobile software client should work with a connection to the cloud as well as if it is offline. When there is a connection to the server it synchronizes the data as a background task. If the connection gets lost, it will store and trace changes until a connection is available again.

4.1.3.2 Usability

The UI has to be intuitive for the user and it should be very easy to understand and learn. In addition, the user interface should follow the guidelines of Microsoft regarding Windows 8 Store Apps.

4.1.3.3 Performance

The UI should be responsive all the time and therefore all task, like synchronizing or storing data in the local database, have to be performed in background.

4.1.3.4 Database

The database must be capable of storing and handling large amounts of data. Some of coresystems customers have up to 100’000 unique objects.

4.1.3.5 Maintainability

A modular architecture is used for the MobileERP applications. All interfaces are specified and therefore later changes or extensions can be added in an easy way.

4.1.3.6 Platform

This software is a platform-dependent application. The project is being developed for Windows 8 and tested on this platform.

4.1.3.7 *Online/Offline Support*

The application should work when an internet connection is available (online) and if there is no access to the internet. In this case it should be able to work with local data. As soon as there is a connection available, the local changes are synchronized with the cloud.

4.1.3.8 *Synchronization in background*

In the existing solutions by coresystems, the synchronization with the cloud happens on a user command by pressing the “Synchronize” button. An optional requirement to the new MobileERP project is that the application synchronizes in the background, without waiting for the user to explicitly start a synchronization. So when there is an internet connection available, the data is always synchronized.

4.1.3.9 *Localization*

Since coresystems is a global player, their user base speaks a lot of different languages. Due to this fact, it is very important for a software product to provide users the possibility to choose from a selection of supported languages. In this project only a clarification of the possibility has to be done. The Results can be found in the appendix under section pre-study (see section 8.1.3).

4.1.4 Functional Requirements

The following section describes the functional requirements of the MobileERP app. The goal of this project was to make a prototype based on the new Windows 8 technology. Therefore only a subset of the general use cases, which are done in the other applications of coresystems, are realized. Those use cases are chosen to prove the feasibility.

The following use cases are desired

- UC 01
- UC 02
- UC 03
- UC 05

The following use cases are optional

- UC 04

4.1.4.1 *The Level of Details*

The use cases are all described in in a brief form and the difficult ones have been written fully dressed.

4.1.5 Brief Use Cases

4.1.5.1 *UC 01: Login*

The user starts the app and he/she fills out all the required parameters in the login screen.

4.1.5.2 *UC 02: Search for Business Partner*

The user has successfully logged into the application (UC 01). Now he/she searches for a business partner he/she is working for.

4.1.5.3 UC 03: Add New Customer Address

The user logs into the app (UC 01) and goes to the business partner to whom he/she wants to add new address. He/she changes to the edit mode and adds the address.

4.1.5.4 UC 04: Display Report

The user wants to see the revenue of the top 10 items of his company. Therefore he/she goes to the report module and chooses the report “Revenue by Item (Top 10)” in the section sales.

4.1.5.5 UC 05: Add New Business Partner

The user has successfully logged into the application (UC 01). Now he/she wants to add a new business partner. Therefore the user goes to the Business Partner module and adds a new business partner.

4.1.7 Fully Dressed Use Cases

4.1.7.1 UC 02: Search for a Business Partner

What	*Description*
Use Case ID	02
Primary Actor	Sales person | Consultant
Level	User goal
Stakeholder & Interests	• The user wants to find a customer very quickly
Preconditions	• The user has logged into the mobile app • Data from the cloud has been synchronized with the mobile device • Module Business Partner is available
Post conditions	The actor has found the address of the specific customer

Main Success Scenario

1. The user sees the main view of the app
2. The user clicks on the module called Business Partners
3. A list of all business partners are displayed
4. He is looking for the customer “Nano Shop”, so he uses his fingers to zoom out and navigates to “N”
    a. The list zooms automatically back in when he presses the “N” button
5. The user chooses the customer (Nano Shop)
6. He navigates to the section called All Addresses and clicks on the address he is looking for
7. All details about the address are displayed

Extensions

3a. The list is short, therefore the user navigates directly to the customer

1. Go to step 6

4a. Customer cannot be found in the list

1. The customer has to be created, see UC 04

6a. No address is linked to the customer

1. Click Add Address
2. Fill out the form with all required information

Special Requirements

- Touch friendly User Interface
- Short navigation path to the address

Technology and Data Variation List

- Finding Address on map

Frequency of Occurrence

Consultant: Often, because he uses this use case to find the address of his next customer

Salesperson: Rarely, only when a new customer is acquired

Open Issues

- How to validate the Input? What are the rules?
- Who is the primary actor?

4.1.7.2 UC 03: Add New Customer Address

What	*Description*
Use Case ID	03
Primary Actor	Sales person
Level	User
Stakeholder & Interests	• The user wants to add another customer to their branch office
Preconditions	• The user has logged into the mobile app • Data from the cloud has been synchronized with the mobile device • Module Business Partner is available
Post conditions	• New customer has been created and saved to the local database and is successfully synchronized with the cloud

Main Success Scenario

1. The user sees the main view of the app
2. The user clicks on the module called Business Partners
3. A list of all business partners is displayed
4. He swipes from the bottom of the tablet upwards. The app bar with a “New” button is shown
5. He clicks on the button and the add/edit business partner view is opened
6. In this view the user fills out all the information he knows about the new customer
7. He clicks on save and the new customer is saved

Extensions

4a. The user does not have permission to add new customers

1. Let the user know about the restrictions

6a. Validation of one or more fields failed

1. Mark the correspondent field with a sign that shows the user that there is something wrong with his input

7a. Synchronization process fails

1. An error message is displayed which shows the user that something in the saving process failed

Special Requirements

- Touch friendly User Interface
- Windows 8 Style Guide conform

Frequency of Occurrence

- Several times a week

Open Issues

- How to validate the input strings?

4.1.7.3 UC 04: Display Report

What	*Description*
Use Case ID	04
Primary Actor	Sales person
Level	User
Stakeholder & Interests	The user wants to get an overview about the annual sales
Preconditions	• The user has logged into the mobile app • Data from the cloud has been synchronized with the mobile device • The user has permission to use the report module
Post conditions	-

Main Success Scenario

8. The user sees the main view of the app
9. The user clicks on the module called Report
10. A list of all possible reports shows up; divided into different sections
11. He chooses the report “Revenue by Item (Top 10)” in the section “Sales”
12. A diagram with the percentage sales of the top ten products is displayed
    a. Next to the chart is a legend containing the exact values of each item

Extensions

3a. No Report can be generated

1. An Error is displayed, which informs the user that no data is available to generate a chart

Special Requirements

- Touch friendly UI
- Easy to read

Technology and Data Variation List

-

Frequency of Occurrence

- Once a week

Open Issues

- How to generate the different charts?
- Is there a framework available for generating different charts?

4.2 Graphical User Interface

This section describes the development of the graphical user interface and the resulting paper prototype. After studying the Windows 8 user experience guidelines, the most recommendations described therein have been considered. More details about our implementation will follow in the subsection graphical user interface in the section architecture.

4.2.1 Paper prototype

The coresuite app for the iPad has been used as sample for developing the paper prototype and the resulting user interface for Mobile ERP, or more precisely there has been used it to figure out which information needs to be displayed on which view especially for the detail views.

4.2.1.1 Navigation design

For the navigation we have chosen the Windows Store “style Hub navigation pattern”. This pattern is best for apps with large content collections or many distinct sections of content for a user to explore. [1, p. 15]

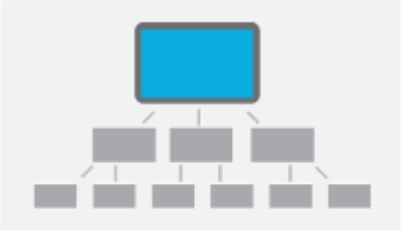


Figure 1: Hierarchical navigation

4.2.1.2 HUB – Home Screen

This is the user’s entry point to the app. After starting MobileERP from the Windows start screen and entering the login data, not shown in the paper prototype, the home screen, respectively the hub, is displayed. There are two groups, “Modules” and “Alerts” which are shown within tiles. Modules are the core components of MobileERP. As shown in the picture below, we wanted to demonstrate the rich horizontally panning view that shows more information by swiping from right to left and allowing users to get a glimpse of what’s available.

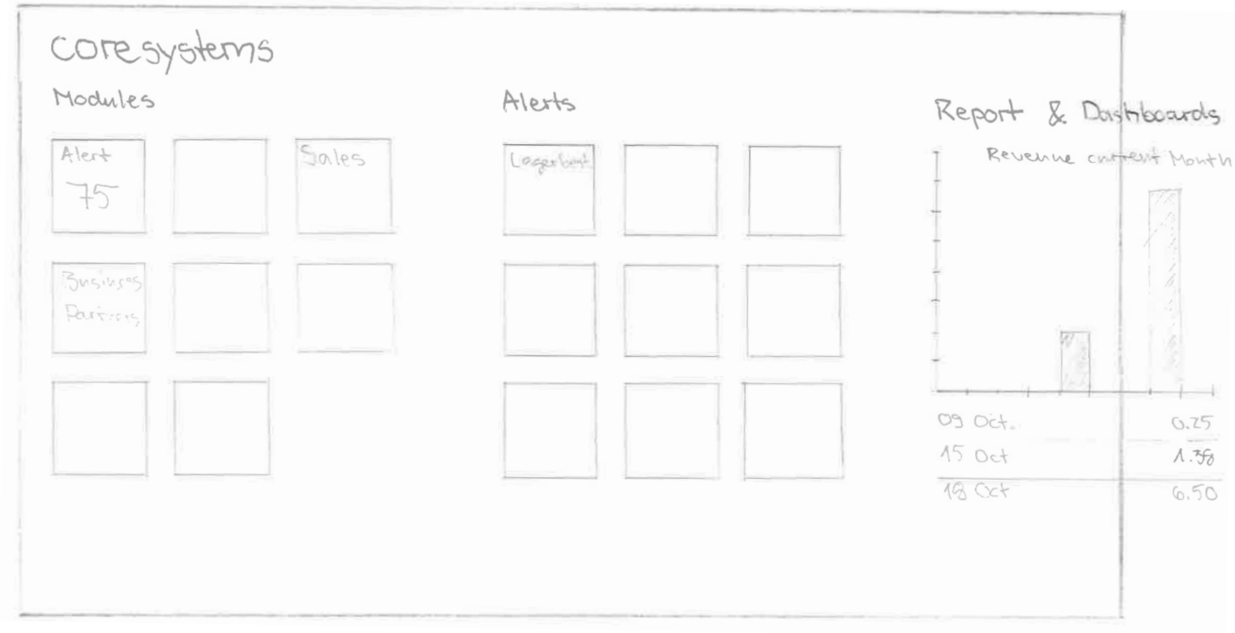


Figure 2: Home Screen Page

4.2.1.3 Section Page – Business Partner Page

This page is the second level of the app and represents the module “Business Partners”. All partners are displayed and each of them has its own detail page, as described below. The first design had tiles for favorite business partners, to be able to open quickly often used partners and a list with all partners which could be filtered. In the final design of MobileERP we discarded the list in favor of tiles that are grouped by characters. There is a detailed description about this page in the implementation section.

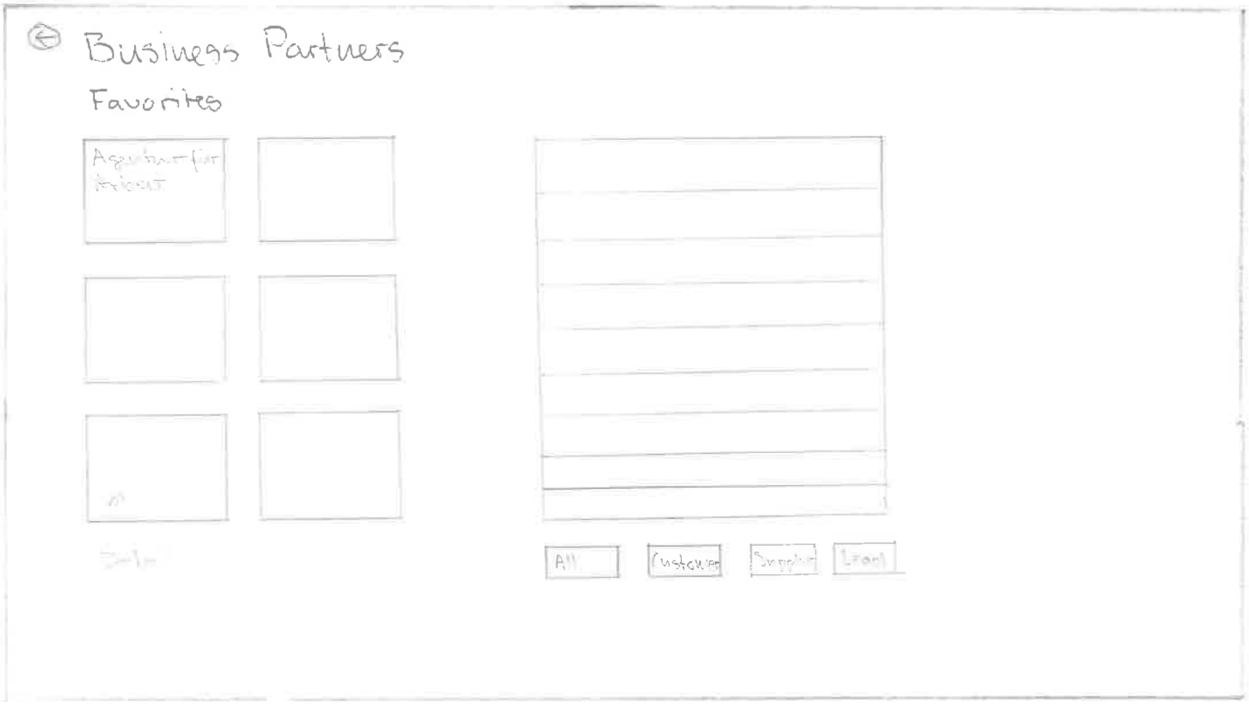
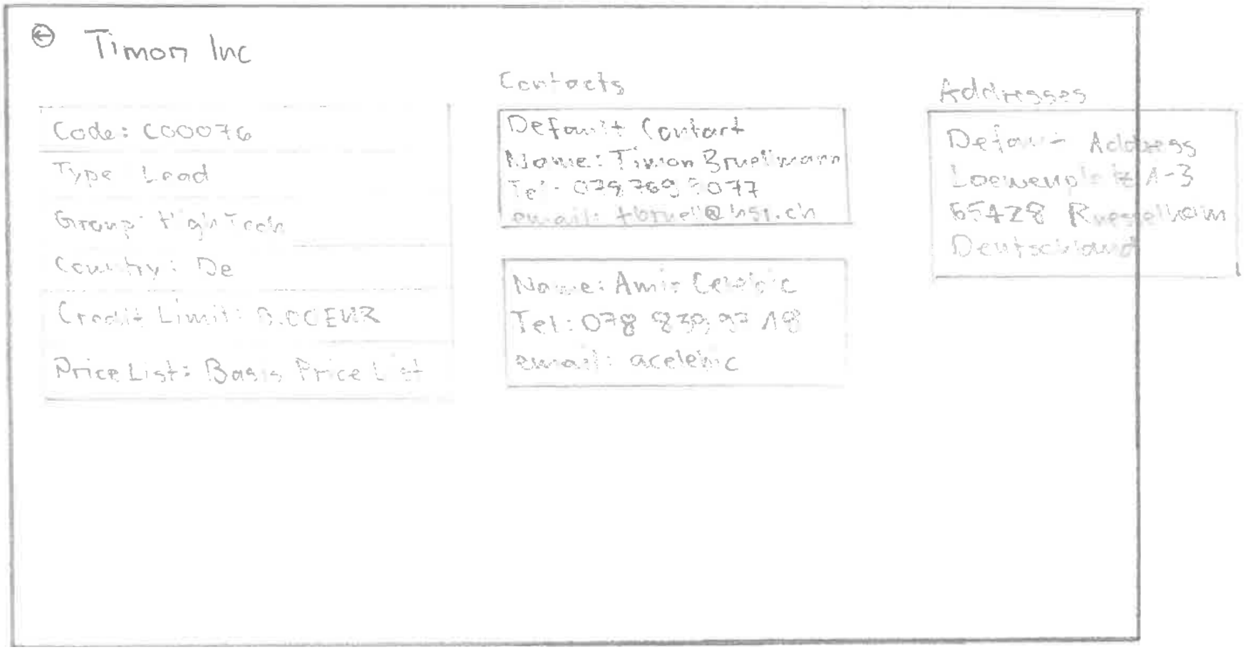


Figure 3: Paper prototype business partners overview

4.2.1.4 *Detail Page – Business Partners*

This page has all relevant information about a business partner that has been selected from the Section Page.



Timon Inc

Code: C00076

Type: Lead

Group: HighTech

Country: De

Credit Limit: 0.00EUR

Price List: Basis Price List

Contacts

Default Contact
Name: Timon Bruelmann
Tel: 079 769 3077
email: tbruel@hsr.ch

Name: Amir Celebic
Tel: 078 839 97 18
email: acelebic

Addresses

Default Address
Loewenplatz A-3
65428 Ruesselheim
Deutschland

Figure 4: Paper prototype business partner detail page

4.3 USED TECHNOLOGIES

4.3.1 Platform

- Windows 8 Professional

4.3.2 External Libraries

- Siaqodb [http://siaqodb.com/]
- JSON.NET [http://json.codeplex.com/]
- Bing Maps SDK for Windows Store apps [http://visualstudiogallery.msdn.microsoft.com/bb764f67-6b2c-4e14-b2d3-17477ae1eaca]

4.3.3 Development tools

- TFS Cloud Preview [http://tfspreview.com]
- Visual Studio 2012

4.3.4 Documentation

- Microsoft Office 2013
- Microsoft Visio 2010
- Astah Community [http://astah.net/editions/community]

5 ARCHITECTURE

This bachelor thesis was a software engineering project, which should deliver a stable and production-mature product (though a first version of the product). While it is not feature-complete, the architecture allows easy extendibility, follows the practices of low coupling and is generic in its core. This section should help to understand our main concepts and is also a guideline to add new modules to the MobileERP app.

5.1 OVERVIEW

5.1.1 Physical architecture

The following picture gives an overview over the coresystems infrastructure. Coresystems provides all necessary tools to bring the data from different enterprise resource planning software to mobile devices as well to browsers. Customers, which already have an ERP tool, can save their data into the cloud through an API from coresystems. On the other side, coresystems provides an API to connect to this data through mobile devices or other software. The goal of this project was to develop a mobile application that uses the same API like the existing apps, but which is running on Windows 8.

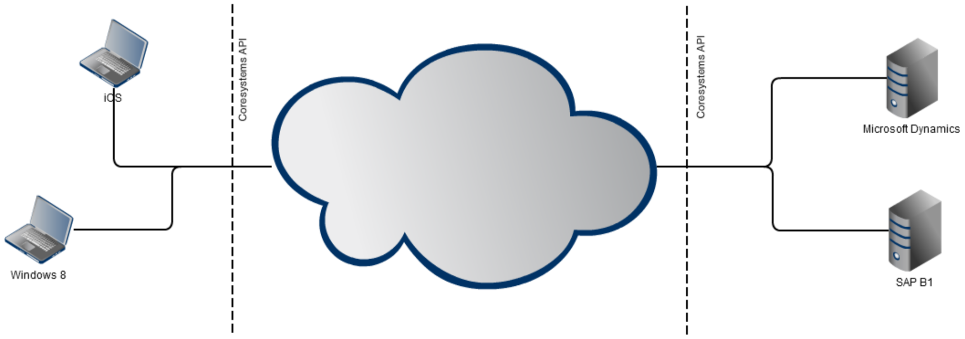


Figure 5: Coresystems existing architecture with a Windows 8 client

Figure 5 shows the overall topology of the coresystems business. On the right side are the common ERP solutions from third parties. Coresystems provides an interface to upload data, for example from SAP B1 or Microsoft Dynamics into the cloud of coresystems. On the other side mobile clients can access those data through a binary API or a JSON API. In this project a Windows Store app that uses the JSON-API from coresystems to exchange data has been realized.

5.1.2 Logical architecture

The MobileERP application is based on a three layer architecture. All Layers below the view layer are implemented in a separate project, called MobileERP. The advantage of this lies in the portability of the application. Each UI implementation is based on the same business logic, so that the Windows Store app or Windows Phone 8 app have no relations to each other. For additional UI projects developers do not have to implement any business logic.

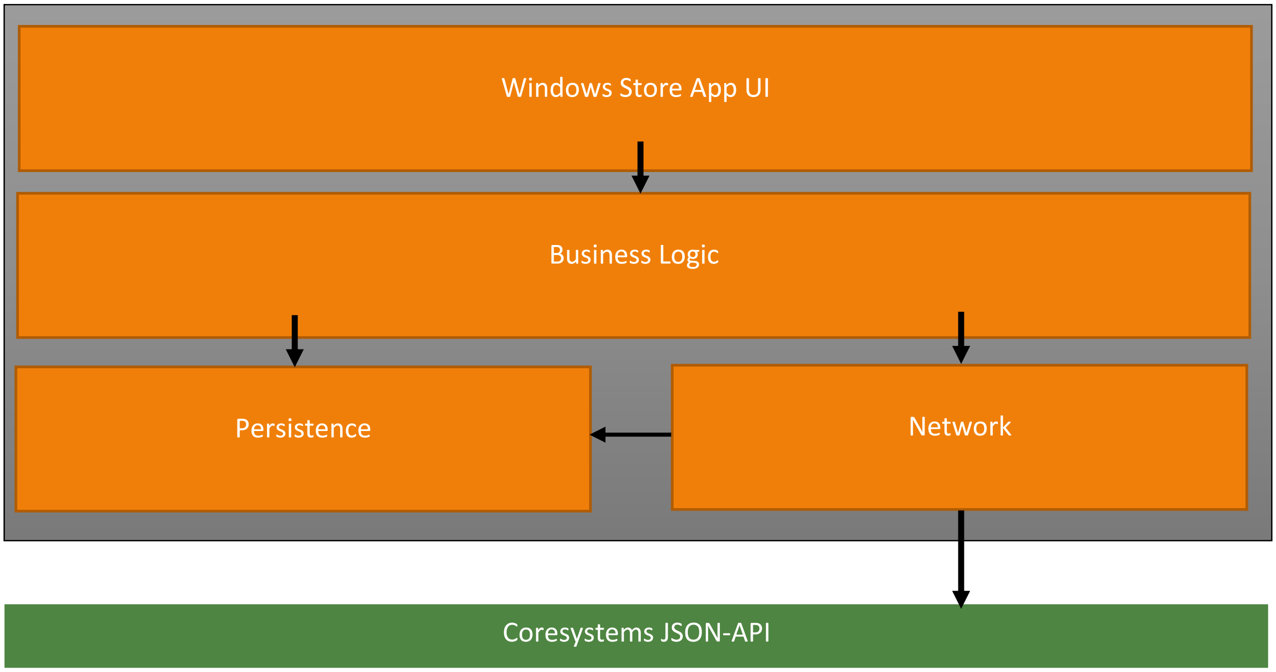


Figure 6: Logical architecture with connection to coresystems API

The Image above shows the layer architecture of the MobileERP application. All layers for the application - excluding the JSON-API of coresystems - have been developed from scratch in this project.

5.1.2.1 Windows Store App UI

For the UI, we closely followed Microsoft's MVVM pattern. Unfortunately Windows Store Apps do not show as much flexibility as standard WPF applications, therefore are some view functions are implemented in the code behind. The view models often enrich the user input, e.g. by adding a key to a new object or by setting the timestamp of a change.

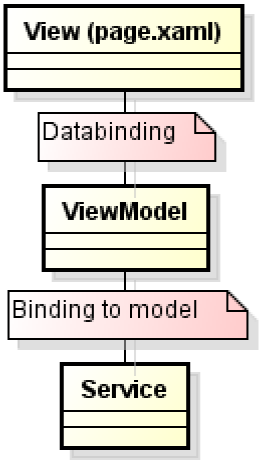
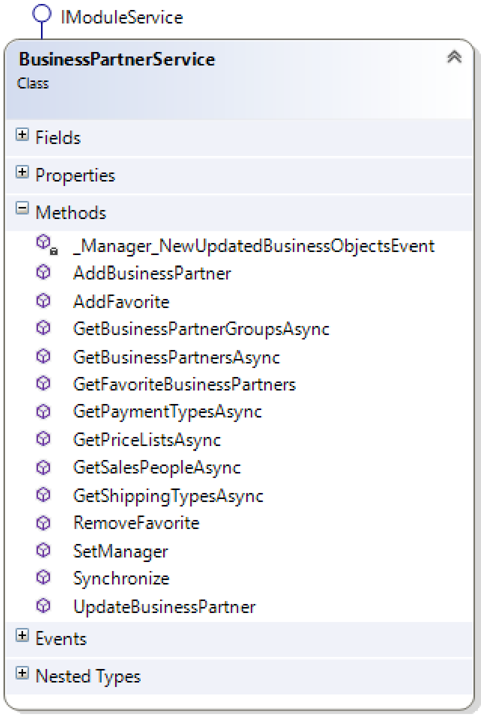


Figure 7: MVVM implementation

5.1.2.2 Business Logic

The business layer consists of business objects and services for logical operations. View models register a service through the ServiceHandler class and delegate their business logic to it.

