VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILE CLIENT SERVER APPLICATION

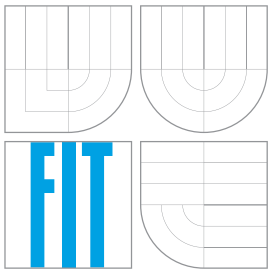
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB DOHNAL

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE TYPU KLIENT/SERVER

MOBILE CLIENT SERVER APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB DOHNAL

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Dr. Ing. PAVEL ZAMČÍK,

BRNO 2015

Abstrakt

Tato diplomová práce se zabývá vývojem mobilní aplikace typu klient-server na platformě Windows Phone. Obsahuje popis platformy Window Phone, vývojové prostředí a jeho nástroj pro ladění a sledování prostředků na této platformě. Rozebírá architektury a protokoly využívající se pro model klient-server. Popisuje sdílení dat a zasílání zpráv mezi uživateli klientem a serverem. Srovnává dostupné protokoly pro komunikaci. V další kapitole je využití paměti na klientu při nedostupnosti připojení k internetu. Závěr se věnuje vizi dalšího vývoje projektu.

Abstract

This thesis is focused at the mobil client-server application development. First chapter contains Windows Phone platform description, software development environment and tools for debugging and monitoring resources. The Master's thesis analyzes client-server architectures and protocols and discuss about users data sharing and instant messaging. Thesis describes XMPP properties and utilization in this field. Thesis deals with using cache memory for offline mode and in the end is vision of further development of the project.

Klíčová slova

Mobilní aplikace, Window phone, klient, server, notifikace, sdílení, mezipaměť

Keywords

Mobile application, Window Phone, client, server, notification, sharing, cache

Citace

Jakub Dohnal: Mobile Client Server Application, diplomová práce, Brno, FIT VUT v Brně, 2015

Mobile Client Server Application

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana profesora Pavla Zemčika.
Další informace mi poskytl Ing. Aleš Láník.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Dohnal
May 27, 2015

Poděkování

Děkuji vedoucímu diplomové práce Prof. Dr. Ing. Pavelu Zemčíkovi za cenné rady, připomínky, metodické vedení práce a za poskytnutí zařízení k vývoji aplikace. Dále děkuji za odbornou pomoc, kterou mi poskytl Ing. Aleš Láník.

© Jakub Dohnal, 2015.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Contents

Chapter 1

Introduction

Today we use mobile phones for calling and sending SMS, but also for gaming, surfing, sending emails, reading news, checking bank accounts, etc. Due to hardware and internet network evolution we can keep in touch and exchange information with another people or just consume the content which is delivered to the phone by mobile network. This gives us a significant platform for creating useful applications, and we can think of the phone as a miniature computer.

This thesis describes architecture and infrastructure design of client/server model, proposes communication structure between client (application) and server, and notes how the application should work without connection to the server. Also it shows the creation of such applications. It compares the client/server architectures and indicates tools and development for the environment of Windows Phone platform.

Development of such applications depends on the mobile network infrastructure. In the past, mobile applications used mainly local content without internet connection or they were getting content from servers. With the development of network infrastructure, user-created data can be sent to the server. In the past, the content was mainly textual. Today it is possible to stream a video from a server to an application, and from an application to the server.

The implementation is a real application intended for specific field, and it does not discuss only the theory, but also the practical use.

The goal of this thesis is to select a suitable application that should use sharing content between the application throw a server, and exchanging data with a server, e.g. of image and texts. We will get acquainted with mobile application development in Windows Phone environment. A method of implementation and testing will be chosen. The application or its necessary fragment will be implemented to test the above mentioned principles of communication and notifications.

Chapter 2 describes client/server model, its architecture options, and existing protocols for communication between the client and the server. The Windows Phone platform and developers tools are described in Chapter 4. In Chapter 5, work plan and comparison of current technologies and specifications of the final application are given.

Chapter 2

Client Server model

It is the network structure separating a client and a server. The client can be understood as the application running on end devices. This kind of model is called application server model. The application communicates with the server via network, and usually the server is connected to the Internet. The application is an active element and initiates communication, or registers here the client's network location for addressing from the server, and the server mainly responses on application requests. This model has usually only one server which is uniquely addressed. On the server, some services are running which is a layer for serving data for example from database to application, depending on the requests. However, applications are numerous and can run on different hardware and software platforms. Communication between an application and a server is clearly specified by API. This model is widely used in applications that use the network. For applications, content or multiplayer games are shared [5].