The service knows the manager, which is found in the business layer. The manager controls synchronization and persistence. The BusinessPartnerService is rich with functions and should serve as a good example:



IModuleService

BusinessPartnerService
Class

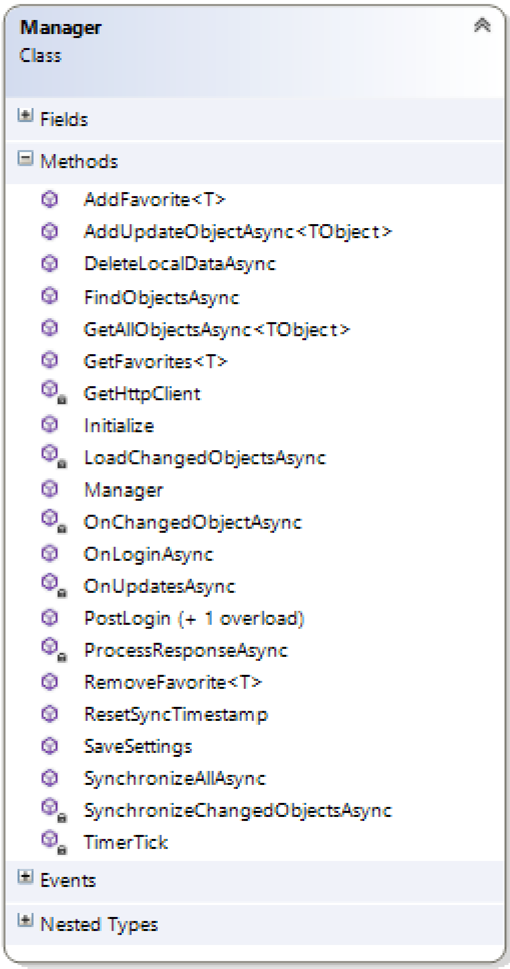
⊞ Fields
⊞ Properties
⊟ Methods
_Manager_NewUpdatedBusinessObjectsEvent
AddBusinessPartner
AddFavorite
GetBusinessPartnerGroupsAsync
GetBusinessPartnersAsync
GetFavoriteBusinessPartners
GetPaymentTypesAsync
GetPriceListsAsync
GetSalesPeopleAsync
GetShippingTypesAsync
RemoveFavorite
SetManager
Synchronize
UpdateBusinessPartner
⊞ Events
⊞ Nested Types

Figure 8: BusinessPartnerService class overview

As shown in figure 8, this class consists of multiple get-methods to deliver the requested data to the view model. Moreover, it provides methods for CRUD operations, which it delegates to the generic core, i.e., the manager class

Manager

The manager acts as a controller which works with all business types and has a high flexibility. Therefore, common access points for the services have been introduced. Only the manager knows the communication and persistence packages and controls the synchronization.



Manager
Class

⊞ Fields

⊟ Methods

- AddFavorite<T>
- AddUpdateObjectAsync<TObject>
- DeleteLocalDataAsync
- FindObjectsAsync
- GetAllObjectsAsync<TObject>
- GetFavorites<T>
- GetHttpClient
- Initialize
- LoadChangedObjectsAsync
- Manager
- OnChangedObjectAsync
- OnLoginAsync
- OnUpdatesAsync
- PostLogin (+ 1 overload)
- ProcessResponseAsync
- RemoveFavorite<T>
- ResetSyncTimestamp
- SaveSettings
- SynchronizeAllAsync
- SynchronizeChangedObjectsAsync
- TimerTick

⊞ Events

⊞ Nested Types

Figure 9: Manager class overview

All CRUD operations in the manager are generic, so that every business object of the base type "BaseEntity" can be managed. The manager also controls the flow of the application. It synchronizes, persists and fires events for updates.

Most functions are async, so that the UI-thread never blocks. This is useful for a good user experience.

Business Objects

The Mobile ERP application uses another domain model than the one coresystems provides. Therefore, all business objects have a sibling which matches the coresystems data format. A result of this parallelism, all object has to be converted when sending and receiving an object.

The main difference between the models is that in coresystems' data transfer objects (DTO), the references are saved in the referenced object. Taking the example of a person having an address, the address in the coresystem model contains the key and type of his parent object, while in our model, the person knows its address.

It has been decided to use an own model, so that it is normalized and for better use in the UI part. This leads to the independency of coresystems' DTOs.

5.1.2.3 Persistence

The data has to be persisted correctly. When a device loses the internet connection, it must be possible to continue working offline. This is why all the objects are always persisted in a local database on the mobile device (see section 5.5).

5.1.2.4 Network

The network component is used for all data exchanges during the lifetime of the app. The central access point to the network is the CloudConnection class. The communication with the cloud is handled by the SyncAPIWithJson class. This class uses a standard HttpClient provided by the .NET framework to send the HTTP requests to the cloud. The POST command of the HTTP protocol is used to send a request to the cloud. On top of the HTTP protocol the sync protocol, implemented by coresystems, is used. The content of a request is represented in JSON format. The data structure of the communication is defined in the sync protocol see section 8.3.

Sync Protocol

This protocol has been developed by coresystems and uses the JSON data format to exchange them between the frontend applications and the cloud. For more information about the structure see section 8.3.

5.1.2.5 Multi-Threaded Architecture

To avoid performance bottlenecks in the responsiveness of the MobileERP application, the async framework is strongly used. The Microsoft .NET 4.5 framework has introduced a simplified model of asynchronous programming based on so-called async methods (partially asynchronous) and await statements (awaiting async method completion). The compiler automatically decomposes the code in a "continuation-style" logic where the method remainder after await is transferred in a completion callback executed after the await asynchronous method.

Unfortunately this also brought some problems, since the flow of the operations cannot be controlled. To prevent synchronization problems, like race conditions, a BlockingCollection has been introduced for synchronization with the cloud (see section 5.3.18)

5.1.3 Packaging

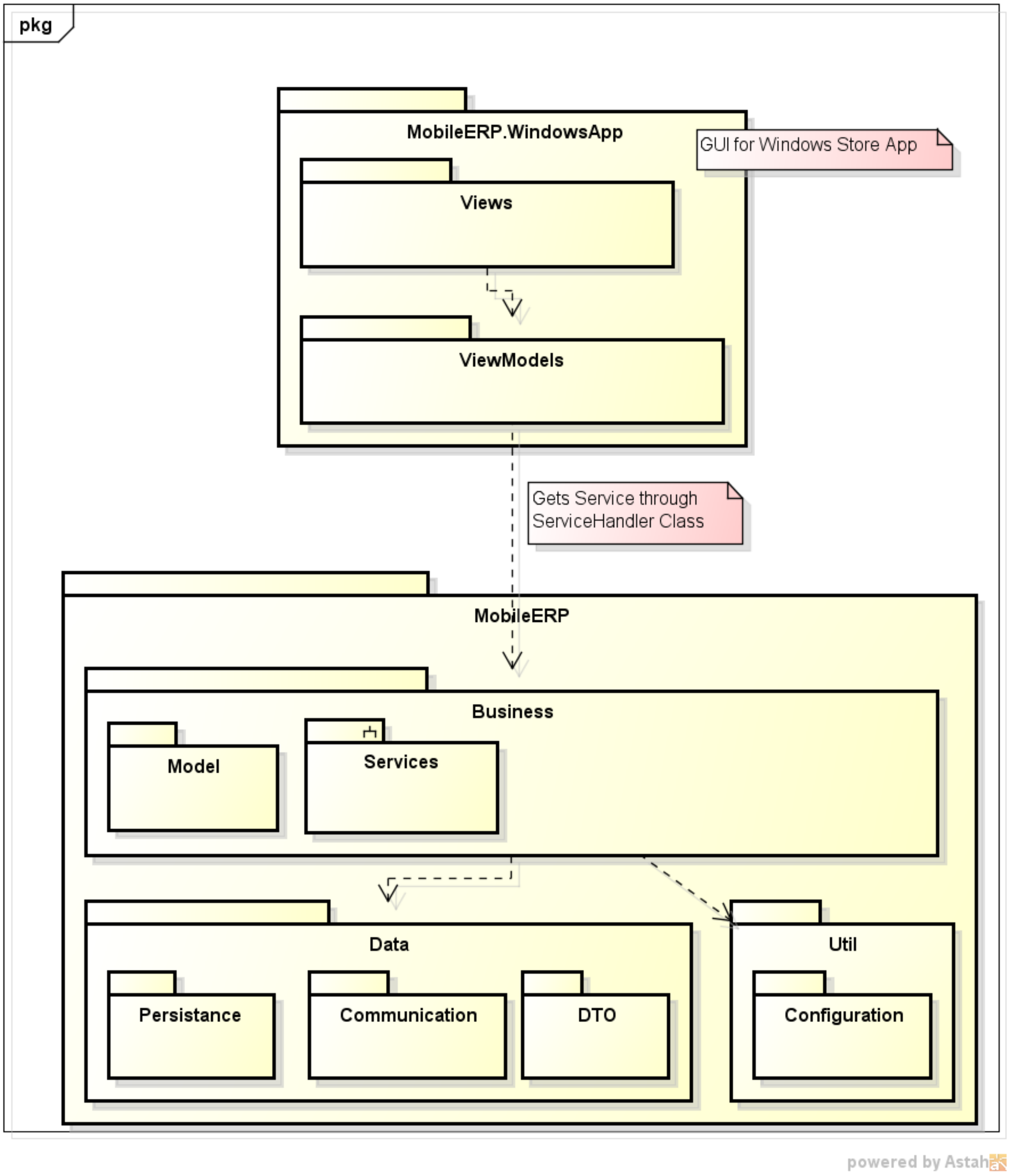


Figure 10: MobileERP solution packaging

To separate the business logic from the UI, two projects have been created. The MobileERP project is a Class Library and generates a DLL-file that is used by the MobileERP.WindowsApp project. The reason is a simple extensibility for UI developers, so that a Windows Phone 8 app can be easily added.

5.2 GRAPHICAL USER INTERFACE

5.2.1 Tiles

5.2.1.1 Overview

Since tiles are a core element of Windows Store Apps and Windows 8 in general, someone would think there is an easy, standardized way of using them in an application. So it came as a surprise that there is no simple way for using tiles, which meets normal demands.

That is why a specific model for showing business objects as tiles has been introduced. It is built so that it works with all the standard styles.

5.2.1.2 Implementation

Model

To build the tiles there is the BusinessTileDecorator class, which takes a business object as parameter and fills the required fields of a tile depending on the object's type.

```
switch (businessObject.GetType().Name)
{
    case "Address":
        Address a = businessObject as Address;
        this.UniqueId = a.UID;
        this.Title = a.City;
        this.Subtitle = a.Street;
        this.SetImage("Images/address512.png");
        break;
```

This BusinessTileDecorator objects can be added to an observable collection and then bound to a grid.

```
private ObservableCollection<BusinessTileDecorator> _addressTiles;
public ObservableCollection<BusinessTileDecorator> AddressTiles
{
    get
    {
        if (_addressTiles == null)
        {
            _addressTiles = new ObservableCollection<BusinessTileDecorator>();
            if (BusinessPartner.Addresses != null && BusinessPartner.Addresses.Count > 0)
            {
                foreach (Address a in BusinessPartner.Addresses)
                    _addressTiles.Add(new BusinessTileDecorator(a));
            }
        }
        return _addressTiles;
    }
}
```

XAML

In the XAML code a grid view can be added, which is bound to the collection and standard style for a tile can be selected or defined in a template.

```
<GridView
x:Name="addressesGridView"
```

```
ItemsSource="{Binding AddressTiles}"
ItemTemplate="{StaticResource PicSubtitleText150x150ItemTemplate}">
</GridView>
```

Finally it looks like this:

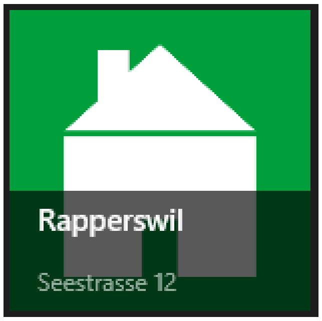


Figure 11: Address tile

Tiles Dictionary

A “ResourceDictionary” called TilesDictionary.xaml has been implemented, where all data templates for tiles reside. As can be seen in the xaml code above, there is a property “ItemTemplate” where add a tile as a “StaticResource” can be used.

5.2.2 The App Bar

5.2.2.1 Overview

The app bar is a new feature that comes with Windows Store apps. It contains transient access to commands relevant to a particular view [1, p. 19]. It is a context menu which appears when swiping from the bottom or top of the screen. The UX Guidelines for Windows 8 Store Apps recommend to use this new feature instead of a classical context menu of desktop Windows systems. All app bar items should act on the actual content of the view [1, p. 25].

5.2.2.2 Implementation

XAML

There is a property “BottomAppBar” in the Page class, with that an app bar can be added for a specific page, like in the XAML code below for the page BusinessPartnerModulePage.xaml. There are several buttons in the StandardStylies.xaml that come within a Windows Store App project and which can be added as a “StaticResource”. The corresponding click events and their operations are implemented in the code behind.

```
<Page.BottomAppBar>
       <AppBar x:Name="bottomAppBar" Padding="10,0,10,0">
           <Grid>
               <StackPanel Orientation="Horizontal" HorizontalAlignment="Left">
                   <Button Style="{StaticResource EditAppBarButtonStyle}"
                   Click="EditButton_Click"/>
                  <Button Style="{StaticResource RemoveAppBarButtonStyle}"/>
                   <Button Style="{StaticResource AddAppBarButtonStyle}"
                   Click="AddButton_Click"/>
                  <!--<Button Style="{StaticResource AddAppBarButtonStyle}"
                  Command="{Binding AddCommand}"/>-->
```

```
            </StackPanel>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Right">
                <Button Style="{StaticResource FavoriteAppBarButtonStyle}"
                Click="FavoriteButton_Click"/>
                <Button Style="{StaticResource RefreshAppBarButtonStyle}"/>
                <Button Style="{StaticResource HelpAppBarButtonStyle}"/>
            </StackPanel>
        </Grid>
    </AppBar>
</Page.BottomAppBar>
```

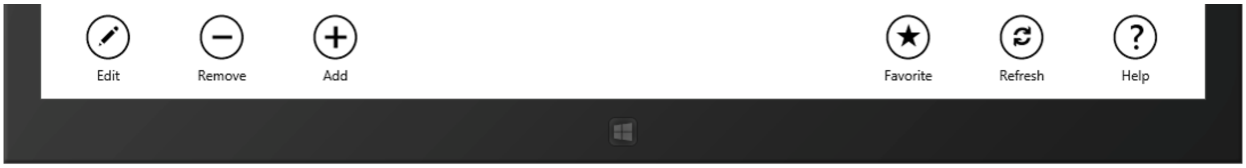


Figure 12: AppBar in business partners module page

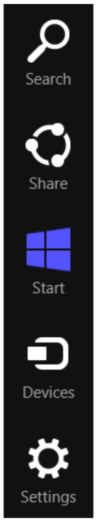
5.2.3 Charms

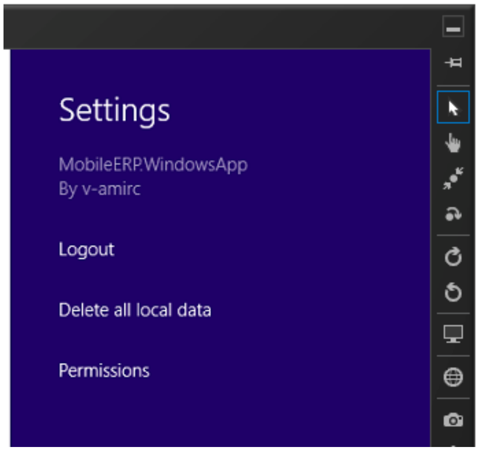
The charms are a set of buttons available in every app: search, share, connect, settings, and start. These should represent important actions that people want to complete in almost every app they use. [1, p. 10]

- **Search:** People can search for content located in MobileERP or in another app. There is an explanation below about the search implementation (see section 5.2.12).
- **Settings:** People can configure the app to their preferences.
- **Start:** People can go directly to the Start screen

Settings Implementation

To add application specific settings, commands need to be added in the App.xaml.cs class which is located in the namespace MobileERP.WindowsApp an comes within every Windows Store App project. This class contains the UtilityService object that provides methods for logout, delete all local data and permissions for MobileERP.





5.2.4 Page layout design

The layout design Microsoft recommends in the UX Windows 8 guidelines have been strictly followed. The following sub sections describe which of them were considered in the application.

5.2.4.1 App page header

In the grid system, the baseline of the app page header is 100 pixels from the top. The left margin for the page header is 120 pixels. The typography is SegoeUI Stylistic Set 20, light weight. [1, p. 29]

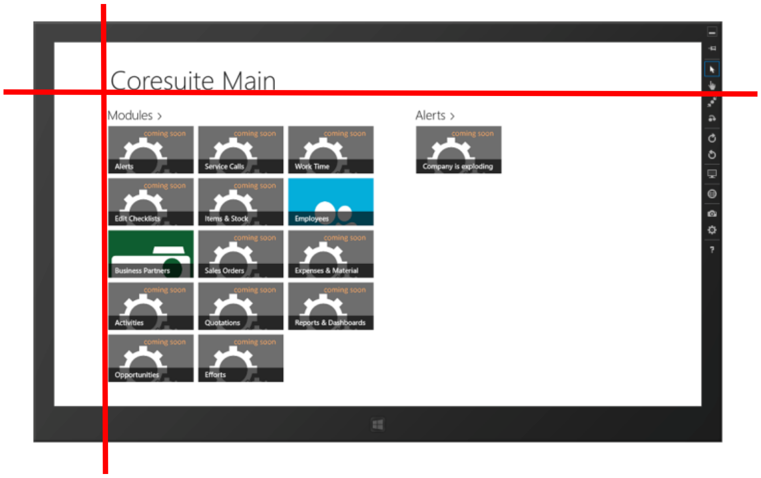


Figure 13: Margin of Windows 8 pages

5.2.4.2 Content region

In the grid system, the content region has a top margin of 140 pixels. The left margin is 120 pixels. The bottom margin is flexible. For horizontally panning content, it's between 50 pixels and 130 pixels. For vertically panning content, the top and left margins remain the same. There is no specified bottom margin because the content scrolls off the screen.

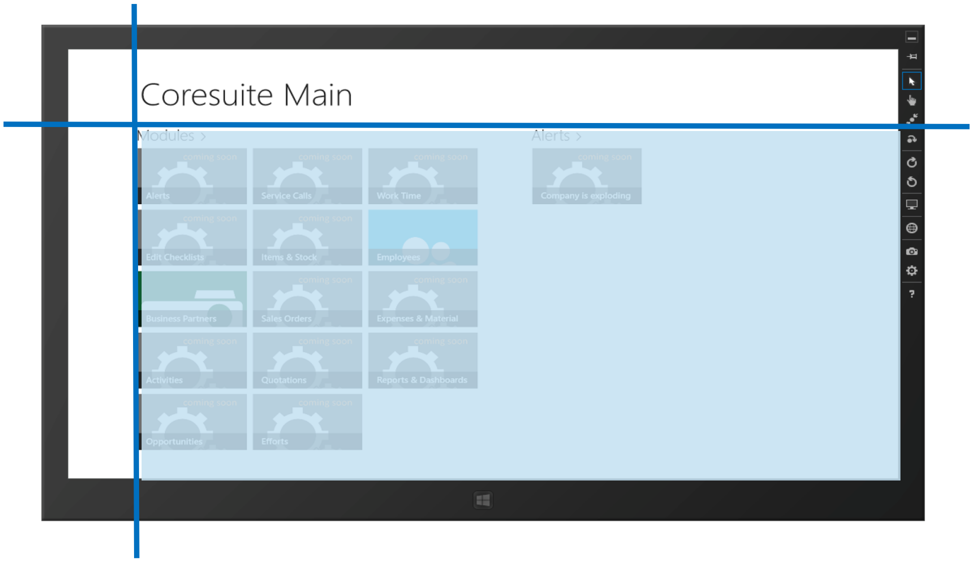


Figure 14: Content of Windows 8 pages

5.2.4.3 Horizontal padding

Horizontal padding between content items varies depending on the items. Hard-edged items like tiles on their own have 10 pixels of padding between columns. [1, p. 31]

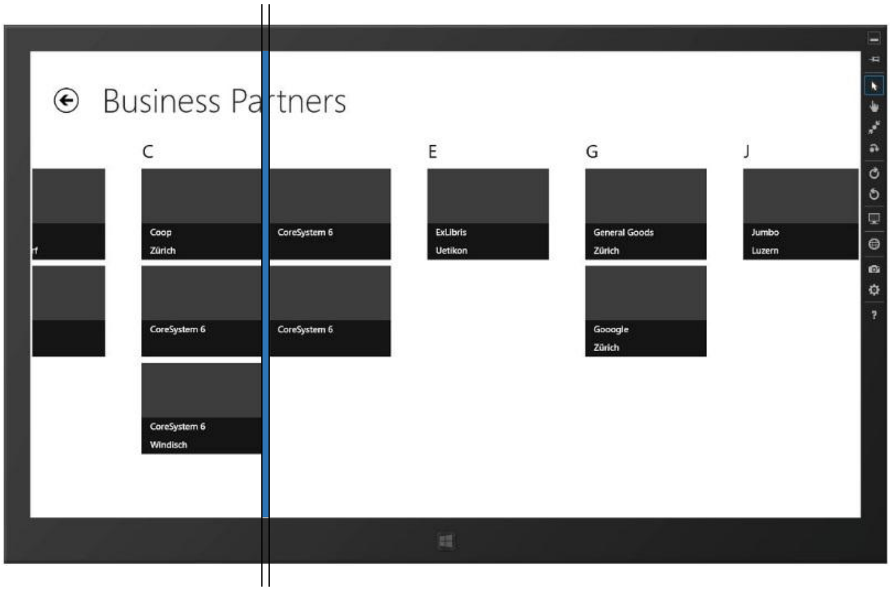


Figure 14: Horizontal padding between tiles

5.2.4.4 Vertical padding

Vertical padding between content items varies depending on the types of items. Hard-edged objects like tiles are separated by 10 pixels of padding between items in a row. [1, p. 32]

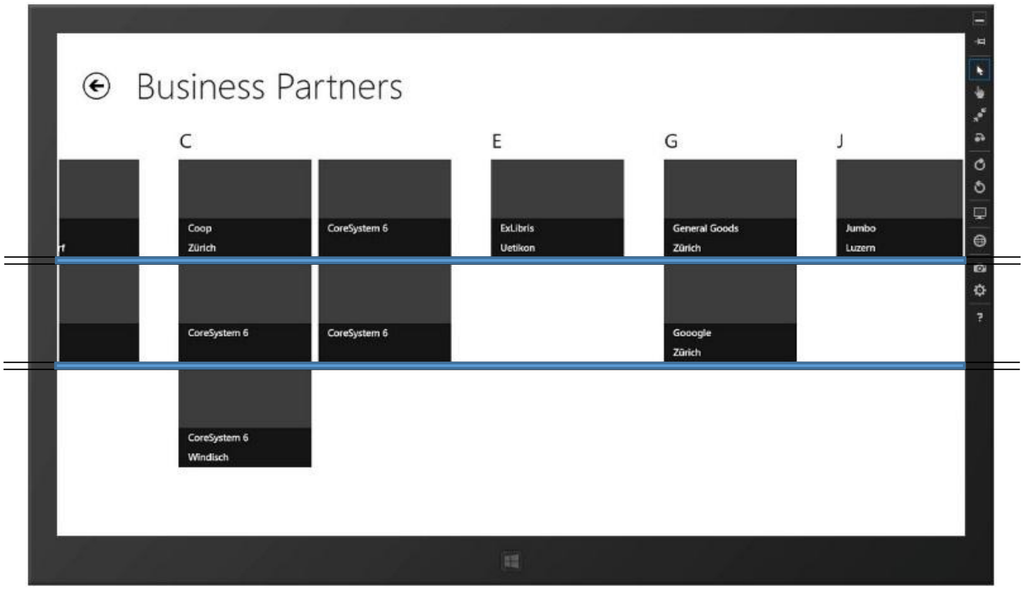
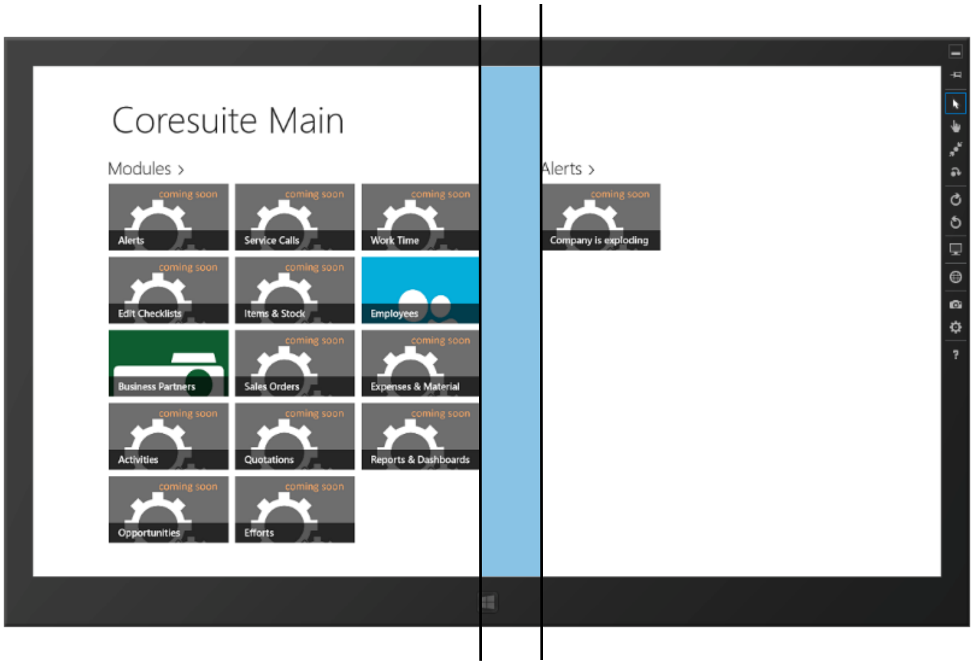


Figure 15: Vertical padding between tiles

5.2.4.5 Horizontal padding between groups

The padding between groups is a generous 80 pixels. This extra padding helps people easily distinguish one group from another, especially when panning across many groups. [1, p. 33]



The recommendations described above for the main page have been considered, because there are only two groups displayed. The module pages like BusinessPartnerModulePage or EmployeesModulePage have a large collection of tiles and therefore a padding of 50 pixels has been chosen. The reason was that more tiles can be displayed on a page.