An important stage in design is to determinate the system architecture and define the underlying infrastructure [1].

2.1 Architecture and infrastructure design

The architecture is the arrangement of a client platform, server platform and network, and the outline split of functionality and data (presentation, application and database components) between client and server platforms.[2] Architecture of a client/server system can be defined in a number of different ways [2]. The differences between the varieties can be seen in how the presentation, application and database components of the system are distributed between the client and server platform:

1. Distributed presentation
2. Separate presentation
3. Separate database
4. Distributed application
5. Distributed database
6. Distributed application and database

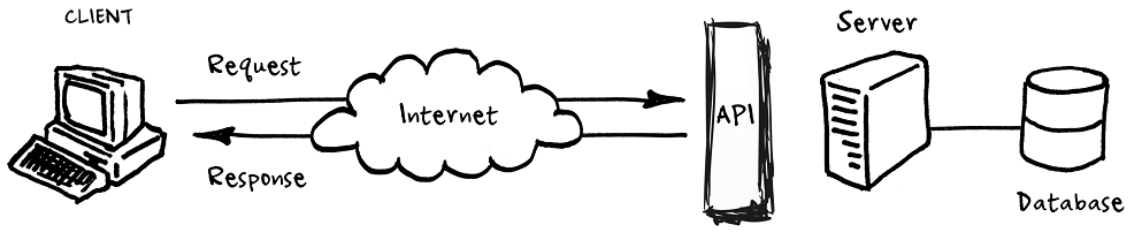


Figure 2.1: Client/server connection [12]

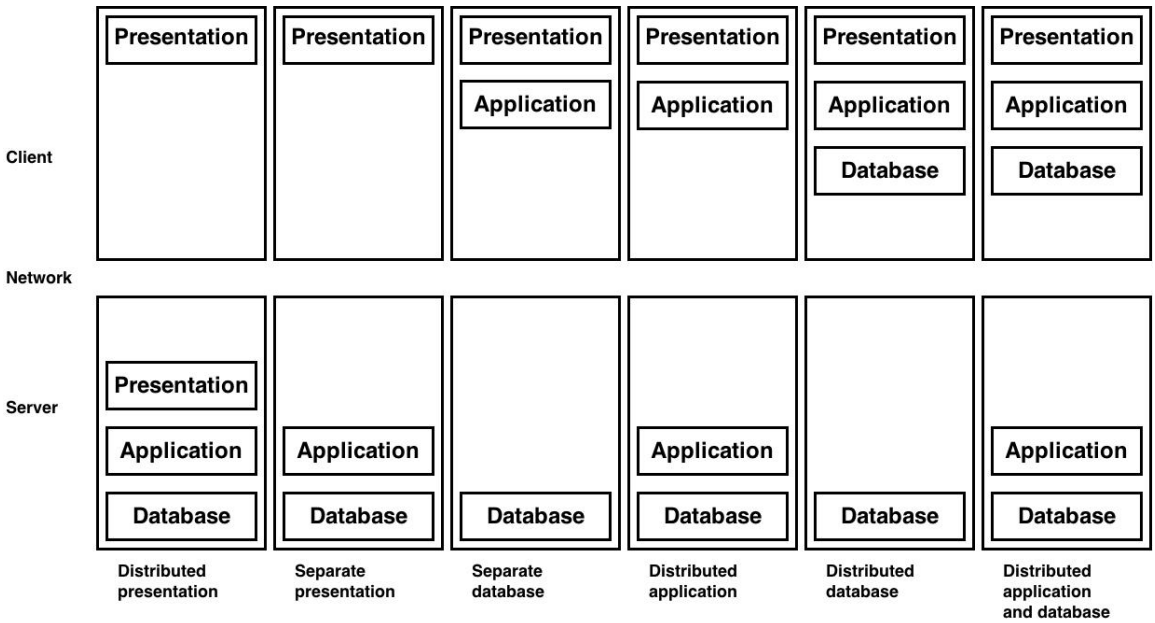


Figure 2.2: Categories of client/server architecture [2]

Distributed presentation

The presentation component is split between the client platform and the server platform, with application and database components located entirely on the server platform. An example of such a system is the applications displaying remote desktop [2].