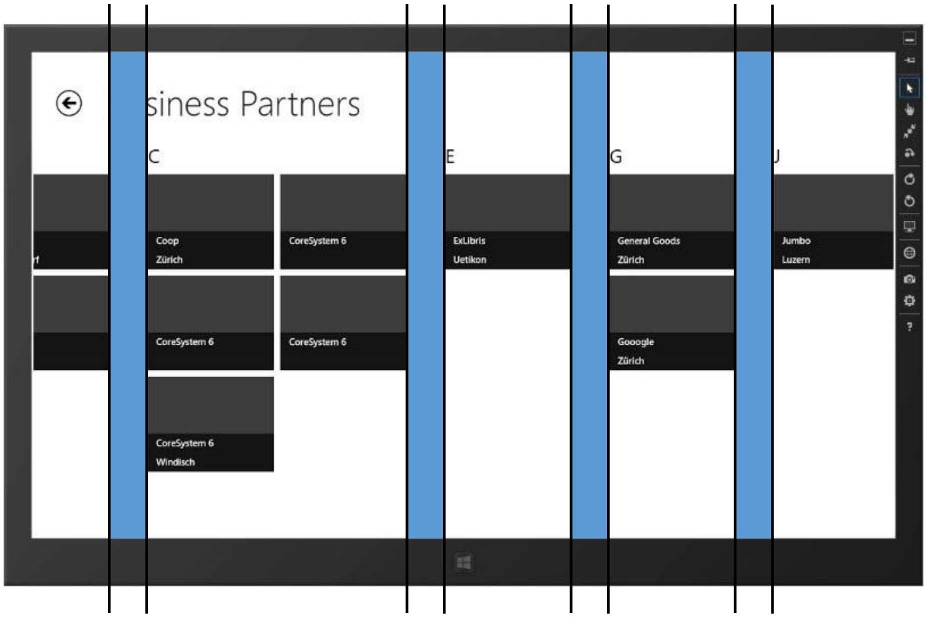


Figure 16: Padding more narrow

5.2.5 Class Design – ViewModels

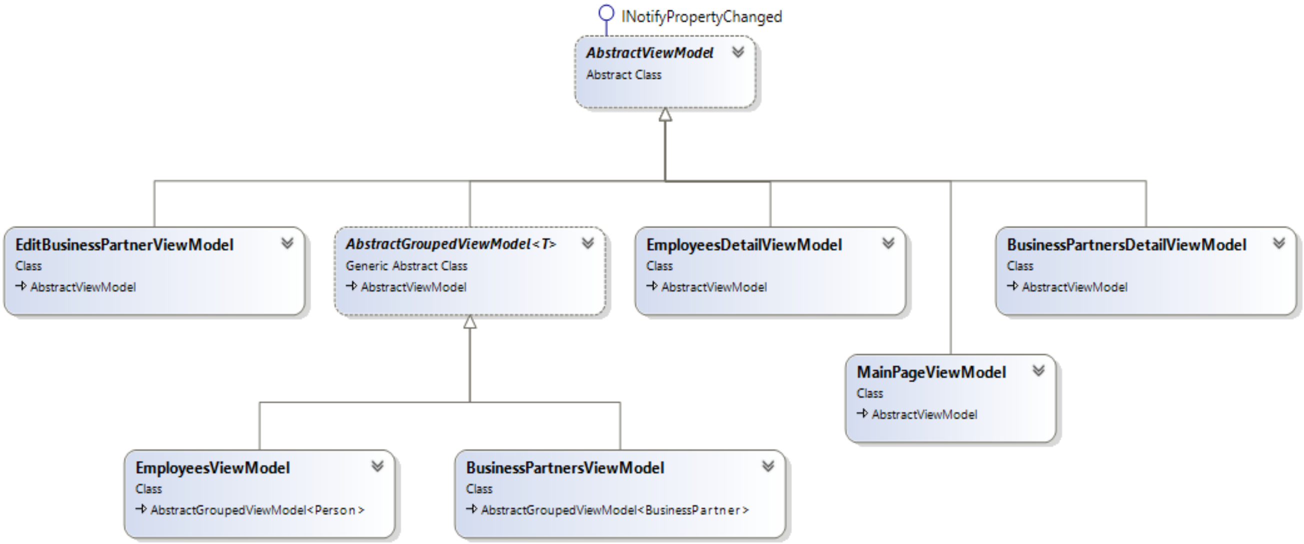


Figure 17: ViewModel design

5.2.5.1 Implementation

AbstractViewModel

This is the base class of all view models and contains a reference to the UtilityService, which is used for the exception handling and global commands.

AbstractGroupedViewModel

This class adds functionality for the module pages that use semantic zoom. It includes a sortable and observable collection. Groups are created with the first character of the ToString() method of a BaseEntity object. The only constraint is, that the object used for grouping implements IComparable. BaseEntities are added with the method AddBaseEntityToGroup() where the entities are added to existing groups or it creates a new group if needed.

Other view models are described within the page description in the sections below.

5.2.6 Class Design – Pages

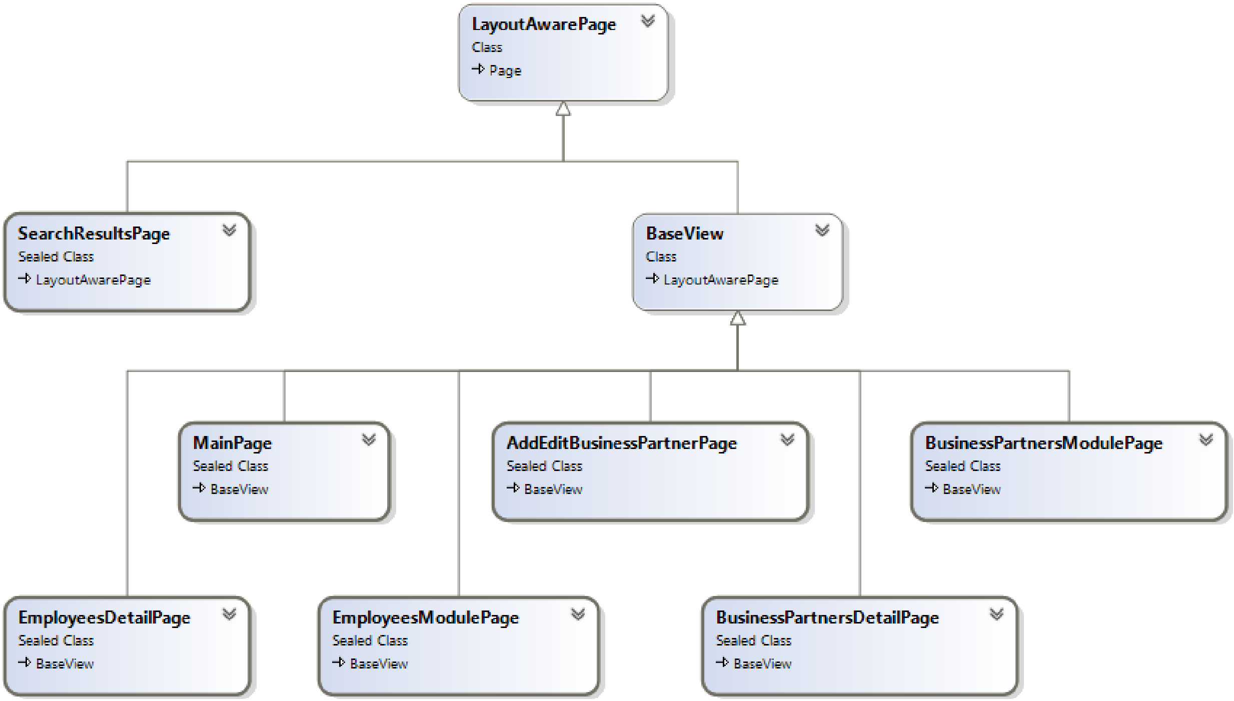


Figure 18: Pages design

5.2.6.1 Implementation

LayoutAwarePage

All Pages indirectly derive from this Page class - the standard class that comes within every Windows 8 app project. Basic controls and typical behavior of Windows 8 Store apps are provided, e.g. the "GoBack"/ "GoForward" button, the state management for navigation, process lifetime management as well as mouse and keyboard shortcuts for navigation.

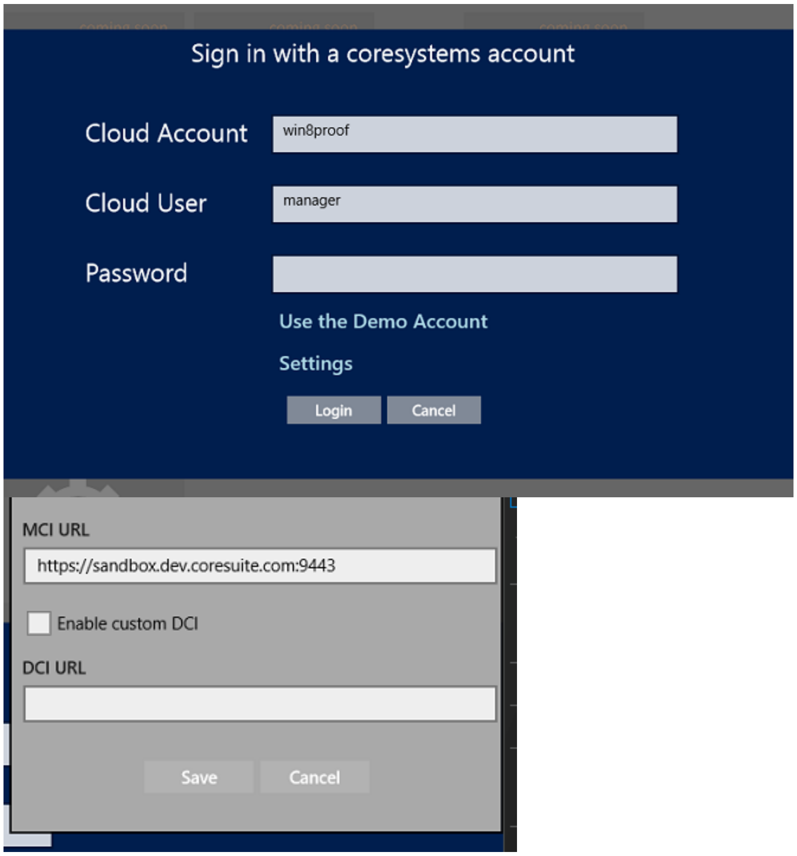
BaseView

This class has a utility service which is used for global commands and exception handling. All our pages derive from this class directly, except the SearchResultPage.

5.2.7 Login and Settings

When using Windows 8 apps, they often ask for Windows Live credentials to synchronize settings or profiles with the Windows Live account. To ensure a seamless user experience, our login follows the design of a Windows Live account login.

5.2.7.1 Overview



Sign in with a coresystems account

Cloud Account: win8proof

Cloud User: manager

Password:

MCI URL: https://sandbox.dev.coresuite.com:9443

□ Enable custom DCI

DCI URL:

Figure 19: Login screen and custom settings

5.2.7.2 Implementation

The login-screen is actually a Popup control, which is overlaid over the MainPage. When the user has not connected successfully the last time or when he uses the “Log out” command in the app, the Popup becomes enabled, while the module overview gets disabled and greyed out.

If the user presses “Login”, a request is sent to the coresystems cloud, which returns a user specific list of available companies to choose from. If this request is successful, the next page is shown where the user can select the company he wants to log in.

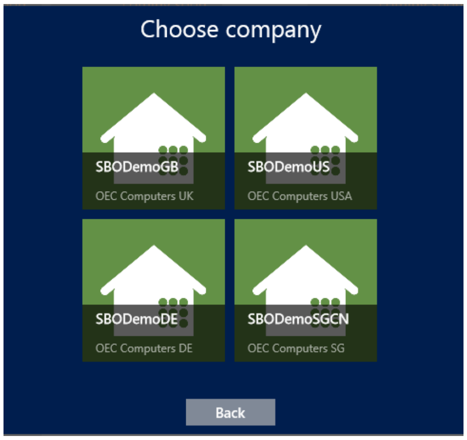


Figure 20: Popup to choose a company

5.2.8 Home Screen

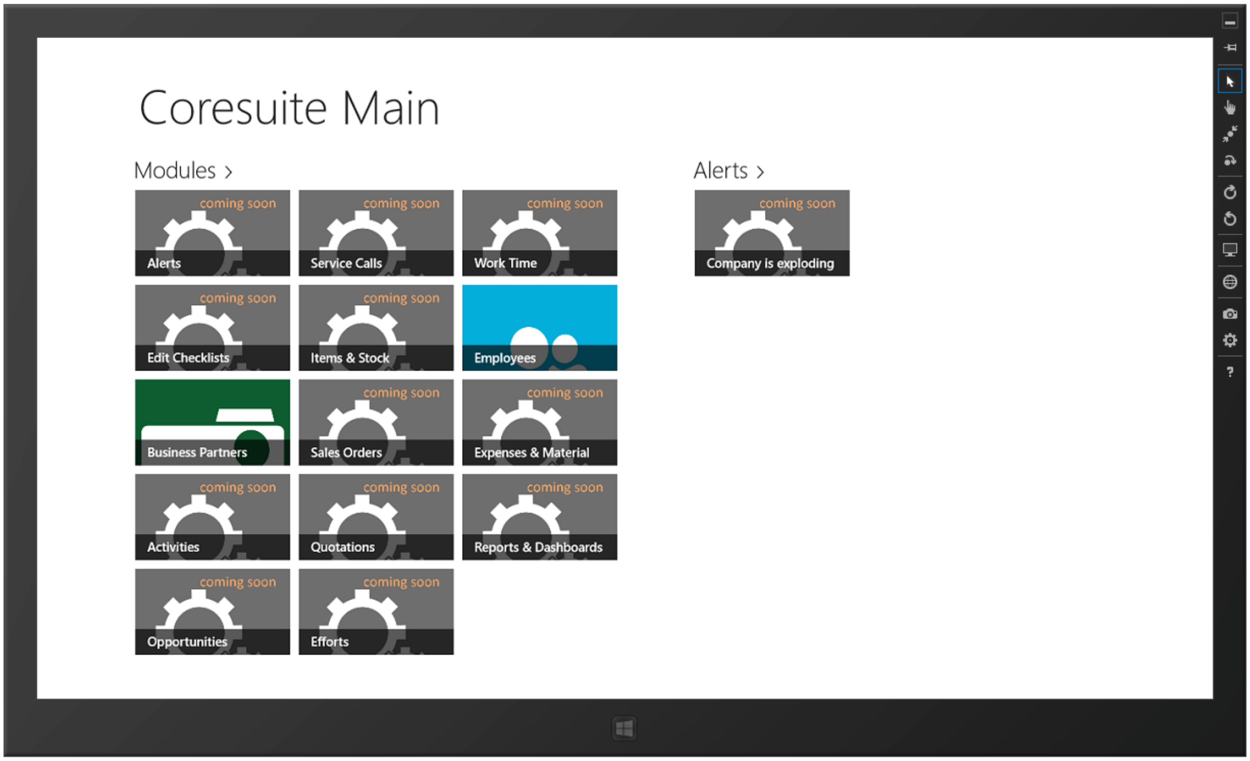


Figure 21: Homescreen with available modules

5.2.8.1 Overview

This is the entry page, if user credentials has been saved. There are two groups on the start screen (Hub) Modules and Alerts. Both groups adapt their arrangement automatically to the screen resolution.

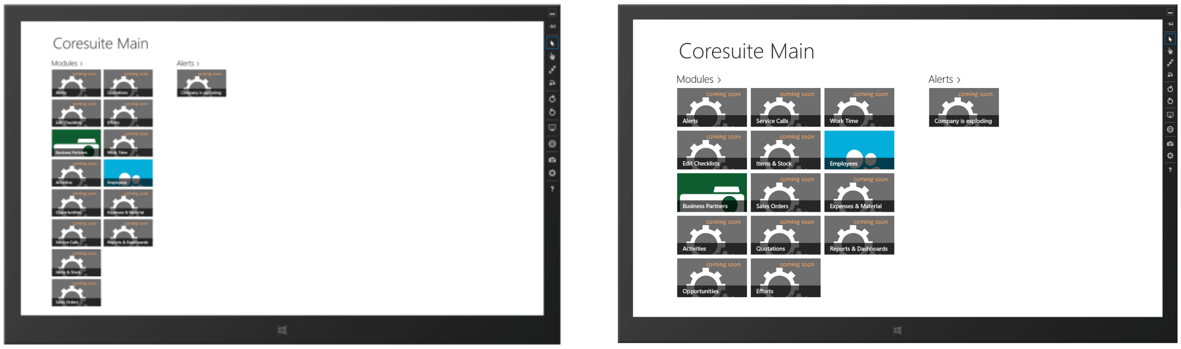


Figure 22: Differences of screen resolutions

5.2.8.2 Implementation

MainPageViewModel

The method AddModules() adds all Modules on the home screen. The number of Modules is always the same and therefore they are added statically with predefined modules.

First an object of “TileGroup” needs to be created, where “TileItems” or rather the Modules are stored. The size of shown TileItems must be set when creating a TileGroup. With the size, the number of TileItems that needs to be displayed on a page can be set.

```
private void AddModules()
{
    var modules = new TileGroup("Modules",
        "Modules",
        15, //Group size
        "Assets/SmallLogo.png",
         "Module Group");
    modules.Items.Add(new TileItem("AlertsModule",
         "Alerts",
         "Assets/SmallLogo.png",
         "Tile for Alerts Module",
         modules));
```

The group Alerts and its Items are added dynamically in contrast to the group modules. TileGroup and TileItems derive indirectly from the class BindableBase.cs which implements the INotifyPropertyChanged interface. Therefore, property changed events will be fired if an item changes or rather a new alert is created, which will then be added to the top of the group.

5.2.9 Module Business Partners

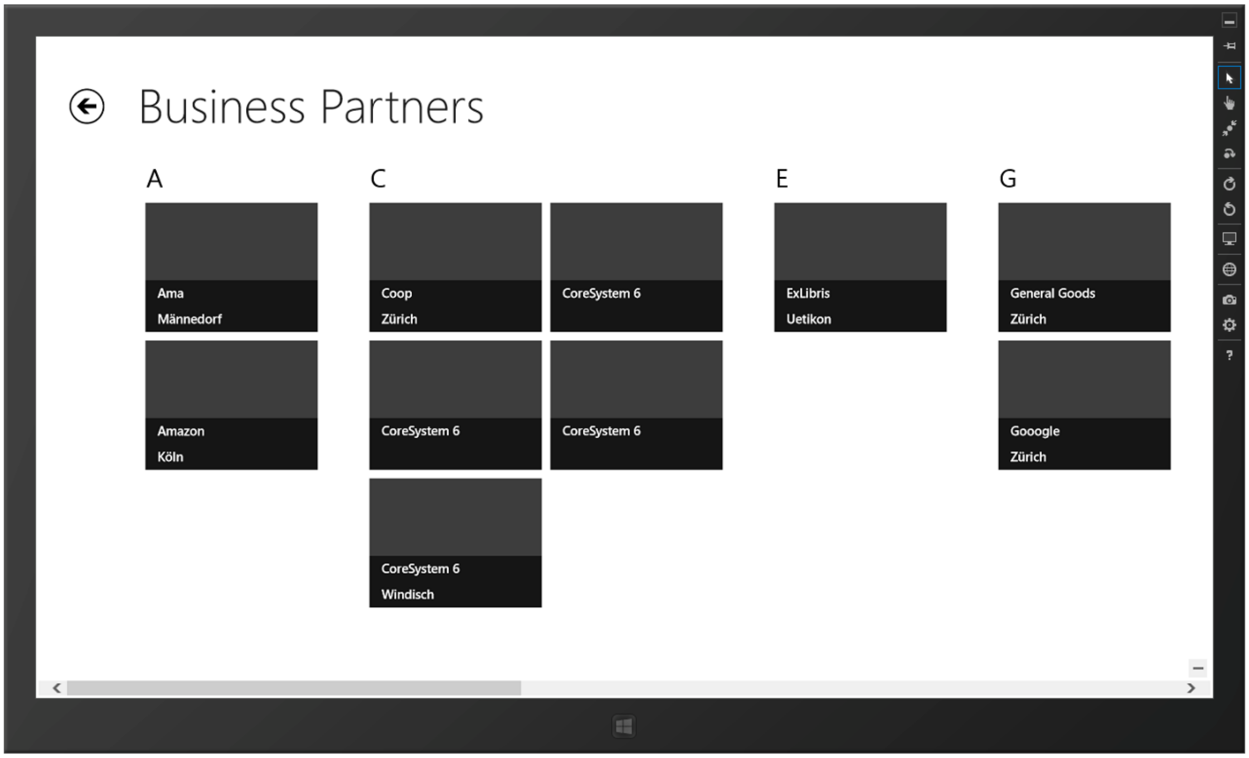


Figure 23: Business partners module page

5.2.9.1 Overview

A customer can have a large set of partners which are added dynamically to the page. These partners are separated into groups with the beginning character of their names. Switching between groups can be done by swiping horizontally or with “Semantic Zoom”. This is a feature that is provided by the newest .Net Framework for Windows Store apps, where all available groups are shown by clicking on the minus on the right corner with the mouse or by pinching with two fingers. The advantage is the possibility to scroll very fast in large collections by clicking or tapping on a character.

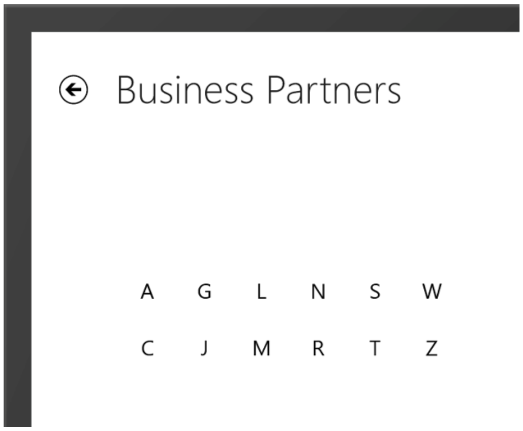


Figure 24: Zoomed-out view

5.2.9.2 Implementation

GroupedData<T>

This class inherits List<T> and implements the IComparable interface to compare data with its Key properties. Business partners with the same key are saved in an ascending list.

SortedObservableCollection

This class inherits from “ObservableCollection” and GroupedData added with the method AddGroup() will be sorted.

AbstractGroupedViewModel

This view model has a property GroupedBaseEntities that is a “SortedObservableCollection” and has a type “GroupedData<T>”. The method AddBaseEntityToGroup() adds an item to an existing GroupedData or creates a new one if no group with the same key has been created so far. This class can be reused for views, where grouping is needed or makes sense.

BusinessPartnersViewModel

This view model derives from AbstractGroupedViewModel given that business partners are grouped by their first character for their name.

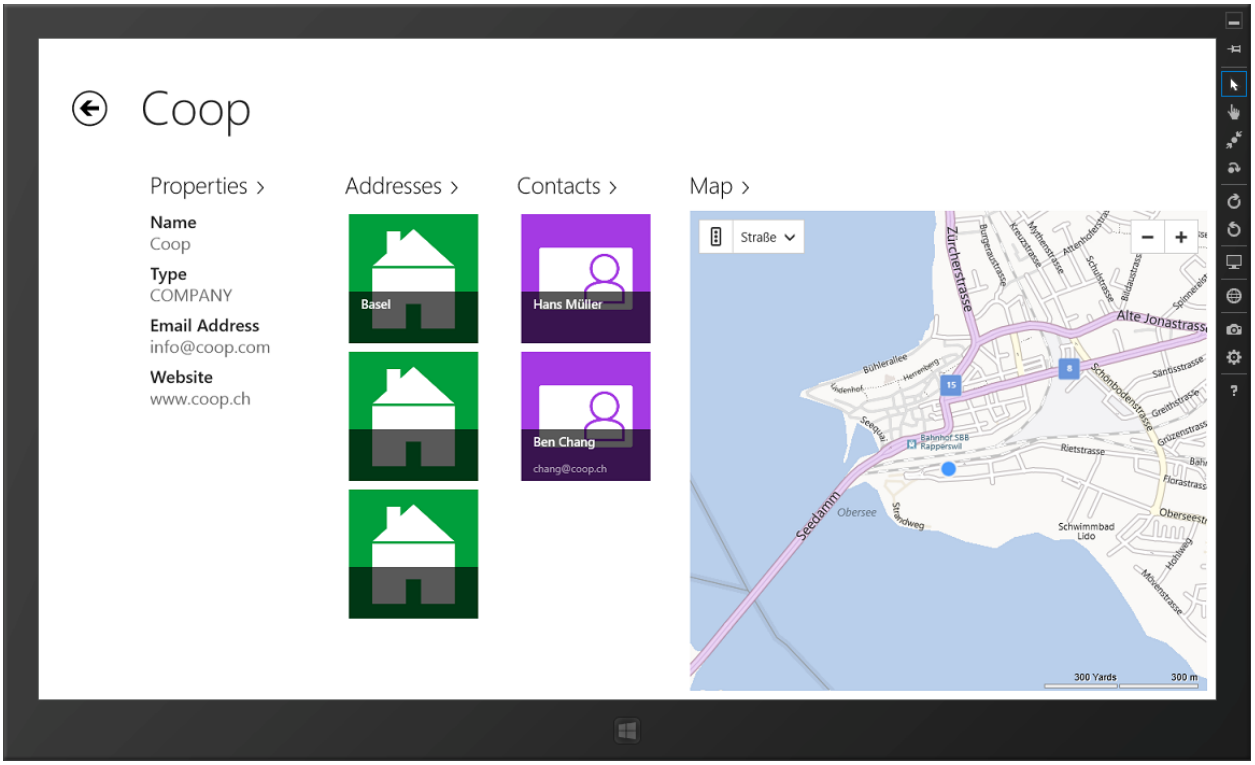
BusinessPartnersModulePage

SemanticZoom

Represents a scrollable control that incorporates two views that have a semantic relationship. For example, the ZoomedOutView might be an index of titles, and the ZoomedInView might include details and summaries for each of the title entries. Views can be changed using zoom or other interactions. [2]

```
<SemanticZoom ...>
  <SemanticZoom.ZoomedOutView>
    zoomedOutViewContent
  </SemanticZoom.ZoomedOutView>
  <SemanticZoom.ZoomedInView>
    zoomedInViewContent
  </SemanticZoom.ZoomedInView>
</SemanticZoom>
```

5.2.10 Business Partner Detail Page



Coop

Properties ›

Name
Coop

Type
COMPANY

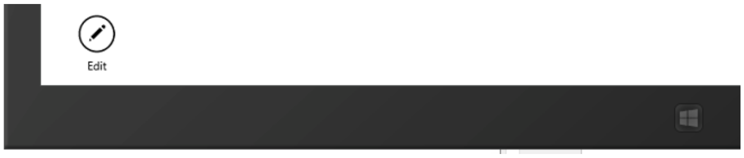
Email Address
info@coop.com

Website
www.coop.ch

Figure 25: Business partners detail page

5.2.10.1 Overview

On a detail page all relevant information are displayed in groups like properties, addresses, contacts and a map. If a user swipes from the bottom into the screen or right-clicks on the screen, the "App Bar" appears. By tapping or clicking the edit button the Add/Edit Page is opening which is explained in the next section.