Separate presentation

The presentation component is located entirely on the client, and application and database components are located entirely on the server. Common example is **X Window System** presentation, driven from a remote application [2].

Separate database

Presentation and application components are located entirely on the client platform and database is on the server. The most common client/server architecture is used by many vendors of DBMS and user tools. An example is Executive Information System (EIS) where data from a corporate database can be accessed using off-the-shelf analyses tools or custom applications [2].

Distributed application

Presentation and part of application are located on the client, and the database and the rest of the application are located on the server. The application functionality can be split *vertically* or *horizontally*. Vertical split is when each function is entirely on one of client or server. Where some parts of the functionality are on the client and communicate with the other part on the server, it is horizontal split [2].

Distributed database

This includes both presentation and application entirely on the client platform, with the database split between platforms. Data can be split vertically or horizontally or replicated. Distributed database split vertically distribute different tables on each platform. Horizontally split database distributes different rows from tables on different platform. Only some data on both platforms can be replicated [2].

Distributed application and database

It's combination of the distributed application and distributed database architecture. Application and database are split across platforms and presentation is entirely on the client [2].

Client cache

Applications with client-server model are dependent on network connectivity. An application without client cache cannot be used if internet connection is unavailable. If the application has a client cache, it can serve data to user without internet connection. There are more methods to cache data on client. The basic one is to save response in serialized form on unique request to memory or to persistent storage with expiration date. Request can be cached only if has always same response. The cache has Key/Value format.

Another way is to store deserialized data on client in database. Application fetches data for display from database. If user wants to update or added data. At first, local database is updated, and after that it is uploaded to the server. This model gets application powerful to use it without connection with server but brings problems with data consistency.

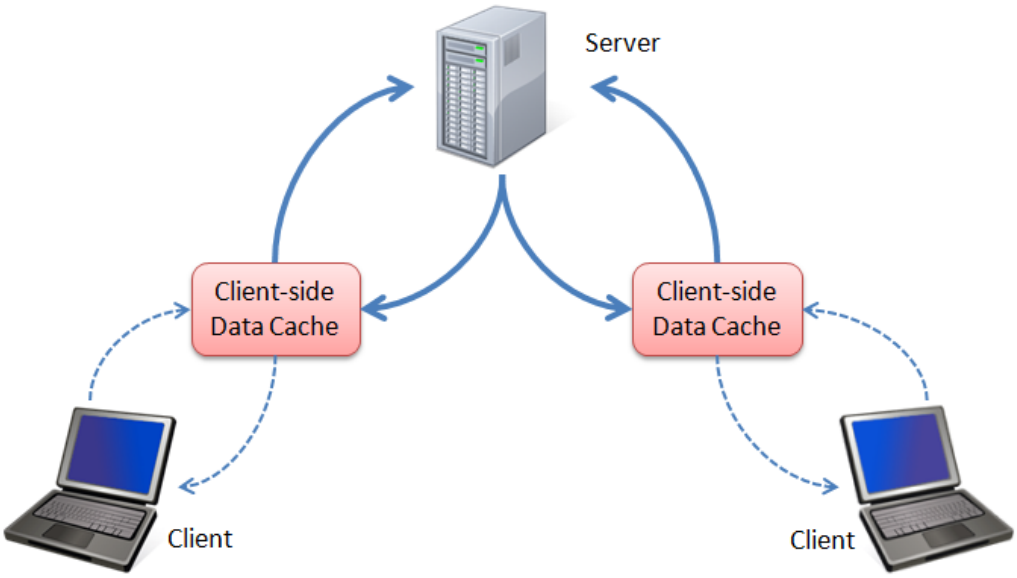


Figure 2.3: Client cache diagram [9]

Chapter 3

Communication

For communication between the application and the server, a communication protocol is used to facilitate addressing, message format, data format and authentication. It is possible to use one of these protocols or architectures:

- SOAP
- XML-RPC
- REST
- XMPP

3.1 SOAP

Simple Object Access Protocol is a lightweight protocol intended for exchanging structured information in XML format. It can be the basis for a Web protocol stack, and provides the basic framework for the messaging for Web services. Its main characteristic is expandability, neutrality (can operate on any transport protocol such as HTTP, SMTP) and independence (it is possible to utilize any programming model). One of its disadvantages is the use of XML for formatting data. Messages are so large and can handle long time. The SOAP contains its own header as shown in Figure 4.1 [4].