5.2.10.2 Implementation

BusinessPartnerDetailViewModel

For each property of business partner exists a corresponding boolean property in the view model. The view models property begins with Is and ends with Valid, e.g. IsNameValid. The view verifies if there is an existing property and shows only the details that are available.

BusinessPartnerDetailPage

In the code behind of the xaml page class a method has been implemented FillContactControl() that adds only available properties of a business partner to the page and which are verified with the Is-Valid methods described above.

Map

To include Bing Maps in detail view, a tutorial on codeproject [3] has been followed. To use Bing Maps, a developer account has to be created. After that it is no problem to include the map in the XAML part of the page.

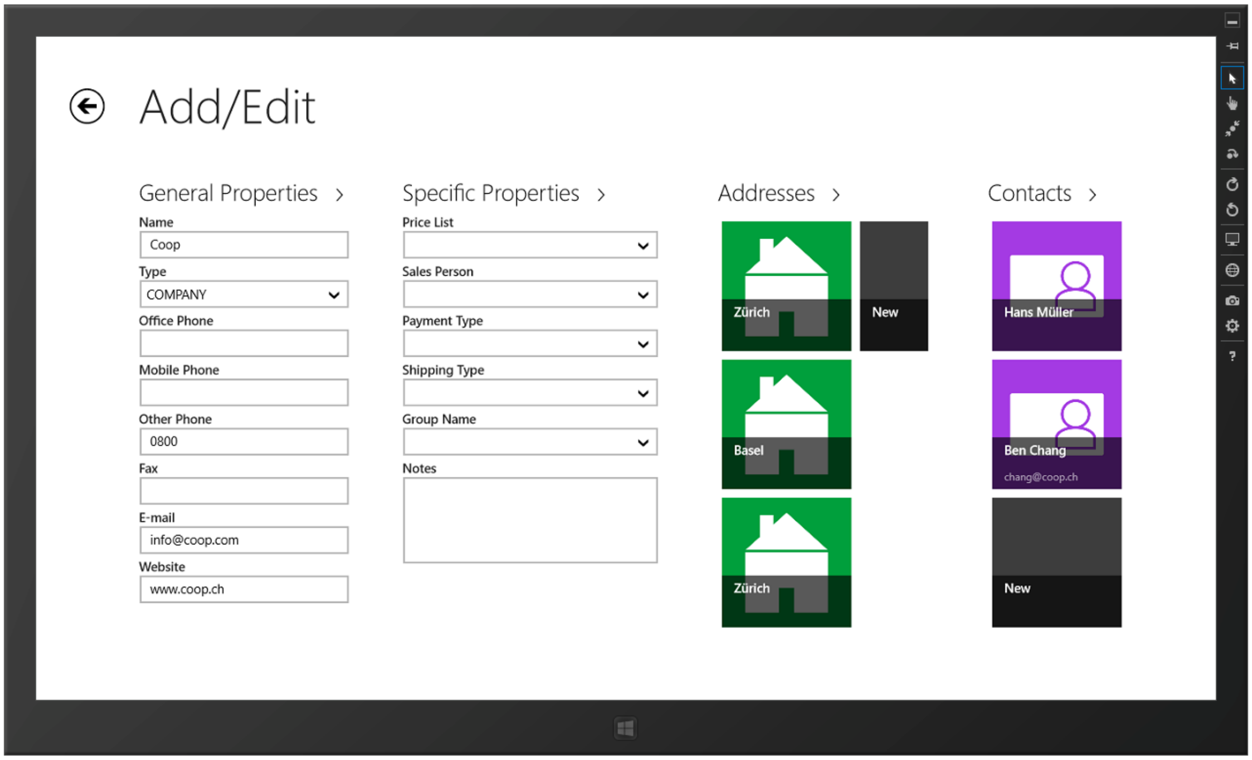
```
<bm:Map ZoomLevel="15" Credentials="<CREDENTIALS>” Grid.Row="1" Margin="10,10,0,0"
Width="600" Height="600">
    <bm:Map.Center>
        <bm:Location Latitude="47.2243186" Longitude="8.8186672" />
    </bm:Map.Center>

    <bm:Map.Children>
        <bm:Pushpin x:Name="pushPin" Tapped="Pushpin_Tapped">
            <bm:MapLayer.Position>
                <bm:Location Latitude="47.2243186" Longitude="8.8186672" />
            </bm:MapLayer.Position>
        </bm:Pushpin>
    </bm:Map.Children>
</bm:Map>
```

In the “Location” setting, the Latitude and Longitude can be set. There is no service to give the coordinates of an address, this is now just a constant value pointing to Rapperswil. Coresystems does have a solution for this, where they send the coordinates through the synchronization. So this is an item for further improvement.

5.2.11 Add/Edit Page

5.2.11.1 Overview

Within an Add/Edit page a user can edit or add new properties of a business partner. The same page is used for adding a new business partner with the "Add" button in the Business Partners page. Special in this view is that addresses and contacts can be added by tapping on the "New-tile". A small popup flies in, where a user can add the details. To see the details of an address or contact or to edit them, a user just needs to tap on it.



Add/Edit

General Properties ›

Name: Coop
Type: COMPANY
Office Phone:
Mobile Phone:
Other Phone: 0800
Fax:
E-mail: info@coop.com
Website: www.coop.ch

Specific Properties ›

Price List:
Sales Person:
Payment Type:
Shipping Type:
Group Name:
Notes:

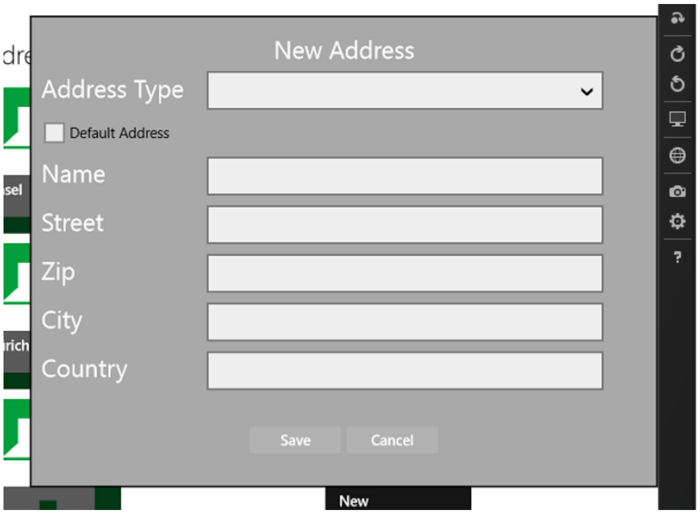
Addresses ›

Zürich
New
Basel
Zürich

Contacts ›

Hans Müller
Ben Chang
chang@coop.ch
New

Figure 26: Business partners edit page



New Address

Address Type:
□ Default Address
Name:
Street:
Zip:
City:
Country:

Save Cancel

Figure 27: Add/Edit address control

5.2.11.2 Implementation

AddEditBusinessPartnerPage

Following xaml code example shows how a popup for the addresses with transition can be added to a page.

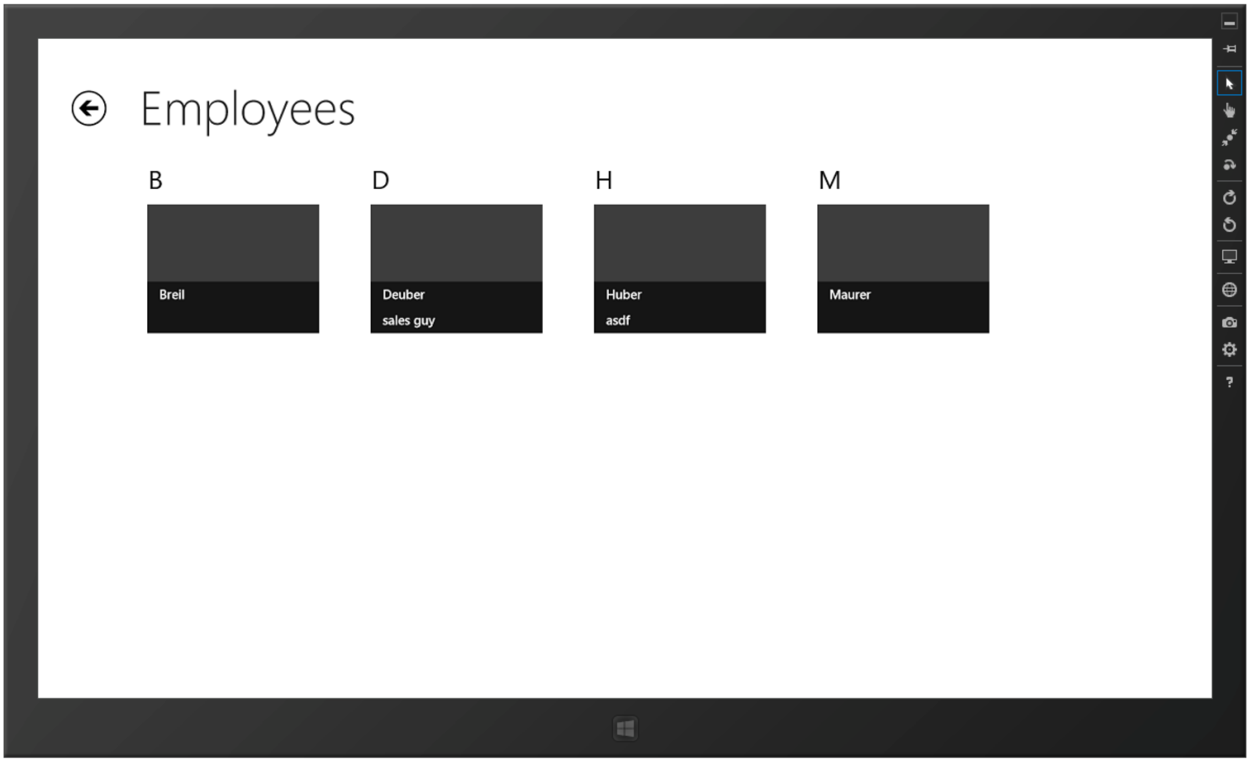
```
<Popup x:Name="addressControl" IsOpen="False" IsLightDismissEnabled="True">
  <Popup.ChildTransitions>
      <TransitionCollection>
          <PaneThemeTransition />
      </TransitionCollection>
  </Popup.ChildTransitions>
```

The same implementation is used for the “contacts-control”.

5.2.12 Employees Page

5.2.12.1 Overview

The “Employees Page” represents the employee module and contains all employees of a chosen company.



5.2.12.2 Implementation

This page has the same behavior like the BusinessPartnersModulePage, i.e. grouping, semantic zoom and loading the data asynchronously from the database.

The view model derives from the AbstractGroupedViewModel like BusinessPartnersModulePage which has been introduced above.

5.2.13 Find

5.2.13.1 Overview

The user can search for objects anytime, from anywhere in the application. This is done via the charms bar which has been described above.

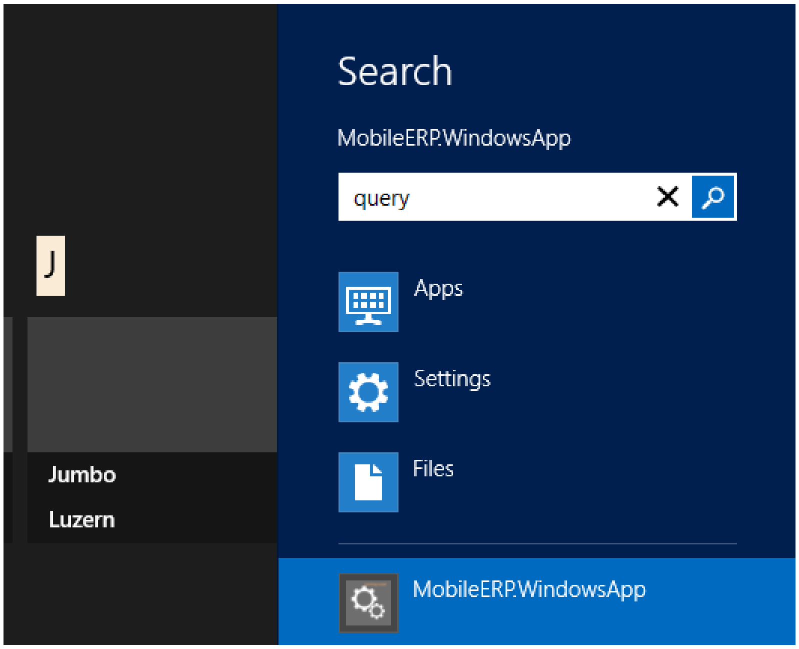


Figure 28: Search in the charms bar

This will then show a filtered view with results in all searchable objects.

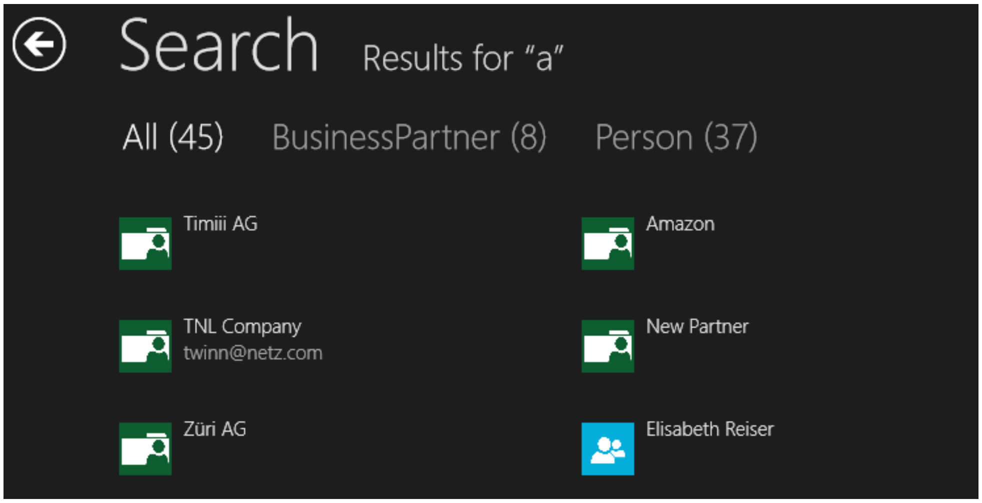


Figure 29: Filtered search results

5.2.13.2 *Implementation*

The search page is called from the charms bar and receives the search query as a navigation parameter. From there, the search service that is located in the business layer is called and an enumerable of BaseEntity is expected. The search results are saved in a dictionary, with the object type as key.

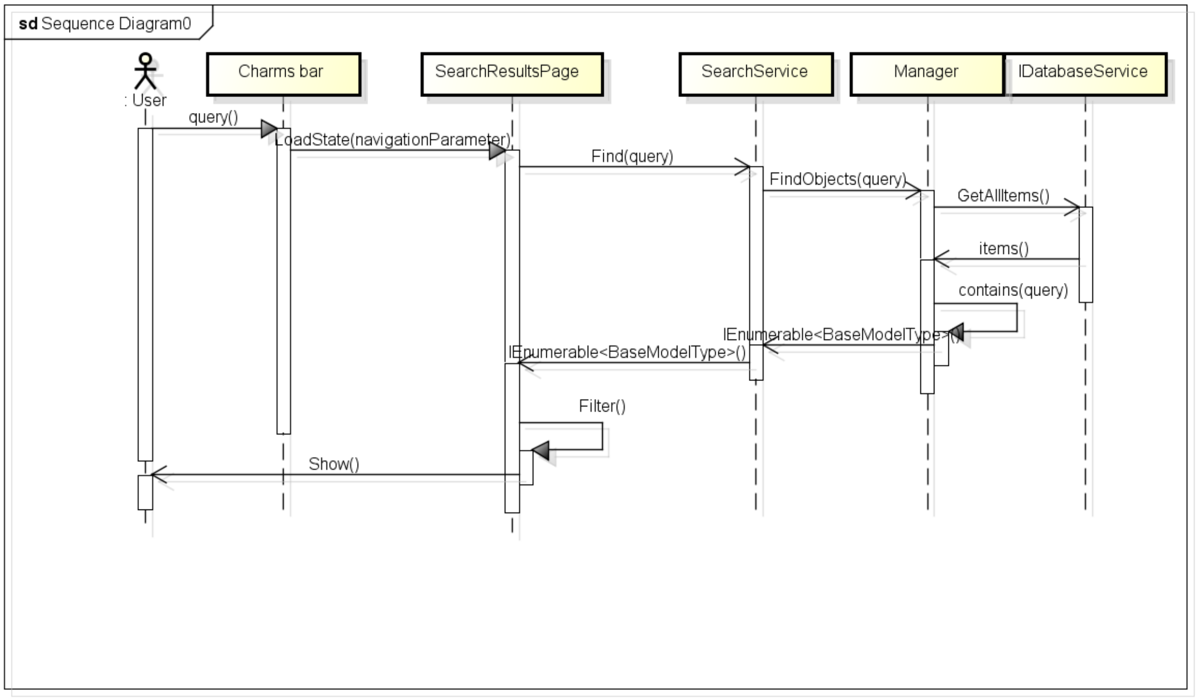


Figure 30: Search sequence diagram

The type is also the filter in the search view, as seen in figure 28.

5.3 SYNCHRONIZATION

Customers of coresystems can upload their business data to the coresystems cloud.
The customer's employees can access and work on this data through mobile devices or a browser. The current situation is that all frontend clients, which are available for different platforms, have to be synchronized manually at some point. The MobileERP solution supports an automatic synchronization of modified data in the background.

This synchronization process is a core feature of the MobileERP application. The goal was to develop an application which works online and offline. If the device is offline, all changes are stored locally in a database and as soon as the connection is available once again, the changes will be uploaded with a resulting synchronized device with the cloud.

5.3.1 Topology

The MobileERP application is connected to the cloud and pulls periodically for changes to it. Local changes on objects are upload directly after saving them. The main advantage of this asynchronous data exchange is that changes on objects are distributed very fast on all devices with the help of the automatic synchronization process that is implemented in the MobileERP application. In this environment conflicts can occur if different users are editing the same object. In that case the merge process will solve these conflicts which is explained below.

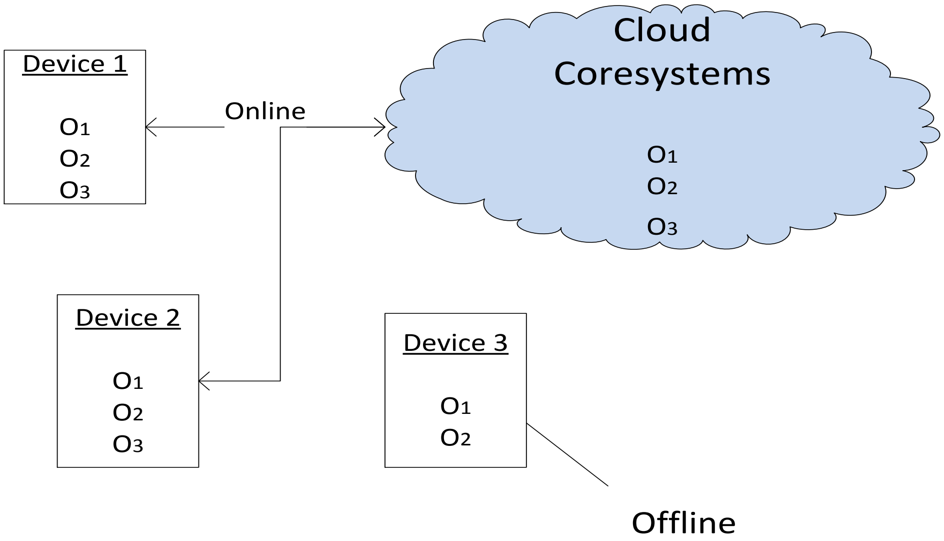


Figure 31: Online/Offline object state

Figure 31 Shows that all devices (Device 1 & Device 2) which are online have all object stored locally. Device 3 that is offline only has two objects stored locally, however the user of this device can work normally with the data stored on device 3. If the user connects the device to the internet the applications downloads automatically the object O_3.

5.3.2 Communication

Coresystems offers a JSON-API to exchange data between a device and the cloud. This API is used for the MobileERP solution. The transport protocol that is used for exchanging the JSON formatted data is called Hypertext Transfer Protocol (HTTP). A detailed description about the JSON format used to exchange information can be found in section 8.2.

5.3.3 Entities

Today's ERP solutions manage a huge amount of data for the users. All this different kinds of information objects are supported by coresystems, however this project only supports a subset of data to demonstrate the possibilities for Windows 8 Store Apps.

The MobileERP applications uses its own domain model, therefore the objects for the model are different than the DTO offered by coresystems. The solution is that an object is being converted between the domain model object and the DTO version of it. All domain model entities are located in the software solution in the namespace MobileERP.Business.Model and the DTO versions of each domain model object can be found in the namespace MobileERP.Data.DTO.

The details and data structure of the data transfer objects supported by coresystems are explained in the section 8.2.

5.3.3.1 Granularity

The granularity of change detection is the object itself. If in the MobileERP application a property of a domain object has been changed, the whole object is marked as modified. The boundary of the granularity has been given by the coresystems API which means that it is not possible to update only one field of an object and that is why the granularity of the given API has been adapted.

5.3.4 CRUD Operation

At the moment, the coresystems API only supports create, read and update of an object but no deletion.

5.3.5 Merge Concept

If an object is modified at the same time from two different devices a conflict between the two versions of the objects occurs. The cloud will accept the first change to the object but all other version will be discarded. The server detects a conflict by looking at the timestamp of each object. If an object with an outdated timestamp is send to the cloud the server will react with an error response.

If the server discards the object changes it will send an error response back to the device, which contains the actual version of the object in the cloud. Now, the MobileERP software has to merge the object locally and retry to upload it afterwards.

The MobileERP application merges an object by overwriting the local state of the object with the version of the cloud. All references to the object and from the object to other objects will be preserved.

5.3.6 Local DB

Coresystems does not offer any possibilities to find out which object are unused for a longer time. Therefore the MobileERP applications follows the existing solution of coresystems by storing a copy of all objects in the cloud locally.

5.3.7 Synchronization Flow

The following two sections describe the process of an update of an object O_1 that has been changed.

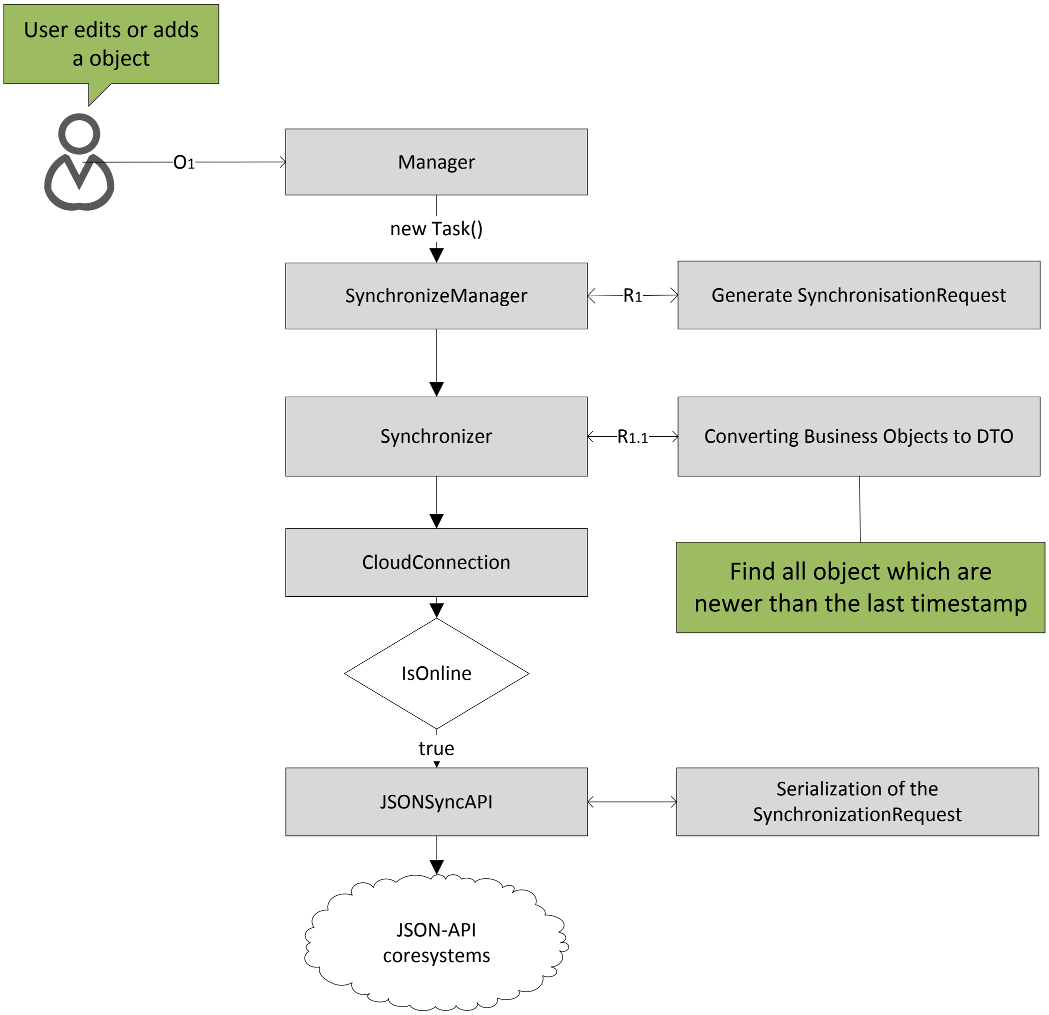

User edits or adds a object
O1
Manager
new Task()
SynchronizeManager
R1
Generate SynchronisationRequest
Synchronizer
R1.1
Converting Business Objects to DTO
Find all object which are newer than the last timestamp
CloudConnection
IsOnline
true
JSONSyncAPI
Serialization of the SynchronizationRequest
JSON-API coresystems

Figure 32: Synchronization flow

5.3.7.1 Sending process

1. Object O1 has been saved by the user
2. Then object O1 which has been modified is stored in a list called _ChangedObjects
3. The object O1 itself is then stored in the local database.
4. Now, the Manager initiates the synchronization process
5. An AddUpdateTask T_1 is generated which contains the object O_1
6. Then the SynchronizeManager is called to run the Task T_1 in a new Thread.
7. The Task T_1 generates a SynchronisationRequest R_1 and calls the Synchronzier
8. The Synchronizer converts all objects in a SynchronisationRequest R_1 to corresponding data transfer objects (DTO). The SynchronizationRequestConverter takes care of this step. A detailed description about it can be found in section SynchronizationRequestConverter
9. The Returned SynchronsiationRequest $_{R1.1}$ is sent to the CloudConnection component.

10. The CloudConnection checks first, if an internet connection is available. If so, the request is forwarded to the ICloudAPI interface.
11. The SyncAPIWithJson, which implements the ICloudAPI interface, now serializes the request to JSON and send the JSON string to the cloud.

5.3.8 Receiving process

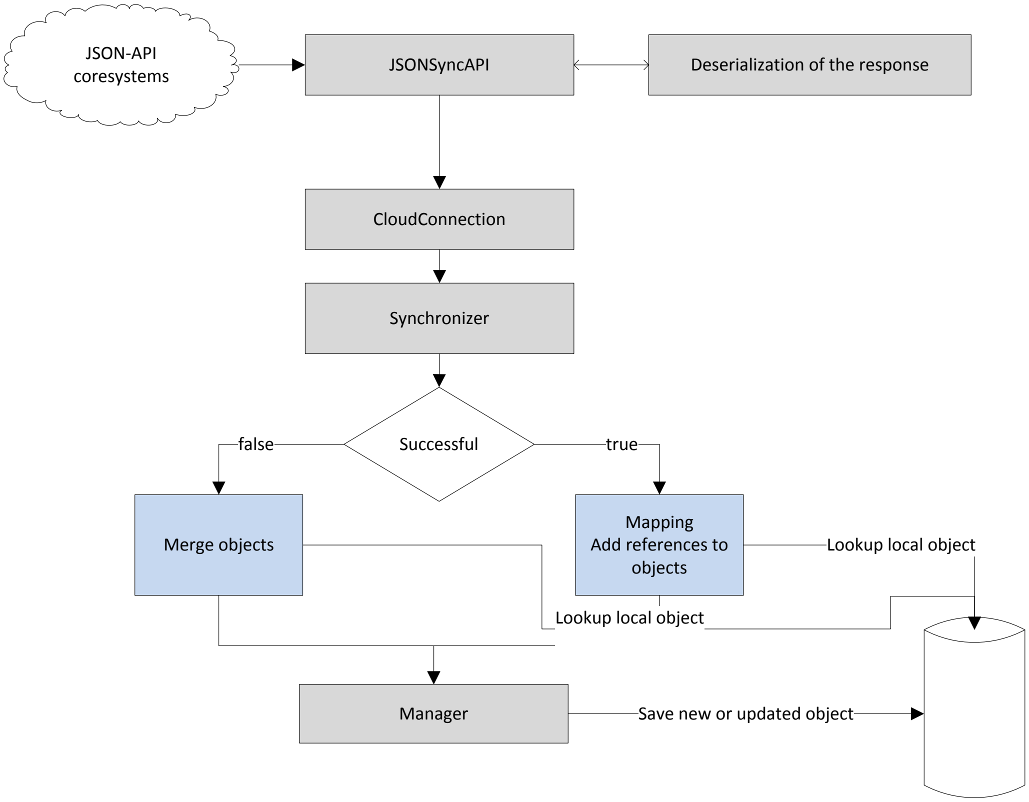


Figure 33: Response received flow

1. The HttpResponseMessage is deserialized
2. SynchronisationResponseSuccessful object is generated when everything was successfully updated
3. SynchronisationResponseWithError object is generated when an error occured
4. The response is returned to the CloudConnection and then to the Synchronizer.
5. The Synchronizer now checks the state of the response, if it was successful the objects in the response are mapped to the required objects. Otherwise the objects which are in a conflict state are overwritten by the server version of the object.
6. The Task T_1 calls now the Manger
7. The Manager now updates the database with the changes object in the response. Before all new or changed objects are saved, the manager compares the _ChangedObjects list with each object and removes the object if it was in the list.

5.3.9 Synchronizer

The Synchronizer class separates the manager from the communications part of the application. It provides one public method called synchronize() to start the synchronization process based on the information in the SynchronisationRequest object. The response of the CloudConnection is validated. If the successful property is true, the ORMapper takes care about further steps, otherwise the merge process starts.

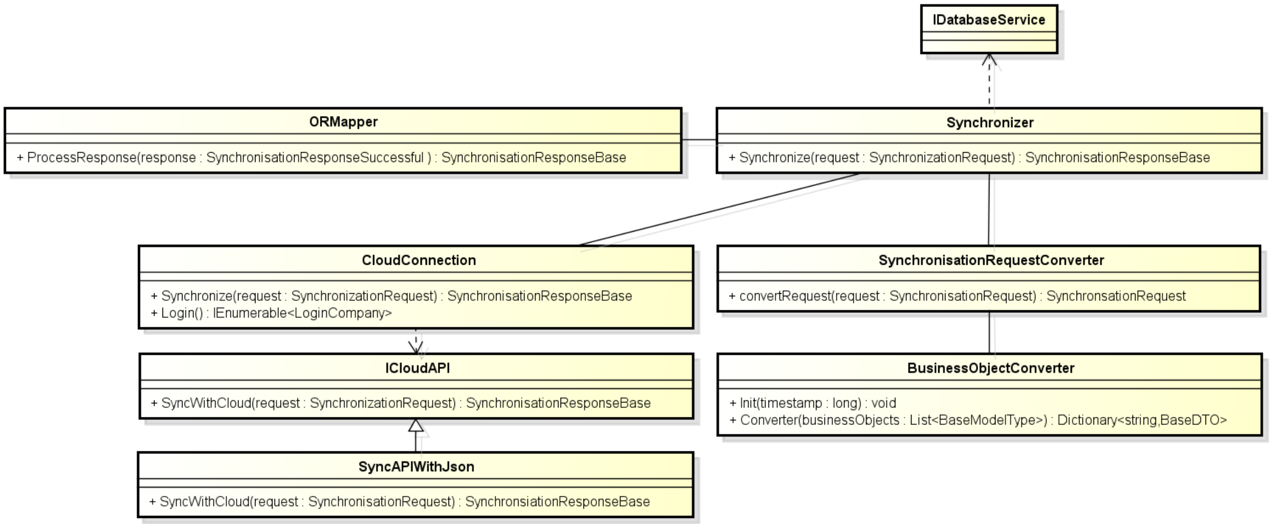
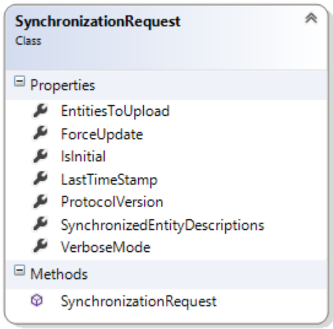

IDatabaseService
ORMapper
+ ProcessResponse(response : SynchronisationResponseSuccessful) : SynchronisationResponseBase
Synchronizer
+ Synchronize(request : SynchronizationRequest) : SynchronisationResponseBase
CloudConnection
+ Synchronize(request : SynchronizationRequest) : SynchronisationResponseBase
+ Login() : IEnumerable<LoginCompany>
SynchronisationRequestConverter
+ convertRequest(request : SynchronisationRequest) : SynchronsationRequest
ICloudAPI
+ SyncWithCloud(request : SynchronizationRequest) : SynchronisationResponseBase
BusinessObjectConverter
+ Init(timestamp : long) : void
+ Converter(businessObjects : List<BaseModelType>) : Dictionary<string,BaseDTO>
SyncAPIWithJson
+ SyncWithCloud(request : SynchronisationRequest) : SynchronsiationResponseBase

Figure 34: Synchronization classes

5.3.10 SynchronisationRequest

The SynchronizationRequest is the core object for any synchronization tasks. It contains all object which have to be uploaded to the cloud. It has some properties which are required by the API of coresystems. The EntitiesToUpload property is a dictionary with a type as key and List of objects of this type as value. This dictionary contains all objects which have to be uploaded to the cloud. The other special property is the SynchronizedEntityDescriptions property that contains a list of SynchronizedEntityDescription.



SynchronizationRequest
Class

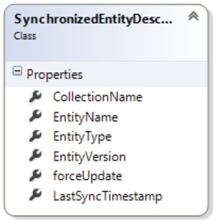
⊟ Properties
- EntitiesToUpload
- ForceUpdate
- IsInitial
- LastTimeStamp
- ProtocolVersion
- SynchronizedEntityDescriptions
- VerboseMode

⊟ Methods
- SynchronizationRequest

Figure 35: SynchronizationRequest class

5.3.11 SynchronizedEntityDescription

Each object that has to be uploaded, needs a description to build the JSON request. This DTO specific information is stored in the SynchronizedEntityDescription.



SynchronizedEntityDesc...
Class

Properties
- CollectionName
- EntityName
- EntityType
- EntityVersion
- forceUpdate
- LastSyncTimestamp

Figure 36: SynchronizationEntityDescription

5.3.12 SynchronisationRequestConverter

In general, this component translates the SynchronisationRequest with the business objects to the data transfer object representation of the business object. Furthermore, this class uses the BusinessObjectConverter to collect all objects which have to be synchronized with the cloud.

5.3.13 BusinessObjectConverter

The BusinessObjectConverter uses reflection to analyze each object. A recursive procedure is used to travel down the reference tree of each object. If an object is found, which LastChangedInApp value is higher than the last successful timestamp and is not already in the list of found objects, this object will also be added to this list

An important note is that the LastChangedInApp timestamp is an internal timestamp and not the timestamp from the cloud that is saved in each object. This was necessary because when an object is serialized, the timestamp form the server has to be preserved. To discover objects which have been changed since the last synchronization, the LastChangedInApp is used.

In addition, this class converts the domain model object to the data transfer object version of each object.

5.3.14 CloudConnection

This component checks the connectivity status of the device and sends the request to the ICloudAPI if an internet connection is available.

5.3.15 SyncAPIWithJson

This class implements the ICloudAPI interface and is the direct connection to the cloud of coresystems. It uses the JsonSerializer class to convert the request to the correct representation of the JSON request. The result will be sent to the cloud. The cloud API is based on the HTTP protocol and therefore the HTTPClient component which is provided by the .NET framework is used to the send the request. The response of the request is then deserialized with the help of the JsonDeserializer class. The JsonDeserializer handles the response and converts the string representation of the content to a SynchronisationResponseBase object. Depending on the successful value, it creates a SynchronisationResponseSuccessful or a SynchronisationResponseWithError object.

The JsonSerializer uses the Json.Net [5] library to generate a string representation of objects. This library is also used to deserialize the string value of the response into a SynchronizationResponseBase object.

The reason why choosing Json.Net can be found in the section 8.1.6.

5.3.16 Object Mapper

The DTOs often stay in a parent-child relation. Thus, only objects which can have a parent object, contain reference information to its parent. On the other hand, in our own business objects implementation, the parent object contains the information about referenced child objects. The ORMapper builds business objects according to the provided references.

A reference to an object is represented in the following format:

"OBJECT":{"OBJECTID" : E509F8ED5C70430D977747C816959F71","OBJECTTYPE" : “BUSINESSPARTNER"}

If an object contains reference information, the ORMapper searches for the required object in the response itself and if it cannot be found there, it makes a lookup in the database to find the domain object with the objectId and the object type.

Then, the ORMapper calls AddReferencedObject() on the object with the current object as parameter. The domain object, which has been found in the list or loaded from the database, is then added to a SynchronisationResponseSuccessful object. The ProcessResponse() method of the ORMapper returns the new created SynchronisationResponseSuccessful object to the Sychronizer.

5.3.17 Synchronization Response

There are three types of responses:

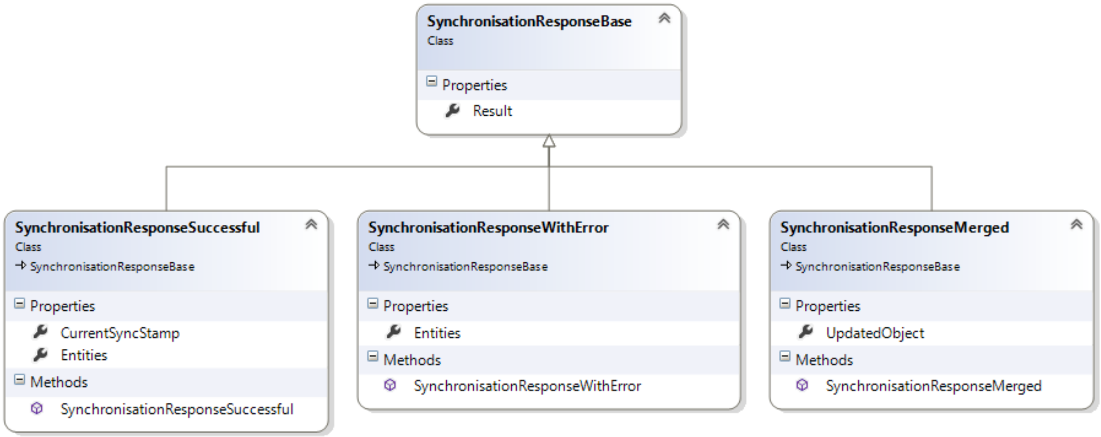

SynchronisationResponseBase
Class
Properties
Result
SynchronisationResponseSuccessful
Class
→ SynchronisationResponseBase
Properties
CurrentSyncStamp
Entities
Methods
SynchronisationResponseSuccessful
SynchronisationResponseWithError
Class
→ SynchronisationResponseBase
Properties
Entities
Methods
SynchronisationResponseWithError
SynchronisationResponseMerged
Class
→ SynchronisationResponseBase
Properties
UpdatedObject
Methods
SynchronisationResponseMerged

Figure 37: Types of responses

5.3.17.1 SynchronizationResponseSuccessful

This object is returned in the normal process.

5.3.17.2 SynchronizationResponseWithError

If an error occurs a SynchronizationResponseWithError object is generated. The property Entities is a dictionary with Type as key and the value is representing the in a ResponseEntityWithError object.

5.3.17.3 SynchronizationResponeMerged

In case of an error a merge is done by the Synchronizer. A SynchronizationResponseMerged object is being returned to the Manager. This object indicates that the timestamp of the last successful synchronization is still valid. The manager then updates the database with the merged objects.

5.3.18 Queuing

Because of the async framework there is not much control over the created threads. That is why some problems with thread synchronization occurred. This lead to errors, like lists being modified while iterating over them or database entries are being overwritten. The new version of Siaqodb 3.5.0.1, that came out on the 17th of December, can since then handle async calls on the database thread-safe [4]. Another solution has been introduced to solve problems with the synchronization. That is why a blocking collection as a queue has been introduced, which gets processed by a separate thread.

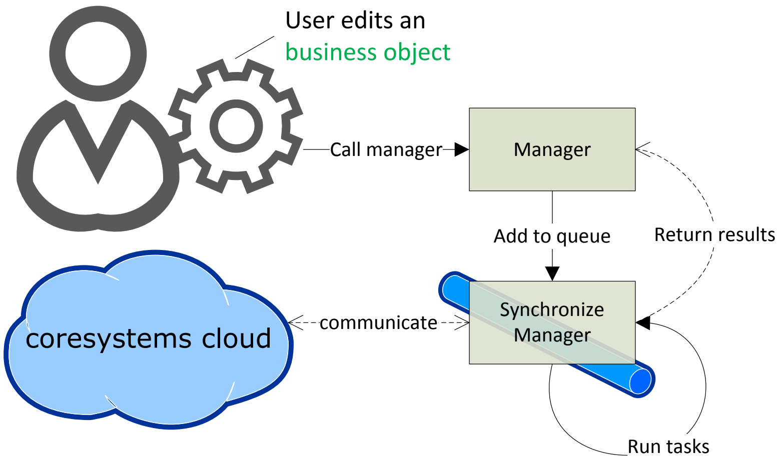


Figure 38: Communication through separate thread

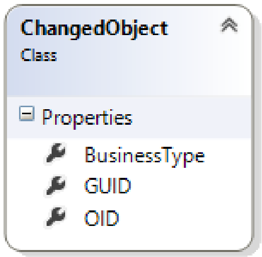
At the end of every task is an HTTP-request, which is waited for. So the flow of the task is synchronously and still does not block the UI-thread. The task will run in its own thread and works through the synchronization steps so the UI-thread is never blocking.

5.4 OFFLINE MODE

As seen in the requirements, the offline mode is a big part of our app. It ensures, that the app can be started and used with all the functions, even without an internet connection. The only constraint is, that the user had to successfully connect to the cloud previously. For best results, he should also have synced some data. Of course, he can add new entities without being synced with the cloud, but the risk of conflicts is a lot higher.

5.4.1 Concept

To reach this goal, a `ChangedObject` class has been introduced.



ChangedObject
Class

⊟ Properties

- BusinessType
- GUID
- OID

Figure 39: ChangedObject class overview

To track the changes, this class contains the IDs and the type of the changed entity.

The following picture should help to visualize the process of storing the changed data and synchronizing it with the cloud:

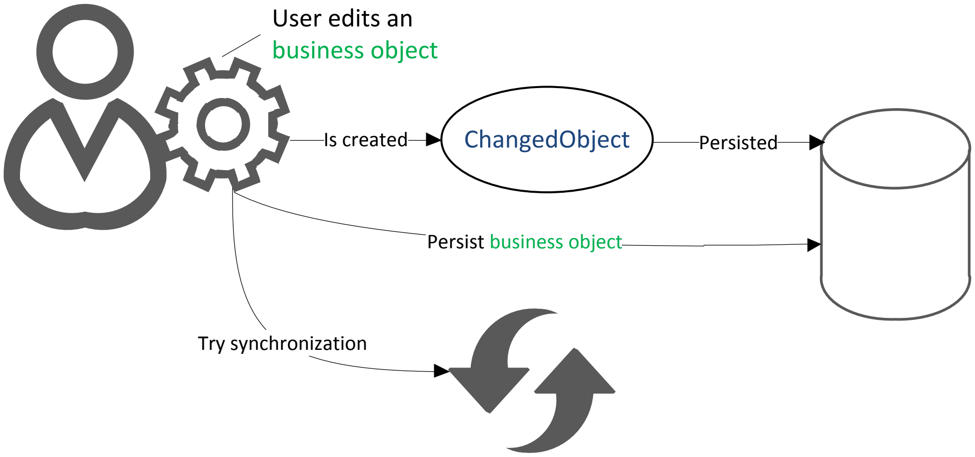


Figure 40: Synchronization process with offline persistence

This scenario runs with every change in a business object, so that the data is persisted, whether the synchronization fails or succeeds.

If the synchronization succeeds, the `ChangedObject` can be removed from the database:

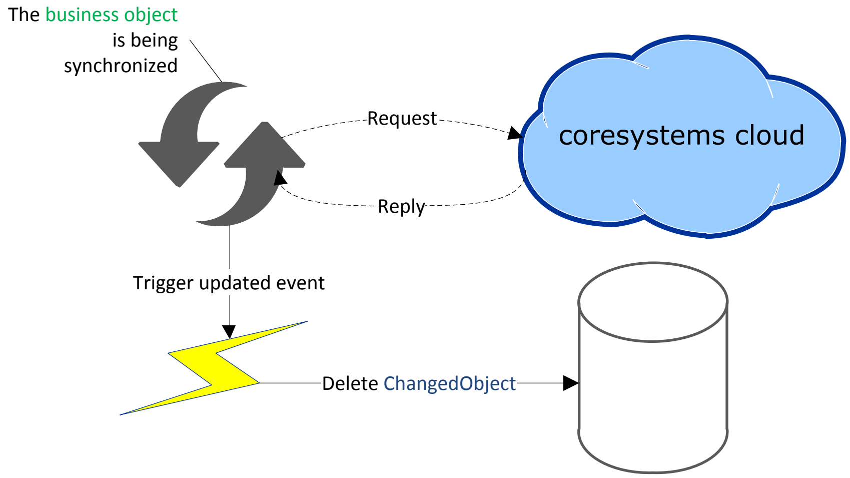


Figure 41: Synchronisation with the cloud

This means, the business object is definitely synchronized with the cloud and no more actions have to be done on the object.

If there is no internet connection, the synchronization will of course fail. That is why the system checks periodically, if there are objects which have to be synchronized. So when the internet is back, they can be synchronized in the background.

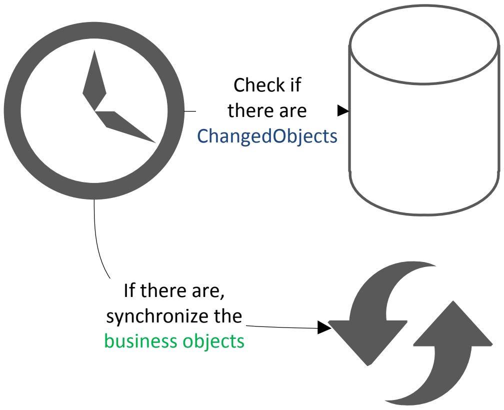

Check if there are ChangedObjects
If there are, synchronize the business objects

Figure 42: Synchronization after timer tick

The synchronization here is the same process as described in figure 41.

5.4.2 Implementation

As described above, the ChangedObject and the business object get persisted, as soon as the command is called from the UI thread. E.g.:

```
public async void AddObjectAsync<TObject>(TObject businessObject) where TObject :
BaseModelType, new()
```

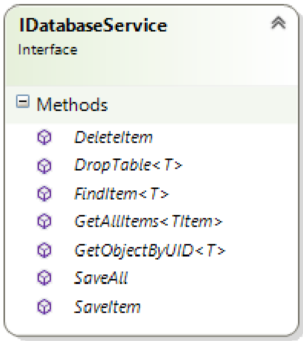
If the synchronization succeeds, ChangedObject is removed from the database again. So its life-span is rather short in a perfect scenario.

In the no-internet use case however, the synchronization depends on a DispatcherTimer in the Manager class. The interval of this timer is at one minute. The tick event calls the SynchronizeChangedObjectsAsync() method. It iterates through a collection of ChangedObjects and calls a synchronize for each change. Obsolete objects will be deleted, to maintain a clean state.

5.5 Persistence

As shown in the architecture, the project consists of two data models. For synchronization with the coresystems cloud their DTOs, as defined in the documentation by coresystems(see section 8.2), has been used. For internal use, the mobileERP applications uses the business objects, optimized for the business logic. This lead to the goal, not to create yet another model just for database access. As there weren't any object relational databases around when the project started, SQLite [5] has been used (see section 8.1.4). While it is fast and easy to use, it does not support complex types for persisting in Linq To SQL style. That is why after some reconsideration the database framework by siaqodb was chosen. Siaqodb is a NoSQL embedded object database engine [4] which allows to simply save, update and delete objects.

5.5.1 Implementation



IDatabaseService
Interface

Methods
- DeleteItem
- DropTable<T>
- FindItem<T>
- GetAllItems<TItem>
- GetObjectByUID<T>
- SaveAll
- SaveItem

Figure 43: Database interface overview

The IDatabaseService interface is similar to the IConnection interface. But since a NoSQL database service is used, there is no conversion to a database model. This way, the database service is fully generic, so that all domain objects can be persisted.

To achieve this, all domain objects must inherit the abstract BaseModelType class, which has two important fields for persistence:

OID

For persisting objects in the siaqodb framework, all objects need a unique identifier called OID.

```
public int OID { get; set; }
```

This value should never be manipulated and is for siaqodb internal use only.

GUID

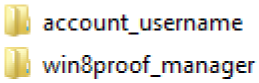
The GUID on the other hand, is the global unique identifier of every business object used for the synchronization to the cloud. It is set by both, coresystems and MobileERP, depending on who created the object. To make sure the database is consistent, the GUID is set as a unique constraint in the business object.

```
[Index]
[UniqueConstraint]
public string GUID { get; set; }
```

Also, it is indexed to increase performance.

5.5.2 Data management

The database is saved in the AppData folder of the logged-in user. From there it is branch for each coresystems user who uses the app on the machine.



account_username

win8proof_manager

This is to allow multiple coresystems users to use the same Windows login. To delete the local data, there is a function in the app's settings:

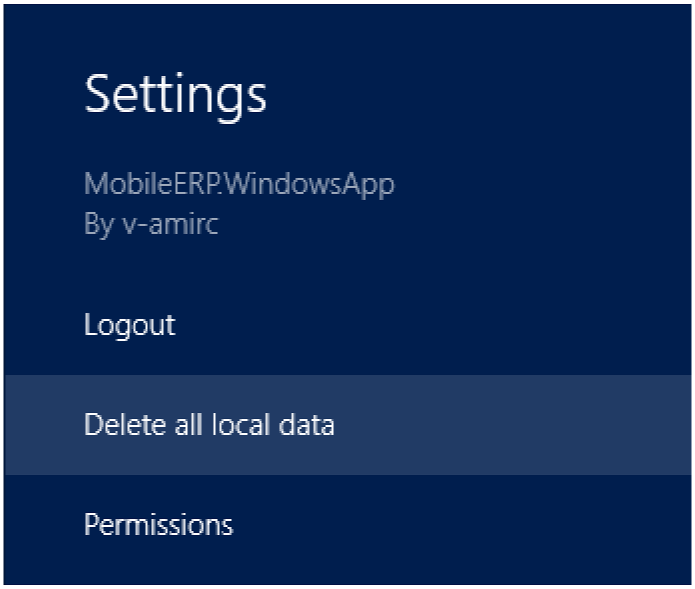


Figure 44: Settings in charms bar

5.6 OTHER

5.6.1 Settings

Not everything in the application is saved in a database. There is also a settings framework provided through the Windows 8 SDK. There are two places to save the settings: Local and roaming.

5.6.1.1 Local

Local settings are saved in the user's Windows profile. This settings are used to save data, which is system specific. There is the login data and the timestamp of the last global synchronization.

Saving local settings is quite simple. It needs just to be called

```
Windows.Storage.ApplicationData.Current.LocalSettings
```

to get the LocalSettings folder and save the settings similar to a dictionary:

```
composite["accountName"] = ApplicationSettings.AccountName;
```

The lifecycle of this is as long as the app is installed.

5.6.1.2 Roaming

For favorites of every type the roaming settings has been used. These settings are synchronized with Microsoft's cloud and are associated with the user's Live account. So if the user switches the device, or even if he has the app also installed on a Windows Phone 8, he will always have his favorites with him.

Since multiple users could use the same device for the application, the favorites are branched to the specific coresystems users. As a result, user A will not have the same favorites as user B, even if he is using the same device.