3.2 XML-RPC

XML-RPC is a Remote Procedure Calling protocol that works over the Internet. An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML. Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures. Unless it has a lower-level error, it always returns HTTP code 200 OK [3].

3.3 REST

The Representational State Transfer (REST) architectural style was developed in parallel with HTTP/1.1. This design pattern is used as a set of instructions to create a web service

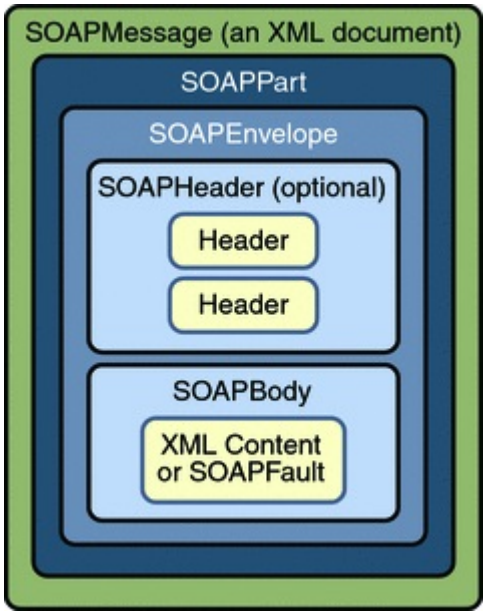


Figure 3.1: SOAP message structure [10]

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT


<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>4</int></value>
               </member>
            <member>
               <name>faultString</name>
               <value><string>Too many parameters.</string></value>
               </member>
            </struct>
         </value>
      </fault>
   </methodResponse>
```

Figure 3.2: XML-RPC request example [3]

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT


<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
         </param>
      </params>
   </methodResponse>
```

Figure 3.3: XML-RPC response example [3]