To add a favorite, some information has to be provided:

```
public static void AddFavorite<T>(string id, string branch)
```

- T: The type of the Favorite to save
- Id: The GUID of the business object
- Branch: The logged in coresystems user (account name & user name)

5.6.2 Exceptions

The exception handling is centralized in the ExceptionHandling class. If the OnException method is called, the error is written in the log and an OnError event is triggered.

The utility service subscribes to these events and bubbles them up to the UI. The active UI page dispatches the event to the UI thread and an error message is shown to the user.

5.6.3 Logging

Logging of exceptions and synchronization information is handled by an external class [6]. The log file is stored under C:\Users\<USERNAME>\AppData\Local\Packages\<Package family name>\LocalState\Exceptions.log and can be opened with a standard text-editor.

5.6.4 Authentication

Because we use the MVVM Pattern, even small field checks go through all layers of the application. The following diagram shows how it is determined whether the login screen is shown.

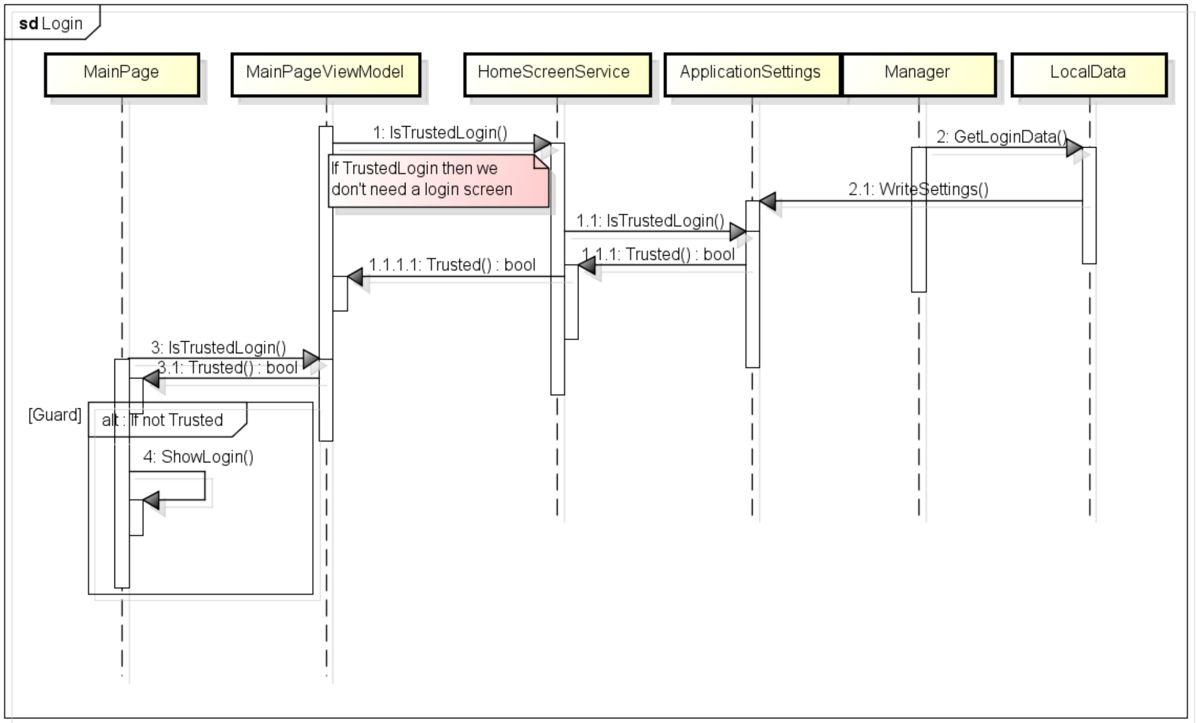


Figure 45: Authentication check

If the login is shown and the user logs in, the login request goes through all layers again to the communication interface. The request is actually an attempt to get the companies of the login, where a user can choose in which branch to log in.

6 TESTING

This section contains all result and describes all variants of tests, which are done during the project.

6.1 CONTINUOUS INTEGRATION

For this project there is no continuous integration system. This is due to some new technology used in it. Windows 8 Store apps requires a build server which is running on Windows 8 as an operation system. TFS-Preview has been used for source control. It is an online source control system with the advantage of an easy setup and a very easy user control. Unfortunately until this day it cannot build Windows 8 Store apps.

6.2 DATABASE PERFORMANCE TEST

The developers of the siaqodb already provide performance tests. Therefore no more testing in this area has been done.

Task	**Time [sec]**
Insert of 100‘000 objects	5.93
Read of 100‘000 objects	2.14
Update of 100‘000 objects	2.08
Delete of 100‘000 objects	1.06

The time values are provided by siaqodb and can be found under [http://siaqodb.com/?p=258]

6.3 Usability Testing

For the usability test a test person has been chosen that knows Windows and especially is familiar with the new gestures for Windows 8 touch devices. The use cases from chapter four have been tested.

6.3.1 Use case 01 – Login

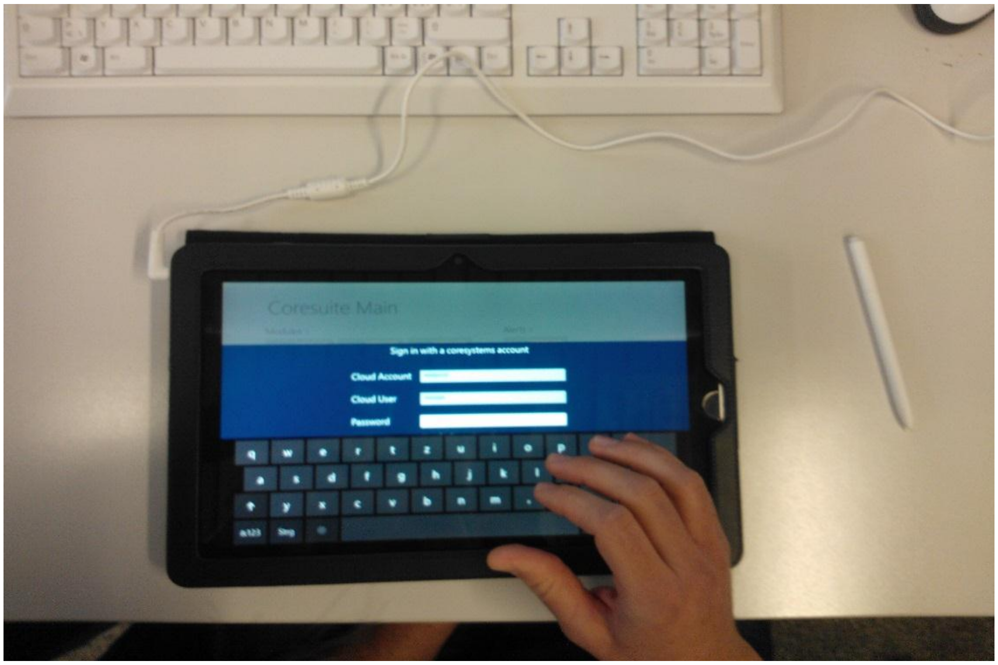


Figure 47: Login with virtual touch keyboard

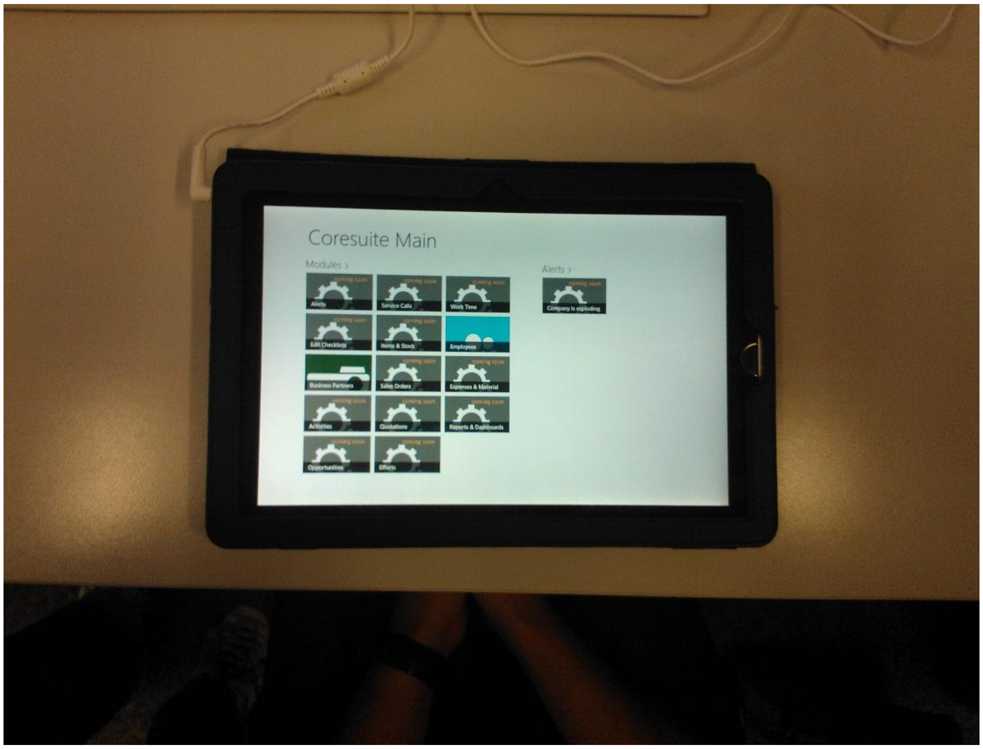


Figure 46: Main page after login

6.3.2 Use case 02 – Search for business partner

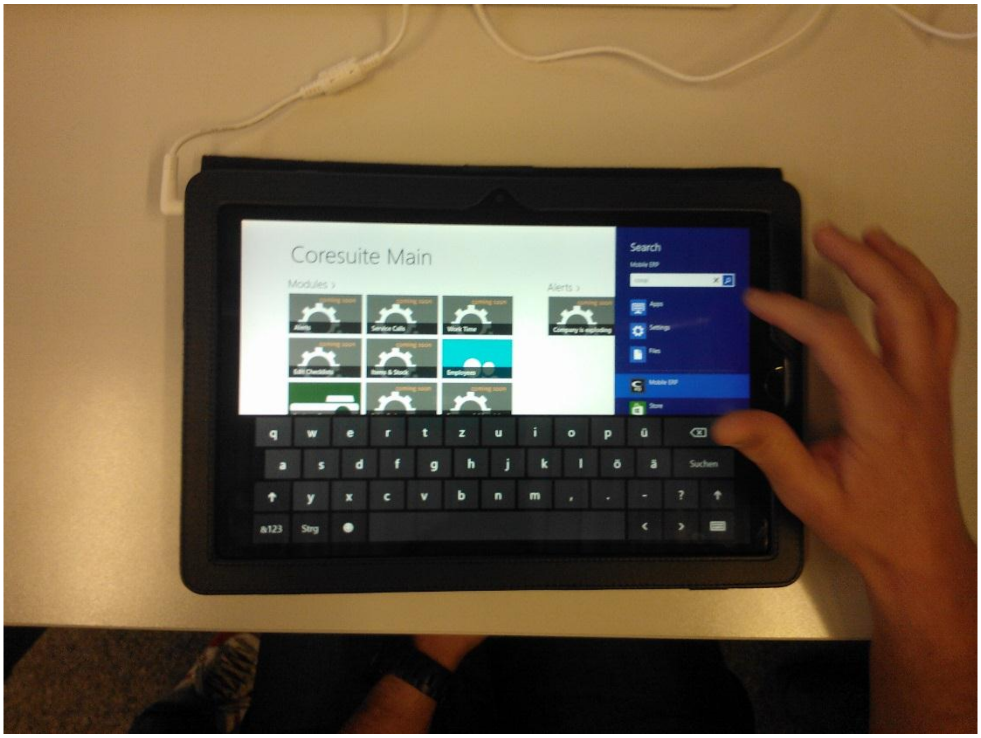


Figure 49: Search from Charm

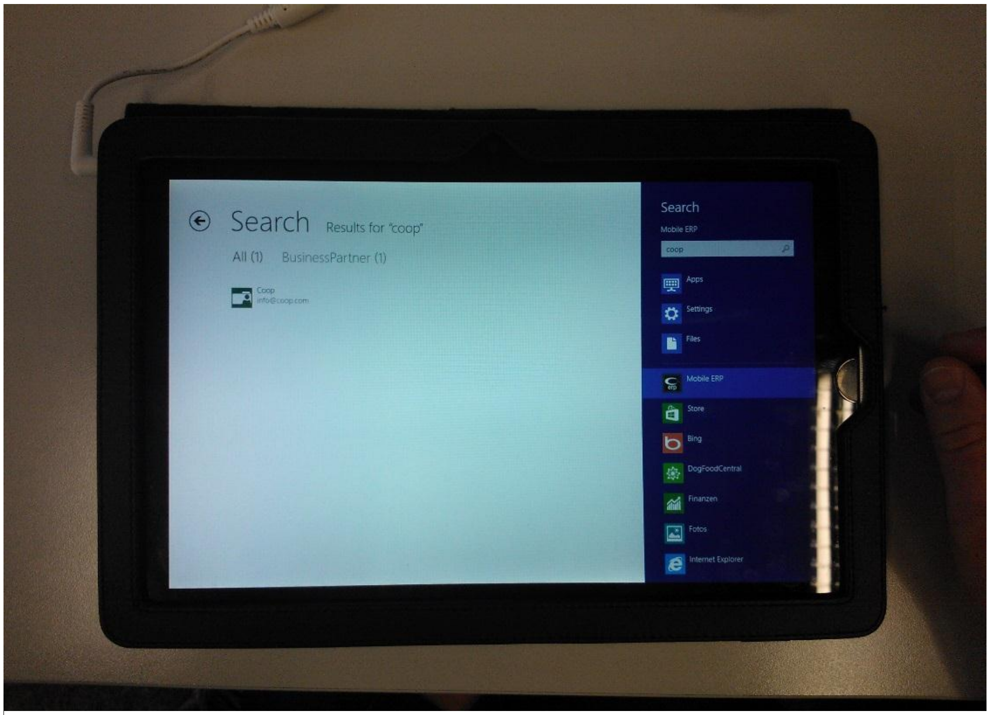
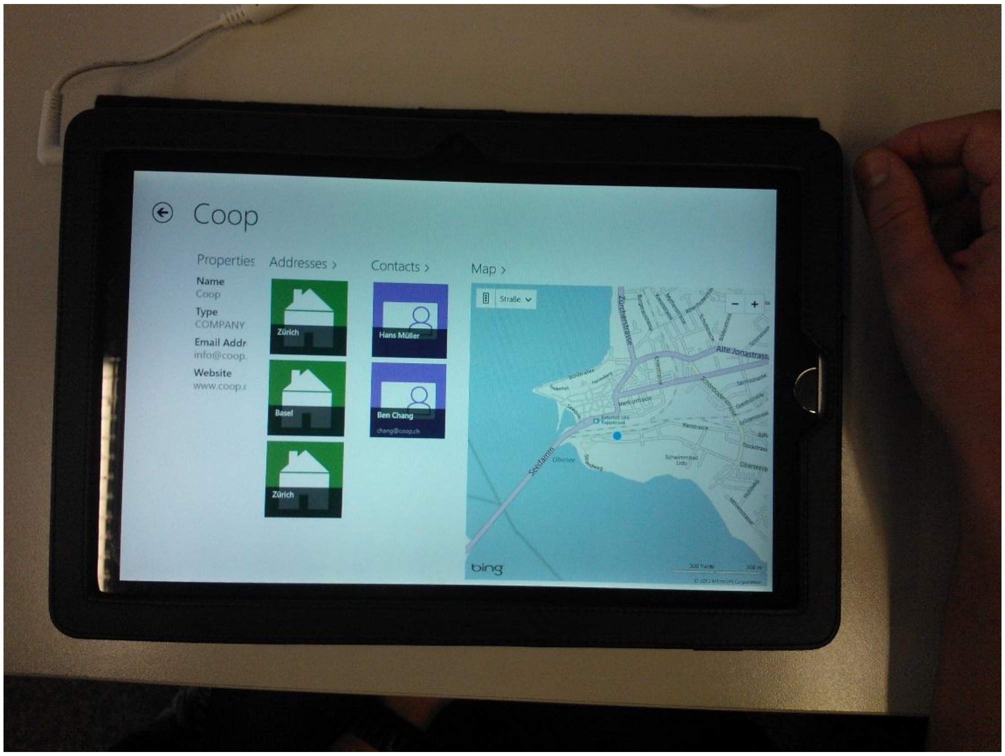


Figure 48: Business partner Coop has been found



Coop

Properties

Name
Coop

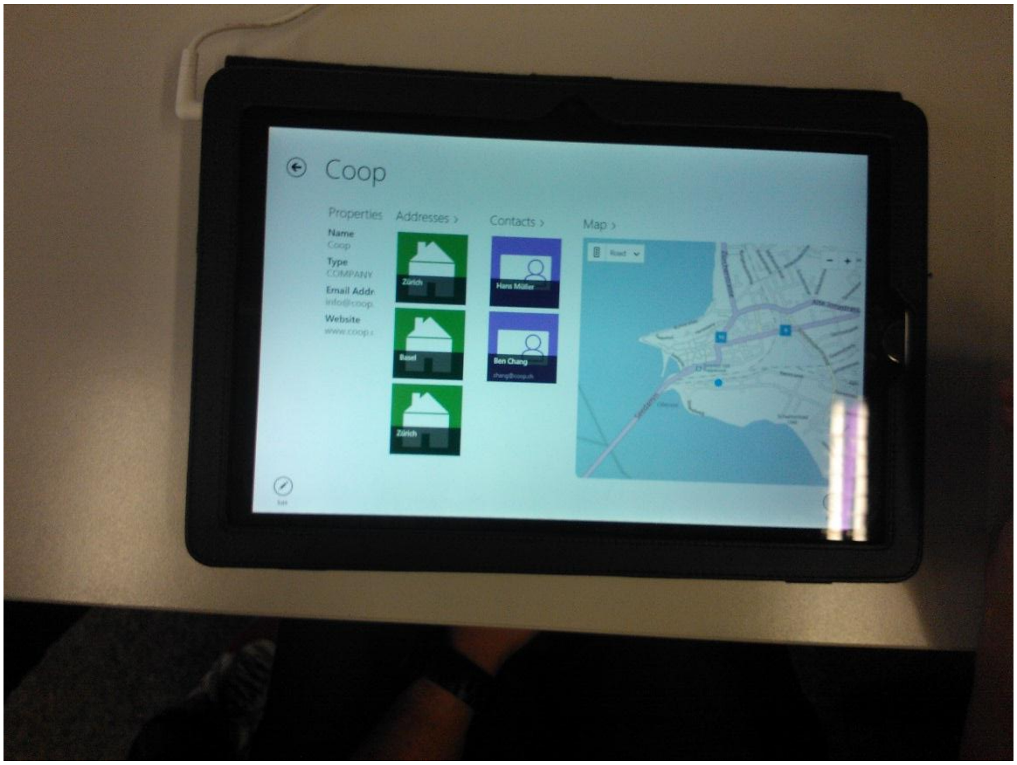
Type
COMPANY

Email Addr
info@coop.

Website
www.coop.(

Figure 51: Found business partner Coop has been opened from search result

6.3.3 Use case 03: Add new customer address



Coop

Properties

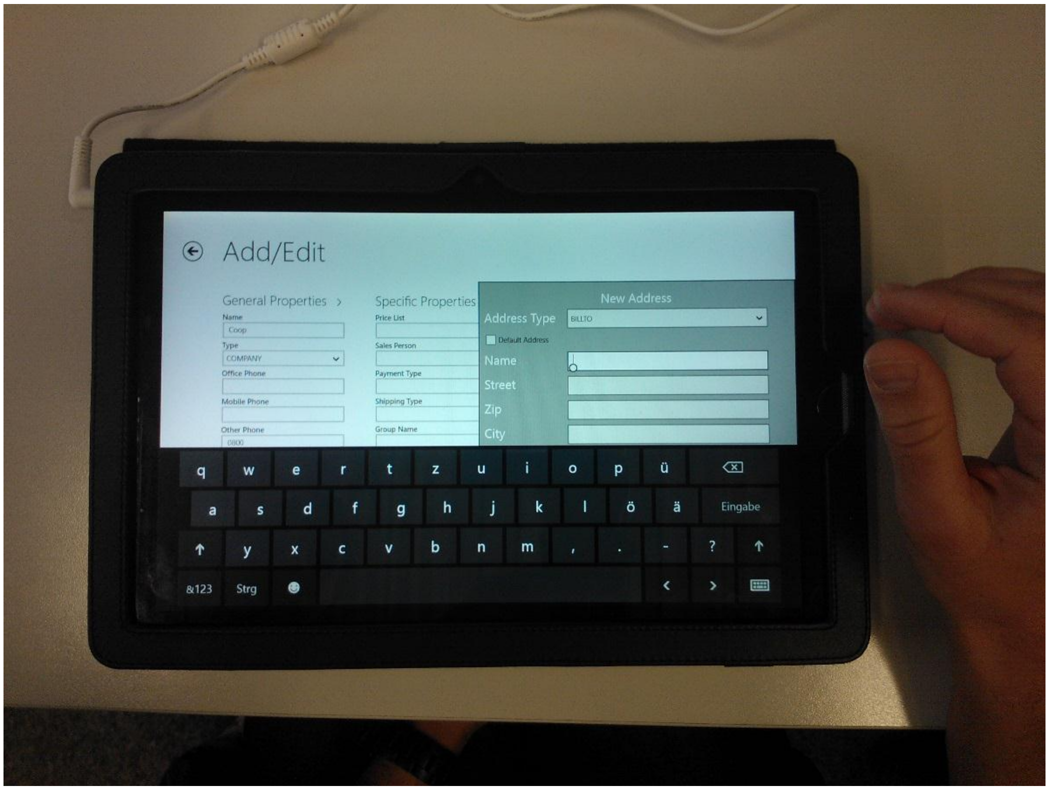
Name
Coop

Type
COMPANY

Email Addr
info@coop.

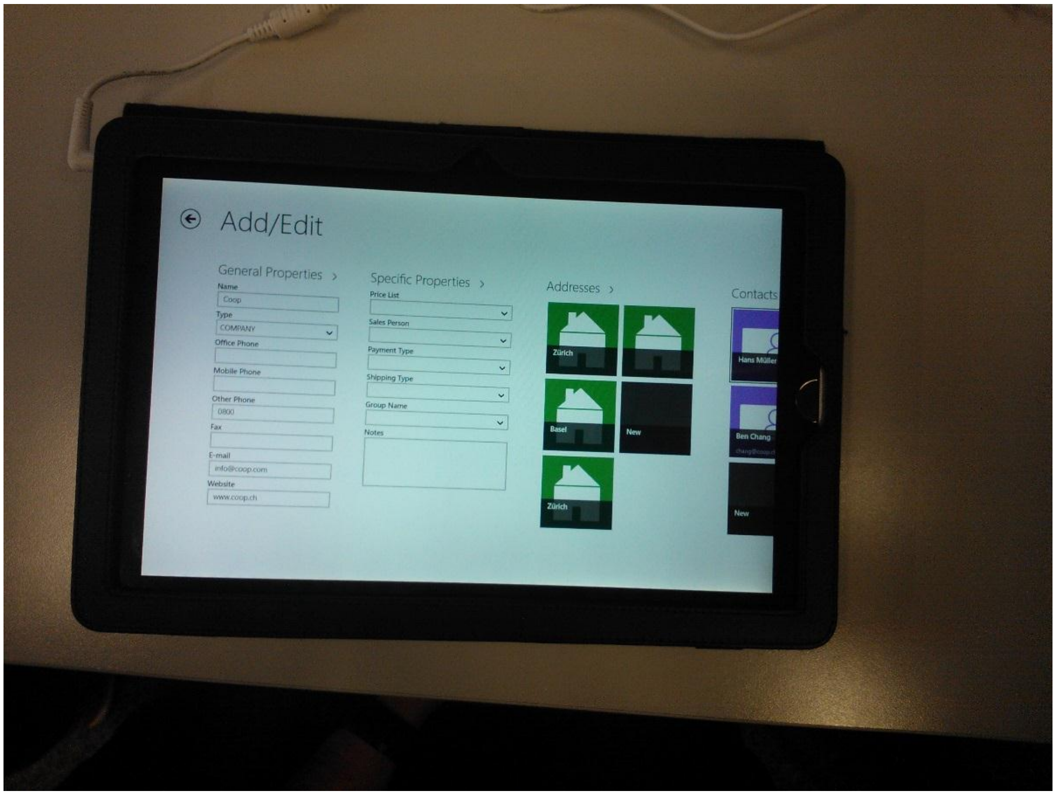
Website
www.coop.(

Figure 50: Opening the App Bar with swiping from below - Edit button appears



Name: Coop
Type: COMPANY
Office Phone:
Mobile Phone:
Other Phone: 0800
Price List:
Sales Person:
Payment Type:
Shipping Type:
Group Name:
Address Type: BILLTO
□ Default Address
Name:
Street:
Zip:
City:

Figure 52: Adding a new address for business partner coop



Name: Coop
Type: COMPANY
Office Phone:
Mobile Phone:
Other Phone: 0800
Fax:
E-mail: info@coop.com
Website: www.coop.ch
Price List:
Sales Person:
Payment Type:
Shipping Type:
Group Name:
Notes:

Figure 53: New address has been added and saved

6.3.4 Use case 05: Add new business partner

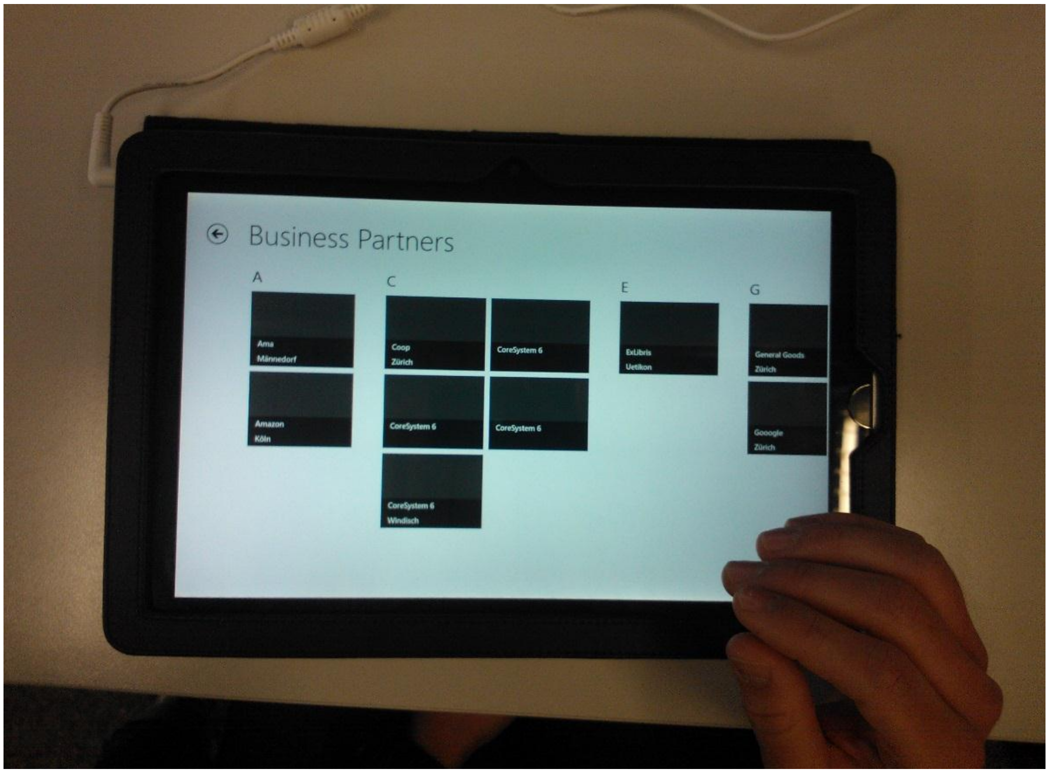


Figure 55: Business partner module page

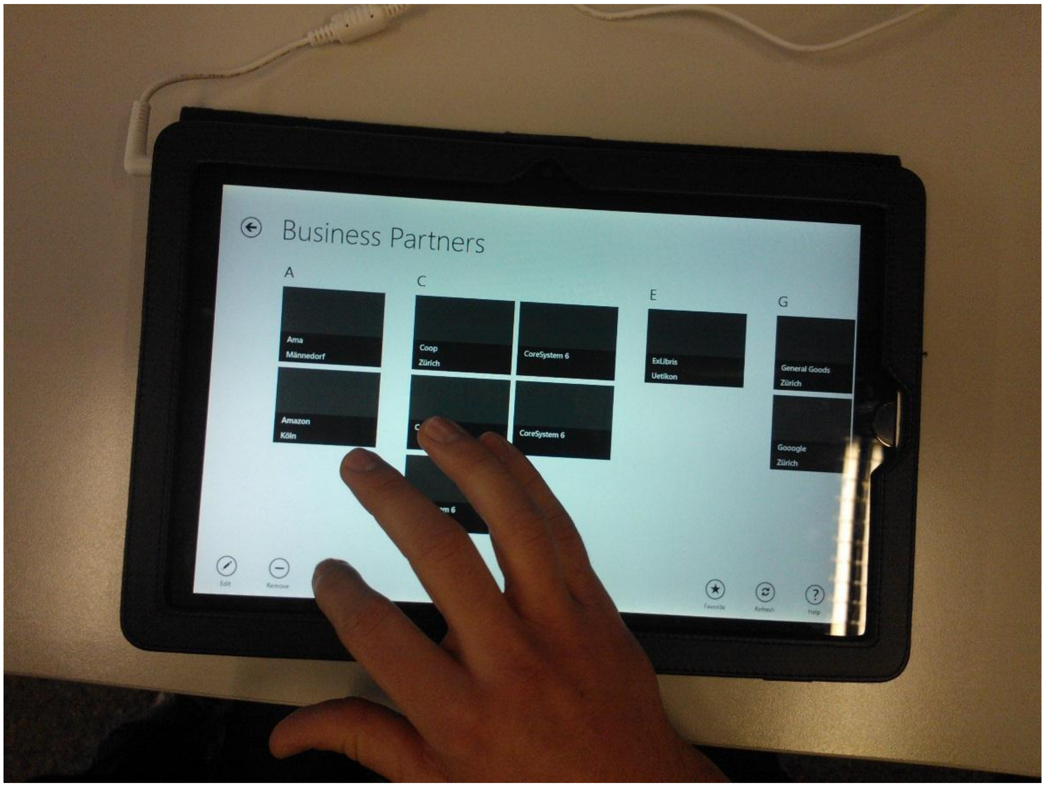


Figure 54: Opening the App Bar

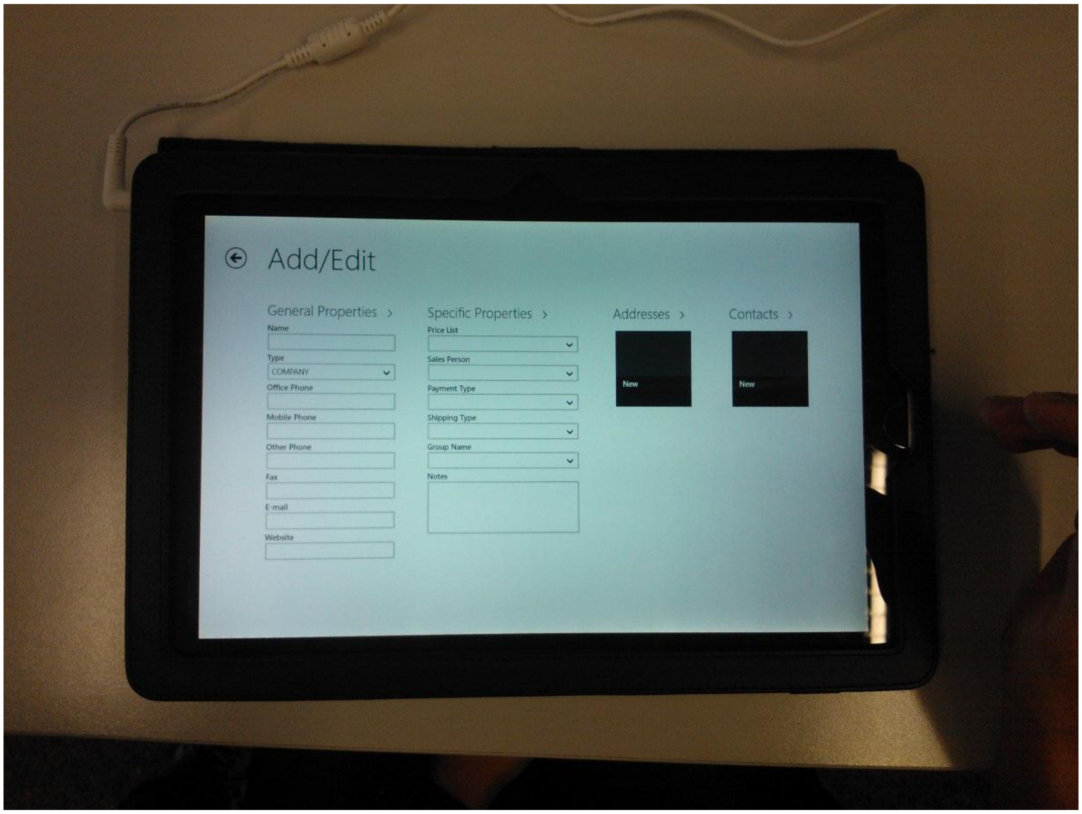


Figure 56: Adding properties, addresses and contacts for a new business partner

6.3.5 Results and conclusion

All uses cases has been done successfully in a good time and without any help.

6.4 System Testing

System tests are used to test the interaction between the application, network and the API from coresystems.

The following test cases are tested:

- Create a business partner
- Change a business partner
- Add an address or contact to an existing business Partner
- Change some business partner data through the Silverlight app and check if the changes are synchronized
- Test the offline modus
- Test the merge process

With the help of unit tests, many operations of the app are covered. However, since they can only run locally on a computer, synchronization operations must be performed and checked manually. For this, the system test have been made.

6.4.1 System-Test Week 07

Revision: 508

Precondition

- The app is installed on a Windows 8 device

Preparation

- App started
    - o Successful login
- Login into the Silverlight app

Process

The test person makes all steps to reach the goal of each test case.

Create business partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “New”. Fill out the form with all information you know. Save the new Business Partner and check online if it appears there too.	👎 partial success	In this version of the app, the goal was only to show the items. This aim was reached. Modifications of information fails

Change business partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “Edit”. Make your changes and save the new Business Partner. Go online and check if the changes are synchronized.	👎 partial success	Modification does not work

Add an address or contact to an existing business Partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “Edit”. Then click the contact or address button and fill out the form. Save the Business Partner	👎 partial success	Edit functionality is available but add fails.

Change some business partner data through the Silverlight app and check if the changes are synchronized		
Steps	Result	Comments
Log into the Silverlight App and make some changes there. Start the Mobile ERP app and check if all changes are appearing in the app.	👎 partial success	No updates visible because the upload does not work

Test the offline modus		
Steps	Result	Comments
Use the app and make some changes. Unplug or switch off the network connection of the device at any time. Now, make some more changes. Make sure the connection is available again and go the Silverlight App and check there if all changes are synchronized.	👎	Offline work not yet supported

Test the merge process		
Steps	Result	Comments
Take the internet connection of the device down. Make one or more changes at an object like address or business partner and memorize the objects you have changed. Then go to the Silverlight App and make at the same objects some modifications, make sure you synchronize the data with the cloud. Next, connect your device to the internet and check the object you have changed.	👎	Not yet implemented

6.4.2 System-Test Week 09

Revision: 902

Precondition

- The app is installed on a Windows 8 device

Preparation

- App started
  - Successful login
- Login into the Silverlight app

Process

The test person makes all steps to reach the goal of each test case.

Create business partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “New”. Fill out the form with all information you know. Save the new Business Partner and check online if it appears there too.	👍	

Change business partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “Edit”. Make your changes and save the new Business Partner. Go online and check if the changes are synchronized.	👍	

Add an address or contact to an existing business Partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “Edit”. Then click the contact or address button and fill out the form. Save the Business Partner	👍	Edit functionality is available but add fails.

Change some business partner data through the Silverlight app and check if the changes are synchronized		
Steps	Result	Comments
Log into the Silverlight App and make some changes there. Start the Mobile ERP app and check if all changes are appearing in the app.	👍	Takes some time, due to the timer

Test the offline modus		
Steps	Result	Comments
Use the app and make some changes. Unplug or switch off the network connection of the device at any time. Now, make some more changes. Make sure the connection is available again and go the Silverlight App and check there if all changes are synchronized.	👎	Partly changes are not upload after reconnecting to the could -> not stable

Test the merge process		
Steps	Result	Comments
Take the internet connection of the device down. Make one or more changes at an object like address or business partner and memorize the objects you have changed. Then go to the Silverlight App and make at the same objects some modifications, make sure you synchronize the data with the cloud. Next, connect your device to the internet and check the object you have changed.	👎	Works only for Business Partners (Known Issue)

6.4.3 System-Test Week 13

Revision: 1347

Precondition

- The app is installed on a Windows 8 device

Preparation

- App started
  - Successful login
- Login into the Silverlight app

Process

The test person makes all steps to reach the goal of each test case.

Create business partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “New”. Fill out the form with all information you know. Save the new Business Partner and check online if it appears there too.	👍	

Change business partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “Edit”. Make your changes and save the new Business Partner. Go online and check if the changes are synchronized.	👍	

Add an address or contact to an existing business Partner		
Steps	Result	Comments
Click on the Module called “Business Partner”, then swipe from the bottom and choose there “Edit”. Then click the contact or address button and fill out the form. Save the Business Partner	👍	Edit functionality is available but add fails.

Change some business partner data through the Silverlight app and check if the changes are synchronized		
Steps	Result	Comments
Log into the Silverlight App and make some changes there. Start the Mobile ERP app and check if all changes are appearing in the app.	👍	Takes some time, due to the timer

Test the offline modus		
Steps	Result	Comments
Use the app and make some changes. Unplug or switch off the network connection of the device at any time. Now, make some more changes. Make sure the connection is available again and go the Silverlight App and check there if all changes are synchronized.	👍	

Test merge process		
Steps	Result	Comments
Take the internet connection of the device down. Make one or more changes at an object like address or business partner and memorize the objects you have changed. Then go to the Silverlight App and make at the same objects some modifications, make sure you synchronize the data with the cloud. Next, connect your device to the internet and check the object you have changed.	👍	

6.5 Unit Testing

All Unit tests are created in the same package structure as the source code itself.

6.5.1 Unit Test Coverage

Visual Studio 2012 does not support code coverage feature for Windows Store Apps. Therefore, no code coverage value could be calculated [7].

6.5.2 Unit Testing Week 09

Revision: 902

Precondition

- Windows 8 installed with the app and the database is empty

Preparation

- Visual Studio is started with the latest checkout

Process

Start all Unit Test via Visual Studio 2012

Unit Test Package	**Result**
MobileERP.Test.Data.Communication	OK
MobileERP.Test.Data.Converter	OK
MobileERP.Test.Data.Persistance	OK
MobileERP.Test.Data.JSON	OK

6.5.3 Unit Testing Week 13

Revision: 1347

Precondition

- Windows 8 installed with the app and the database is empty

Preparation

- Visual Studio is started with the latest checkout

Process

Start all Unit Test via Visual Studio 2012

Unit Test Package	**Result**
MobileERP.Test.Data	OK
MobileERP.Test.Data.Communication	OK
MobileERP.Test.Data.Converter	OK
MobileERP.Test.Data.Persistance	OK
MobileERP.Test.Data.DTO	OK
MobileERP.Test.Data.ORMapper	OK

6.6 Network Testing

System testing is performed between multiple devices which are logged in to the same company. Thus the test network is also covered in the system test.

6.6.1 Network Testing Week 07

Revision: 508

Precondition

- App is freshly installed and the cloud is available

Preparation

- Start the app on a Windows 8 device

Process

Like in the stability test but with a loss of connection at any point of time.

Test	Result	Comment
Work with the app like a normal user and disconnect the device for the internet at some time.	👎	The offline mode is not yet implemented

6.6.2 Network Testing Week 09

Revision: 902

Precondition

- App is freshly installed and the cloud is available

Preparation

- Start the app on a Windows 8 device

Process

Like in the stability test but with a loss of connection at any point of time.

Test	Result	Comment
Work with the app like a normal user and disconnect the device for the internet at some time.	👎	Version is not stable

6.6.3 Network Testing Week 13

Revision: 1347

Precondition

- App is freshly installed and the cloud is available

Preparation

- Start the app on a Windows 8 device

Process

Like in the stability test but with a loss of connection at any point of time.

Test	Result	Comment
Work with the app like a normal user and disconnect the device for the internet at some time.	👍	Full functional

7 RESULTS AND CONCLUSION

All the required and important requirements have been implemented and tested. The MobileERP app supports offline and online mode and has a generic core for handling objects. The user interface follows closely the Microsoft's guidelines for Window 8 Store Apps. To guarantee a good user experience, the synchronization runs as a background task and leaves the UI responsive all the times.

Since the emphasis of the project was on a good architecture, not many modules of an ERP system are implemented in our app.

We did run in some problems however. A goal we clearly missed, was developing an UI for Windows Phone 8. Since the SDK for it only came out at the end of October, we could not create the initial project as a Portable Class Library (PCL) which includes Windows Phone 8. This lead to a development which aimed solely on Windows 8 tablets. Unfortunately there was no time anymore to port the project to PCL, which would be needed to use the business logic in a Windows Phone 8 project.

7.1 USE CASES

The following use cases have been implemented and tested:

- UC 01: Login
- UC 02: Search for Business Partner
- UC 03: Add New Customer Address
- UC 05: Add New Business Partner

This use case could not be implemented in the context of the project

- UC 04: Display Report

The optional use case UC 04 has been skipped due to a lack of time.

7.2 USER INTERFACE

- The remaining modules have to be implemented
- Hard to implement the UI in tile style

7.3 PERFORMANCE

We figured out that during the initial synchronize, e.g. when opening the BusinessPartnersPage, it takes some time until the first partners are shown. One reason is that the data first needs to be downloaded from the cloud and then the corresponding objects have to be created. One approach to avoid waiting time is to implement a synchronization process with eager loading the objects when starting the app. So once the objects are downloaded and stored in the local database the partners are added immediately after opening the BusinessPartnersPage. Another approach is to use gzip for data synchronization that is used for zipping files and exchanging them through the network.

At the beginning we didn't plan to do a bulk test.

7.4 Database

Siaqodb [4] has proved itself as a good object oriented database service. Until there are new alternatives we suggest to continue using it in this project.

7.5 Synchronization

The MobileERP application synchronizes at change on an object instantaneous with the cloud. At the moment all mobile solutions of coresystems have to be synchronized manually at some time.

7.5.1 Merge Concept

In this project we have realized only a simple merge process by using the server version of a object. However the software is designed to implement a more sophisticated algorithm to handle merge conflicts.

7.6 Online / Offline Support

MobileERP is working in an offline environment with a local database and starts synchronizing with the cloud in the background as soon as an internet connection is available.

7.7 Open Points

- Bing Maps address search on the BusinessPartnersDetailPage
- Validation of the properties like adding a correct email address
- Performance/Bulk test with a big amount of data
- Implementing the remaining modules
- Windows Phone 8 development

8 APPENDIX

8.1 PRESTUDIES

8.1.1 Windows 8 user experience guidelines

The guidelines can be found in the file Win8_UXG_RTM.pdf.

8.1.2 Portable Class Library

Our goal for the bachelor thesis was to develop a Windows Phone 8 and a Windows Store App. In our pre-studies we discovered “Portable Class Library” (PCL). The PCL project in Visual Studio 2012 supports the cross-platform development of .Net Framework apps. [8] Within this new project portable assemblies can be written and built that work without modification on multiple platforms. The whole business logic and communication layer can be developed and reused to target multiple platforms. It only needs then to create a new UI layer that uses this assembly. Nevertheless, with all the described advantages we had to drop this approach, because no references can be added given that we had to use a local database for our application. The consequence was that we had to start a new common class library project, with the risk that not all platforms will be supported.

8.1.2.1 Supported features

When specifying the platforms that wants to target in a Portable Class Library project, the supported assemblies for those platforms are automatically referenced in the project. [8] The referenced assemblies are automatically updated if the target platforms changes. [8]

8.1.2.2 Supported types and members

The types and members that are available in Portable Class Library projects are constrained by several compatibility factors:

- They must be shared across the target platforms that will be selected.
- They must behave similarly across those platforms.
- They must not be candidates for deprecation.
- They must make sense in a portable environment, especially when supporting members are not portable.

For example, the Portable Class Library project does not contain any UI components because of behavioral differences between the UIs of different devices. There may also encounter limitations if targeting platforms (such as Xbox, the .NET Framework 4, and Windows Phone 7) that were released before the introduction of the Portable Class Library. [8]

8.1.2.3 Supporting the Model-View View-Model (MVVM) pattern

When targeting the .NET Framework 4.5, .NET for Windows Store apps, Silverlight, and Windows Phone, the MVVM pattern can be implemented in the solution. [8] The classes to implement this pattern include the following:

- System.Collections.ObjectModel.ObservableCollection<T>
- System.Collections.ObjectModel.ReadOnlyObservableCollection<T>
- System.Collections.Specialized.INotifyCollectionChanged
- System.Collections.Specialized.NotifyCollectionChangedAction
- System.Collections.Specialized.NotifyCollectionChangedEventArgs
- System.Collections.Specialized.NotifyCollectionChangedEventHandler
- System.ComponentModel.DataErrorsChangedEventArgs
- System.ComponentModel.INotifyDataErrorInfo
- System.ComponentModel.INotifyPropertyChanged
- System.Windows.Input.ICommand

8.1.3 Localization

A windows user can specify their location, which can be anywhere in the world. Additionally the user can define that they speak any language in any location. (The location and language can be independently chosen.)

The User can run apps in a completely different language than Windows. In all Windows Store apps, a language is represented by a BCP-47 language tag.

8.1.3.1 Instruction

1. The user languages preferences list is an ordered list of languages that describe the user's languages in the order that they prefer them. This list can be set under the Language setting of Windows. This list can contain multiple languages and regionals.
2. The app developer specifies the list of supported languages in the Resource element of the apps manifest file.
3. If an app does not support any of the languages that the user has defined in the Language settings. The app uses the default language.
4. At runtime, the system determines the user language preferences that the app declares support for in the manifest file. Based on this information it creates an application language list.
5. If a user wants to use a specific language supported by the app, the ApplicationLanguages class provides the possibility to override the user's ranked list of preferred languages. The app can override the top language of the list. An example how to do this is shown in the next section.

8.1.3.2 Code Example

This code example shows how to override the language list by another language the user has chosen from the set of supported app languages.

```
// [9]aren't the default show them first
            if (ApplicationLanguages.PrimaryLanguageOverride != "" ||
ApplicationLanguages.Languages.Count > 1)
            {
                for (var i = 0; i < ApplicationLanguages.Languages.Count; i++)
                {
                    if ((ApplicationLanguages.PrimaryLanguageOverride == "" && i != 0) ||
(ApplicationLanguages.PrimaryLanguageOverride != "" && i != 1))
                    {
                        this.LanguageOverrideComboBox_AddLanguage(new
Windows.Globalization.Language(ApplicationLanguages.Languages[i]));
                    }
                }
                comboBoxValues.Add(new ComboBoxValue() { DisplayName = "——————", LanguageTag =
"" });
            }

            // Finally, add the rest of the languages the app supports
            List<Windows.Globalization.Language> manifestLanguageObjects = new
              List<Windows.Globalization.Language>();
            foreach (var lang in ApplicationLanguages.ManifestLanguages)
            {
                manifestLanguageObjects.Add(new Windows.Globalization.Language(lang));
            }
            IEnumerable<Windows.Globalization.Language> orderedManifestLanguageObjects =
              manifestLanguageObjects.OrderBy(lang => lang.DisplayName);
            foreach (var lang in orderedManifestLanguageObjects)
            {
                this.LanguageOverrideComboBox_AddLanguage(lang);
            }

            LanguageOverrideComboBox.ItemsSource = comboBoxValues;
            LanguageOverrideComboBox.SelectedIndex = comboBoxValues.FindIndex(FindCurrent);
            LanguageOverrideComboBox.SelectionChanged +=
              LanguageOverrideComboBox_SelectionChanged;
```

This is a code snippet from the code sample of Microsoft [10], which shows how to change the default language settings.

8.1.4 Persistence Study

One of the core-requirements, as defined by CoreSystems, is the persistence of the data in the app. Since it should be possible to access all the data even if there's no mobile internet available, a structured storage must be included.

Windows RT does provide local storage mechanisms, but only file-based. So we had to find an appropriate alternative, which excels through good performance. In their Windows Silverlight application CoreSystems uses the Perst database system. Unfortunately there wasn't a Windows RT version available when we started the thesis, so we took a closer look at SQLite [5].

8.1.4.1 SQLite

It is relatively simple to create a SQLite [5] application. First the library has to be compiled for the according CPU-type and then it can be integrated in an application.

Building the library

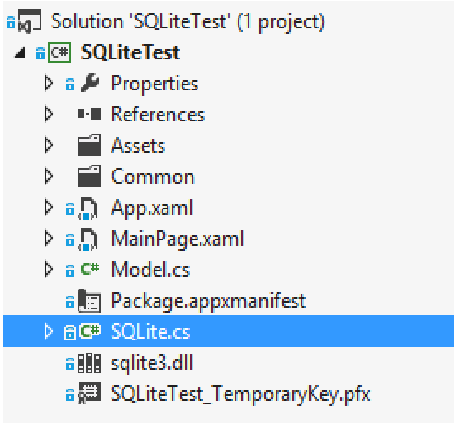
To build the library, we followed closely the steps described here http://sdrv.ms/Kz8XKV. These are the main steps:

1. Start a VS command prompt as admin. The chosen arch is the bitness the library will be created for (x64, x86, ARM).
2. Pull the latest SQLite [5] source code from the fossil repository
   a. *fossil clone http://www.sqlite.org/cgi/src sqlite3.fossil*
3. Open the repository
   a. *fossil open sqlite3.fossil*
4. Build the dll
   a. *nmake -f makefile.msc sqlite3.dll FOR_WINRT=1*

Now the library can be added to a project as content file, adding as a reference isn't possible yet. Additionally the Microsoft C++ Runtime Library must be referenced.

Using the library

To access SQLite's features, a client library is needed. We chose sqlite-net, since it's simple to include in the project and it follows an object-oriented approach, similar to Linq To SQL.



```
Solution 'SQLiteTest' (1 project)
  SQLiteTest
    Properties
    References
    Assets
    Common
    App.xaml
    MainPage.xaml
    Model.cs
    Package.appxmanifest
    SQLite.cs
    sqlite3.dll
    SQLiteTest_TemporaryKey.pfx
```

Figure 57: SQLite class in project

It contains the synchronous API SQLite.cs and the asynchronous API SQLiteAsync.cs. To use them, they simply have to be included in the project.

With these classes we can create tables, queries or manipulate data. We created a small example as a performance test:

Write in DB (synchronous)

```
// Create the table
string path = Windows.Storage.ApplicationData.Current.LocalFolder.Path;
var db = new SQLiteConnection(path + @"\foofoo");
db.CreateTable<Stock>();

// Create a list of stock-objects and insert them in the database
List<Stock> col = new List<Stock>();
for (int i = 1; i < 100000; i++)
{
    col.Add(new Stock() { Symbol = "symb " + i });
}
db.InsertAll(col);
```

Fill ListView (synchronous)

```
// Clear the bound list
_Stocks.Clear();
itemsListView.ItemsSource = null;

string path = Windows.Storage.ApplicationData.Current.LocalFolder.Path;
var db = new SQLiteConnection(path + @"\foofoo");

// query is a select over all items
var query = db.Table<Stock>();

DateTime startTime = DateTime.Now;

// Bind the query to the view
itemsListView.ItemsSource = query;
```

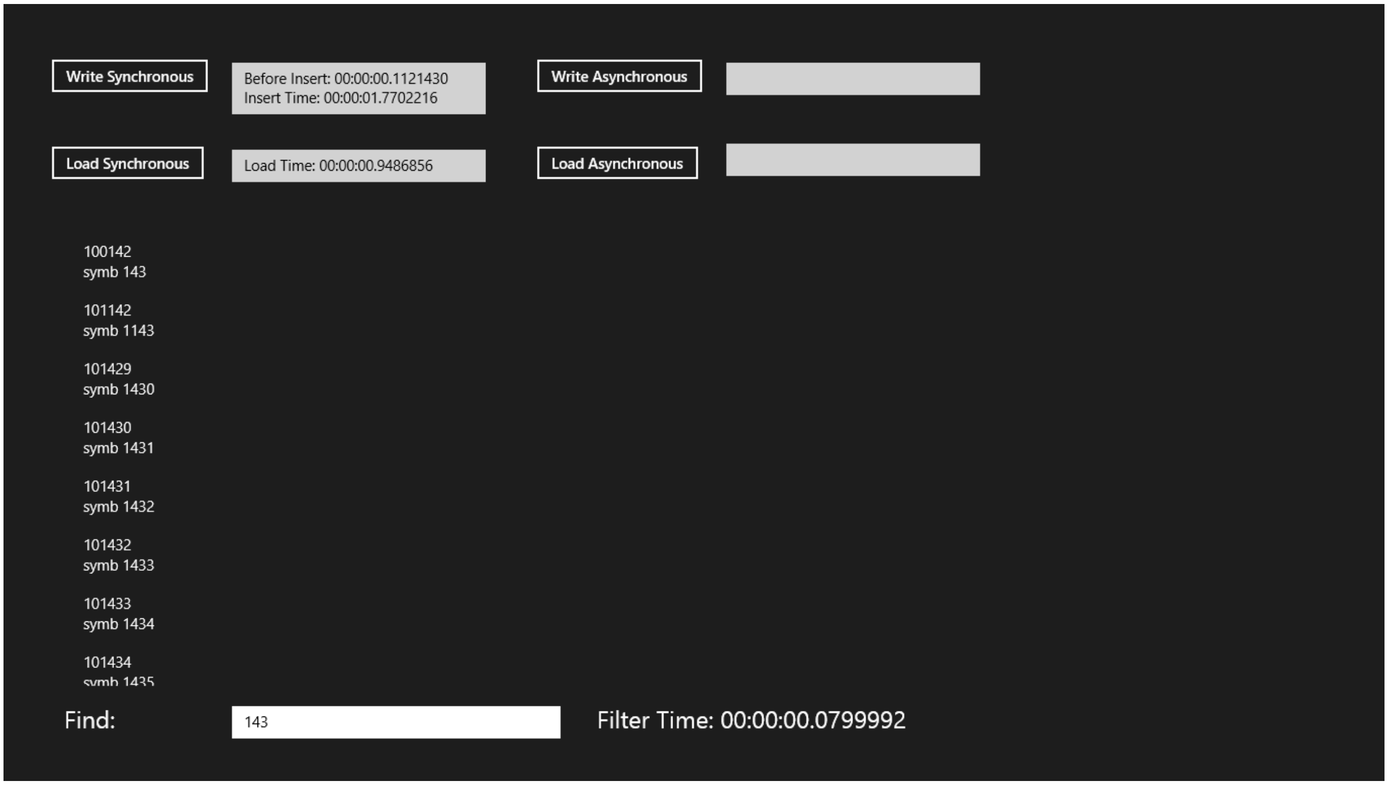
Find

2 seconds after the last entry in the searchbox the DispatchTimer ticks and filters the ListView.

```
void timer_Tick(object sender, object e)
{
    timer.Stop();
    if (findTextBox.Text != string.Empty)
    {
       itemsListView.ItemsSource = null;
       string findText = findTextBox.Text;
       string path = Windows.Storage.ApplicationData.Current.LocalFolder.Path;
       var db = new SQLiteConnection(path + @"\foofoo");

       // Create a select statement with lambda expressions
       var query = db.Table<Stock>().Where(s => s.Symbol.Contains(findText));

       itemsListView.ItemsSource = query;
      }
}
```

Performance of SQLite

As demanded by CoreSystems, it should be possible to write 100’000 objects in the database and show them in a list – in a timely fashion. We created 100000 objects, each with a reference to another object. The write is filling an empty database; the read is a binding to a listview through a join-operation on the database.



Before Insert: 00:00:00.1121430
Insert Time: 00:00:01.7702216

Load Time: 00:00:00.9486856

```
100142
symb 143
101142
symb 1143
101429
symb 1430
101430
symb 1431
101431
symb 1432
101432
symb 1433
101433
symb 1434
101434
symb 1435
```

Find: 143

Filter Time: 00:00:00.0799992

Figure 58: Prototype for SQLite database transactions

Insert the data (synchronous)	1.5 s
Read data and bind it to a ListView (synchronous)	0.9 s
Insert the data (asynchronous)	1.7 s
Filter Data	0.07 s

Nested Transactions

There are several ways to use nested transactions in SQLite [5]. The easiest way would be using the method RunInTransaction provided by SQLite-Net:

```
db.RunInTransaction(() =>
{
    Stock s = new Stock() { Symbol = "symb " + i };
    int key = db.Insert(s);

    Valuation v = new Valuation() { Price = 0.8m * i++, Time = DateTime.Now, StockId =
key };

    db.RunInTransaction(() => db.Insert(v));
});
```

The inserts for the Stock object and the Valuation run nested. If the insert of Valuation fails, it will just rollback this transaction.

This behavior can also be achieved manually:

```
try
{
    var savePoint = db.SaveTransactionPoint();
    db.Insert(valuation);
    db.Release(savePoint);
}
catch (Exception)
{
    db.RollbackTo(savepoint);
}
```

Important:

There is a bug in SQLite 3.7.13 for Windows RT which causes a CannotOpen-exception. This is caused because SQLite [5] tries to access a temp directory without the right permission. As a workaround the temp path has to be changed manually:

```
db.Execute("PRAGMA temp_store_directory = '" +
Windows.Storage.ApplicationData.Current.TemporaryFolder.Path + "'");
```

Joins

Ideally we would use a nice Linq-query to join tables. E.g.:

```
var query = from s in db.Table<Stock>()
            join v in db.Table<Valuation>() on s.Id equals v.StockId
            select new
            {
                s.Id,
                s.Symbol,
                v.Price,
                v.Time
            };
```

Unfortunately this is not supported yet in SQLite-Net. Again, there are different ways to achieve the goal. One would be to create a new object for the merged tables:

```
var query = db.Query<StockValuation>("SELECT s.Id StockId, s.Symbol Symbol, "
                + "v.Price Price, v.Time Time "
            + "FROM Stock AS s INNER JOIN Valuation AS v ON s.ID = v.StockId");
```

But this does not seem very practical. A fast and easy-to-use alternative is to query both tables and join them manually:

```
var stocks = db.Table<Stock>();
var vals = db.Table<Valuation>();

var query = from s in stocks.ToList()
            join v in vals.ToList() on s.Id equals v.StockId
            select new
            {
                 s.Id,
                s.Symbol,
                v.Price,
                v.Time
            };
```

While this solution is still not optimal, it is a good alternative to the unfortunately not implemented feature of SQLite-Net.

Multithreading

Since SQLite [5] itself is not thread safe we have to be careful when using it, so that we do not access the database from different threads at the same time.

Conclusion

SQLite [5] is both simple and fast. The results of the performance tests show, that the database access will not be a bottle neck in the synchronization and persistence. Since there is no real alternative to SQLite [5] at the moment, these are indeed very good news. Also, the SQLite [5] team is working closely with Microsoft, so the Microsoft Store Validation should not be a problem.

But as descripted in the architecture chapter, SQLite [5] has its shortcomings, which we became aware of only later in the project. So we did not choose SQLite [5]. The reasoning behind this is documented in the architecture chapter.

8.1.5 Windows 8 Quota

There seems to be no local quota for storing application data (like our database), however the roaming quota is limited to 100 kb. But since this is only for application settings, it is enough space for our application. [11]

8.1.6 Coresystem Protocol

8.1.6.1 Study Coresystems Synch Protocol/JSON

After having received the documentation of the Coresystems Synch Protocol Version 4, our goal was to ensure not to run in any problems of connecting to the cloud. Therefore, we went through the documentation and created a small feasibility prototype which fetches objects from the REST-based web service of the Coresystem business data cloud. The Prototype can also update data in the cloud. It is found in the TFS repository.

8.1.6.2 Json.NET vs Windows JSON

James Newton-King, the creator of Json.NET [12], wrote an interesting read on the differences between Json.NET [12] and the new Windows.Data.Json, which was introduced with Windows 8. [12]

After testing both frameworks, we chose to use Json.NET [12]. It is simpler than the Windows variant and more importantly, it is very well documented and widely used in the community. Also it runs on Windows RT.

8.1.6.3 Synchronization

For testing the Synchronization we used a simplified JSON object, derived from what Coresystems provided us as an example:

```
{
  "header": {
    "protocolVersion": 4,
    "verboseSync": true,
    "requestedDTOs": [
      {
        "name": "BusinessPartner",
        "version": 11,
        "forceUpdate": false,
        "lastSyncStamp": 0
      }
    ]
  },
  "body": {
    "businessPartners": {
      "newOrUpdatedObjects": [
      ],
      "deletedObjects": [
      ]
    }
  }
}
```

Essentially we just want to get the BusinessPartner objects from the cloud and show them in a list.

So we parse this request

```
JObject obj = JObject.Parse(json);
```

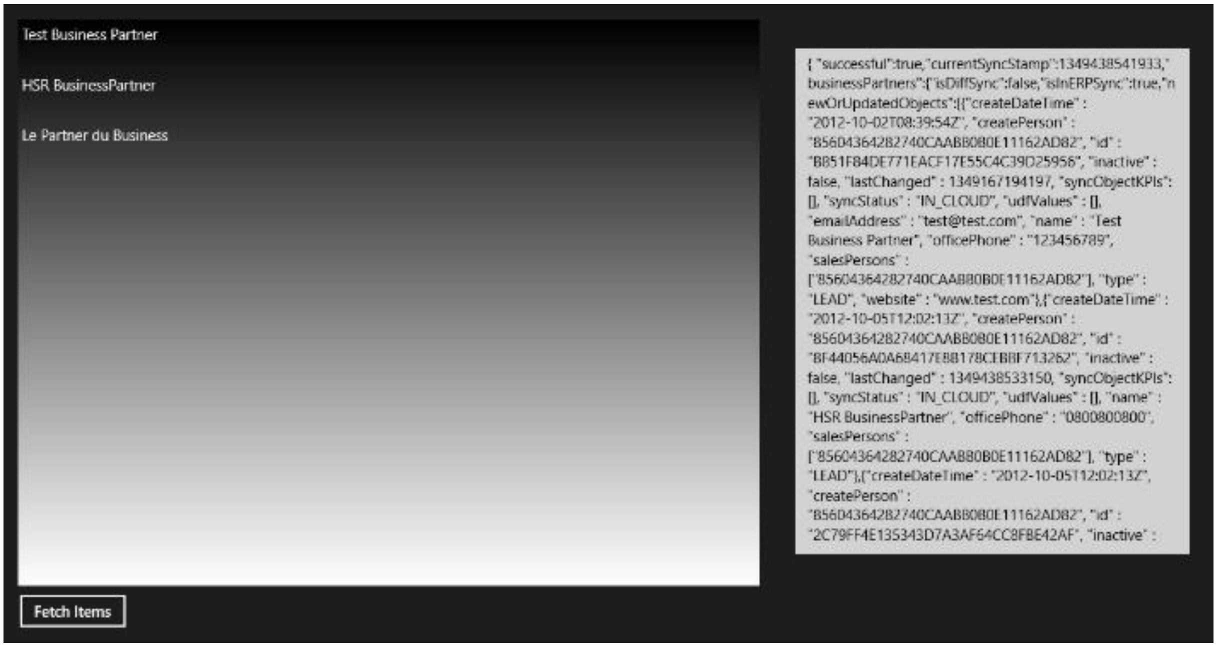
create an HTTP-request

```
HttpContent content = new StringContent(obj.ToString());
HttpResponseMessage response = await http.PostAsync(uri,  content);
```

and write the objects in a list

```
string rep = await response.Content.ReadAsStringAsync();
var json = JObject.Parse(rep);

var businessPartners =
     from p in json["businessPartners"]["newOrUpdatedObjects"].Children()
     select (string)p["name"];

foreach (var item in businessPartners)
{
     businessList.Items.Add(item.ToString());
}
```

This is the Result:



Test Business Partner

HSR BusinessPartner

Le Partner du Business

```
{ "successful":true,"currentSyncStamp":1349438541933,"
businessPartners":{"isDiffSync":false,"isInERPSync":true,"n
ewOrUpdatedObjects":[{"createDateTime" :
"2012-10-02T08:39:54Z", "createPerson" :
"B56043642B2740CAABB080E11162AD82", "id" :
"BB51F84DE771EACF17E55C4C39D25956", "inactive" :
false, "lastChanged" : 1349167194197, "syncObjectKPIs":
[], "syncStatus" : "IN_CLOUD", "udfValues" : [],
"emailAddress" : "test@test.com", "name" : "Test
Business Partner", "officePhone" : "123456789",
"salesPersons" :
["85604364282740CAAB80B0E11162AD82"], "type" :
"LEAD", "website" : "www.test.com"},{"createDateTime" :
"2012-10-05T12:02:13Z", "createPerson" :
"85604364282740CAABB080E11162AD82", "id" :
"BF44056A0A68417E88178CEBBF713262", "inactive" :
false, "lastChanged" : 1349438533150, "syncObjectKPIs":
[], "syncStatus" : "IN_CLOUD", "udfValues" : [], "name" :
"HSR BusinessPartner", "officePhone" : "0800800800",
"salesPersons" :
["85604364282740CAA880B0E11162AD82"], "type" :
"LEAD"},{"createDateTime" : "2012-10-05T12:02:13Z",
"createPerson" :
"B56043642B2740CAABB080E11162AD82", "id" :
"2C79FF4E135343D7A3AF64CC8FBE42AF", "inactive" :
```

Figure 59: Simple fetch from cloud

8.1.6.4 Conclusion

Using the Synch Protocol in combination with Json.NET [12] proved to be a very simple and effective approach. Although the small prototype does not yet address conflict behavior, it demonstrates the feasibility of the essential fetch communication.

Granted, the small prototype does not define any DTOs and does not use the synchronization options provided by the protocol, but still it has helped to form the understanding of the communication as a foundation for challenges to come.

8.2 Coresystems DTOs

For insight please contact us

8.3 Coresystems sync protocols version 4

For insight please contact us

8.4 Field Report (In German)

8.4.1 Timon Brüllmann

Für mich persönlich war diese Bachelorarbeit eine Supererfahrung. Natürlich habe ich bei meiner Abschlussarbeit positive als auch negative Erfahrungen gemacht. Zu den positiven Erfahrungen zählen sicherlich, dass ich jetzt ein breiteres Wissen über die .NET Technolgie von Microsoft besitze. Desweitern konnte ich mich während der Arbeit stark mit dem neue User Interface von Windows 8 beschäftigen. Positive fand ich auch die sehr gute Zusammenarbeit mit unserem Industriepartner coresystems. Sie halfen uns bei technischen Fragen und gaben uns immer konstruktives Feedback und Denkanstösse. Was ich ebenfalls sehr geschätzt habe, war die gute Zusammenarbeit mit Amir und Thomas. Probleme und Schwierigkeiten konnten problemlos diskutiert werden und Entscheidungen wurde rasch getroffen.

Die Betreuung durch Herrn Prof. Dr. Luc Bläser war in meine Augen ebenfalls sehr positiv. Herr Bläser hat uns in den wöchentlichen Sitzungen immer Verbesserungsvorschläge und Ideen zur Umsetzung gegeben.

Was ich als negativ empfunden habe und bei meinen nächsten Projekten verbessern werde, ist sicherlich die Zeitplanung. Gegen Ende des Projekts stellte sich klar heraus, dass wir uns klar verschätzt haben.

Rückblickend lässt sich sagen, dass dies eine sehr lehrreiche und spannende Projekt war. Ich möchte mich ebenfalls bei Amir Celebic und Thomas Geiser für die gute Zusammenarbeit bedanken.

8.4.2 Thomas Geiser

Nach anfänglichen Schwierigkeiten, war ich sehr froh endlich die Bachelorarbeit beginnen zu können. Es begann eine spannende und bereichernde Phase, als wir uns an dem Technologiestudium zuwandten. Vielleicht zu spannend - wie sich später herausstellen sollte, ging da wohl ein bisschen zu viel Zeit verloren. Aber dazu später mehr.

Als wir uns schliesslich voll auf das Projekt konzentrierten, bekamen wir es fast täglich mit neuen Herausforderungen zu tun. Wie setzen wir unsere geplante Architektur in dieser Technologie um? Was sind die best-practises in Windows 8 Apps? Nun, die zweite Frage konnten wir nur anhand von Tutorials und Styleguides erahnen. Generell fehlte uns häufig eine Hilfestellung in Form von Erfahrungsberichten, was allerdings nicht grösser Überraschte, da ja die Technologie sehr neu und stellenweise anders zu den .NET Vorgängern ist. Aber genau das machte auch den Reiz aus.

So nehme ich den ganzen Programmieraspekt der Arbeit als positive Erfahrung mit. Ebenfalls positiv, war die Zusammenarbeit mit unserem Industriepartner coresystems. Wir konnten unbeschwert Fragen stellen und bekamen in den Meetings jeweils guten Input. Diesen Input und Konzepte im Team mit Amir und Timon zu diskutieren und umzusetzen zu können werte ich ebenfalls als positive Erfahrung.

Froh war ich auch, dass wir mit Herrn Prof. Dr. Luc Bläser einen engagierten Betreuer gewinnen konnten, der sich immer Zeit nahm um unseren Fortschritt in Meetings zu besprechen und uns auch mal wieder auf den richtigen Pfad schubste, wenn wir uns in Details zu verzetteln anfingen.

Dies war manchmal durchaus ein Problem, was sich durch eine sehr arbeitssame Schlussphase zeigte. In Künftigen Projekten werde ich das Augenmerk wohl früher auf produktionsfähigen Code und eine gute Architektur legen, anstatt zu viel Zeit auf ein Proof-of-Concept eines Details des Gesamtprojektes zu verschwenden.

Alles in allem war es ein gutes Projekt und ich möchte mich herzlich bei allen beteiligten bedanken.

8.4.3 Amir Celebic

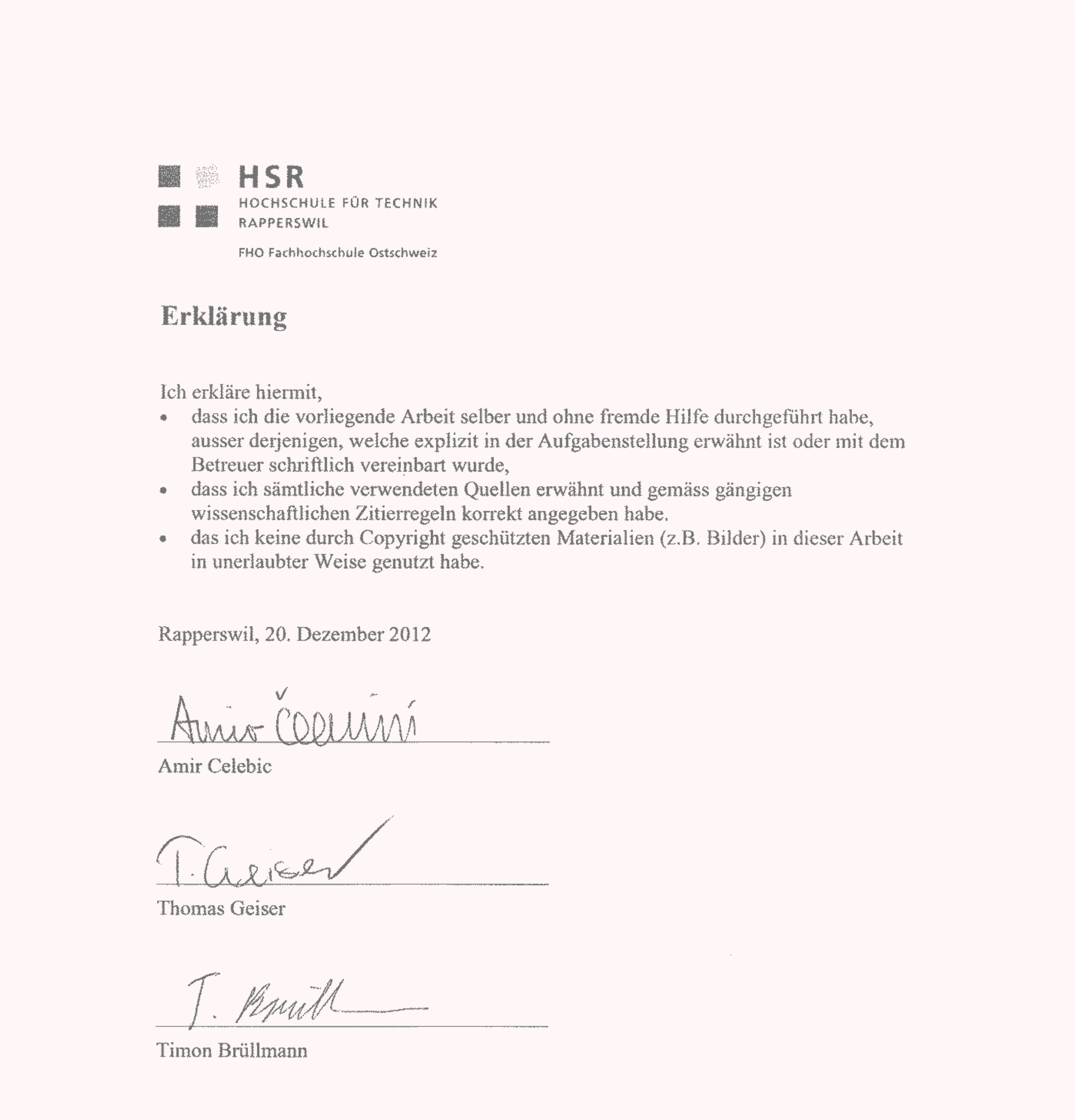
Wie ich schon in meiner Studienarbeit festgestellt habe und dies auch als sehr positiv empfand, arbeite ich sehr gerne in grösseren Projekten bzw. Arbeiten im Team. Dies vor allem, weil man sich in einer Gruppe sehr gut ergänzen kann und jedes Mitglied sowohl seine Schwächen als auch seine Stärken hat, letztere sich aber sehr positiv auf den Einzelnen und die Gruppe auswirkt und man viel dabei lernen kann. Auf diesem Weg will ich mich bei Timon Brüllmann und Thomas Geiser für die gute Zusammenarbeit bedanken!
Interessant war auch immer wieder festzustellen, wie mächtig das .NET Framework ist. Das Modul Microsoft Technologien hatte ich zuvor besucht und erfolgreich bestanden, jedoch waren mir die Möglichkeiten, welche die Technologie zu bieten hat nicht so bewusst.
Zudem war die Zusammenarbeit mit unserem Industriepartner coresystems sehr erfolgreich und sie haben uns jederzeit geholfen und uns unterstützt. Auch Ihnen gebührt ein grosses Dankeschön!

Zudem möchte ich mich bei Herrn Prof. Dr. Bläser für die Unterstützung bedanken. Die wöchentlichen Meetings waren immer sehr aufschlussreich und haben uns für unsere Arbeit sehr geholfen.

Als negativen Punkt, wenn man das so sagen darf, empfand ich die Projektplanung. Wir waren ein bisschen zu enthusiastisch, hatten uns für die begrenzte Zeit zu viel vorgenommen und die Komplexität unterschätz. Die Ziele mit der höchsten Priorität wurden aber trotzdem erreicht und somit war das resultierende Produkt ein Erfolg.

Ich möchte mich nochmals bei allen Beteiligten für die sehr anstrengende aber tolle Zeit bedanken!

8.5 COPYRIGHT STATEMENT (IN GERMAN)



HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Rapperswil, 20. Dezember 2012

Amir Čelebić

Amir Celebic

T. Geiser

Thomas Geiser

T. Brüll

Timon Brüllmann

8.6 REFERENCES

[1] Microsoft, Windows 8 - User experience guidelines, Redmond, 2012, p. 6.

[2] "SemanticZoom class (Windows)," Microsoft, 12 April 2012. [Online]. Available: http://msdn.microsoft.com/library/windows/apps/Hh702601. [Accessed December 2012].

[3] S. Raiten, "Using Bing Maps For Windows 8 Metro Apps - C# / JavaScript," 2012. [Online]. Available: http://www.codeproject.com/Articles/408457/Using-Bing-Maps-For-Windows-8-Metro-Apps-Csharp-Ja.

[4] "siaqodb homepage," 2012. [Online]. Available: http://siaqodb.com/. [Accessed 30 October 2012].

[5] "SQLite," 2012. [Online]. Available: http://sqlite.org/index.html.

[6] C. Bilgin, "Logging Sample for Windows Store Apps," 10 10 2012. [Online]. Available: http://code.msdn.microsoft.com/windowsapps/Logging-Sample-for-Windows-0b9dffd7#content.

[7] A. Mathias, "Code coverage Results is disabled in Visual Studio ultimate 2012 RC.," 19 June 2012. [Online]. Available: http://social.msdn.microsoft.com/Forums/en-US/vstest/thread/8407ecd1-2981-4e44-bb9b-a6956e93de1b/.

[8] "Cross-Platform Development with the .NET Framework," Microsoft Inc., 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/gg597391.aspx.

[9] "Application resources and localization sample," 2012. [Online]. Available: http://code.msdn.microsoft.com/windowsapps/Application-resources-and-cd0c6eaa.

[10] "Application resources and localization sample," 2012. [Online]. Available: http://code.msdn.microsoft.com/windowsapps/Application-resources-and-cd0c6eaa.

[11] D. Bennett, "Windows 8 app developer blog," 18 July 2012. [Online]. Available: http://blogs.msdn.com/b/windowsappdev/archive/2012/07/17/roaming-your-app-data.aspx.

[12] J. Newton-King, "Json.NET vs Windows.Data.Json," 14 September 2012. [Online]. Available: http://james.newtonking.com/archive/2012/09/14/json-net-vs-windows-data-json.aspx.

[13] coresystems ag, "Sync Protocol Version 4," p. 16, 2012.

8.7 LIST OF FIGURES

8.8 GLOSSARY

API	Application Programming Interface
HTTP	The Hypertext Transfer Protocol (HTTP) is a very common protocol for transferring content on the internet.
JSON	JavaScript Object Notation (JSON) is a notation for representing objects in a string representation.
Restful	Representational State Transfer
ERP	Enterprise Resource Planning
DTO	Data Transfer Object
App	Short for application
SAP	ERP Software Company
Windows RT	Windows on ARM processors
CRUD	Create, Read, Update, Delete operations
NoSQL	Object orientated database
gzip	Open source compression library
SQL	Structured Query Language