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT


<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
         </param>
      </params>
   </methodResponse>
```

Figure 3.4: XML-RPC fault example [3]

that enables network-connected device to communicate with one another through a shared basic HTTP communication protocol. HTTP components are defined by four basic methods (verbs) to communicate with the server: GET, POST, PUT and DELETE. Method **GET** has no message body and is used for fetching data from the server. The POST method uploading a new records on the server the records are contains in message body structured by JSON or XML. The **PUT** method is almost same such as **POST** method but the records are update on server and **DELETE** method is for deleting record on the server [6].

In REST architecture the URI *http://server.cz/users* is used to define resources. A method decide on the action to be performed. URI http://server.net/resources/ example, in the case of the method:

- **GET** returned URI list of all members of the collection
- **POST** creates a new record in the collection
- **PUT** replaces a whole collection of other
- **DELETE** deletes the entire collection

Using a URI like *http://server.net/resources/item* the actions of particular methods are defined as follows:

- **GET** returns a representation of the desired member of the collection represented the correct type
- **POST** usually not used
- **PUT** replaces addressed to a member of the collection. If not exist, create it
- **DELETE** deletes addressed to a member of the collection

3.4 XMPP

Extensible Messaging and Presence Protocol is a protocol for streaming XML elements in order to exchange information on a user's presence in almost real time [4]. His form and scalability describes RFC3921. His original name had been Jabber created by Jeremie Miller in 1998 and modified by Jabber open-source community in 1999 [5]. Thanks to the extensibility the protocol can be used in many applications with client-server model, operating in near real time. These include publish-subscribe systems, signalling for VoIP, file transfer, connection *Internet of things* Intelligent networks and social networks [8].

XMPP network

It consists of XMPP clients and XMPP servers. Architecture of XMPP is a decentralized network, similar to e-mail. Anyone can create and run their own XMPP server for your domain. It uses a client-server model and therefore the connection between clients is not straightforward, but the client is connected to your home server. All client communication passes through the home server and XMPP servers communicate among themselves. The Figure 2.4 shows how such communication between clients looks. The client sends a message to the home server with Jabber ID (JID) to whom the message is intended. JID contains the recipient's home server domain. If the recipient's home server is blocked, the server

returns an error message to the sender. The sender server sends a message to the recipient's server. The recipient home server eventually sends a message to the client defined by JID. If the recipient is blocked, the server returns an error message. Recipient's home server is responsible for delivering a message if the recipient is not connected. JID is a unique identifier for a client similar to an email address *username@server.domain[/mobile]*. It consists from a username and domain server separated by *at (@)*.

The original and native transport protocol XMPP is the Transmission Control Protocol (TCP), using open XML stream over the long-term TCP connections. As an alternative to TCP connection, XMPP community has developed HTTP streaming for Web clients and users for strict firewalls. The original specification for HTTP XMPP can be used in two ways: **Polling** and **Binding** method [8]. Using the method of polling (polling) the server must have a big database to store messages, and if the client is active and always request in a time period if there is a new message for him. For requesting is used HTTP **GET** and **POST** method.

Binding method is implemented as a two-way stream of synchronous HTTP (BOSH) [17]. It lets server send messages to the client without client request as soon as the server has new data for the client. This method is much more efficient than polling where many requests have empty answers. The XMPP using HTTP transport protocol or TCP. Using HTTP allows that the connection has not been limited mostly firewalls. This fact is usually used when the TCP port 5280 for XMPP is blocked [17]. HTTP server listens on the TCP port and communications should run without restrictions. The last method is to use WebSocket who is providing full-duplex communication channel over a single TCP connection. WebSocket binding for XMPP is defined as a standard in IETF RFC6455 [11].

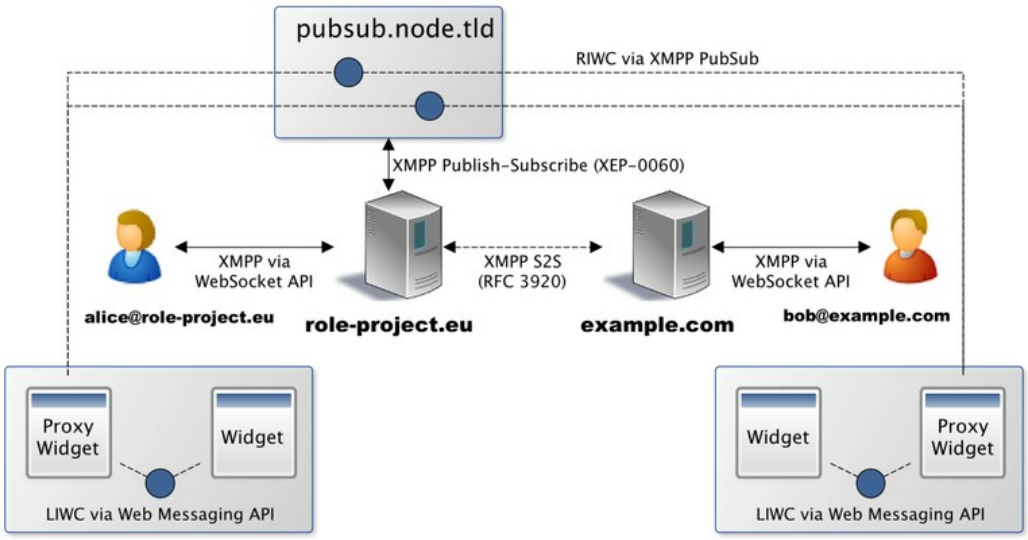


Figure 3.5: Example of XMPP network [7]

Chapter 4

Windows Phone

Windows Phone 8.1 is a mobile operating system from Microsoft Corporation. Microsoft Windows comes with a term Runtime app that identifies two types of applications. Windows Store app (application for PCs, tablets and laptops) and Windows Phone Store app (application for Windows Phone). Is it possible to create one project which contains separate projects for design and feature experiences unique to tablet & PC and Phone device, as well as a shared project that promotes the reuse of code relevant to both. This means that it is possible to write a single application for both mobile and PC platform including design. Of course, in some cases it is not so simple, and some of the frameworks and functions are not available for Windows Phone Store app [14].

Application design for Windows Phone is not limited, but it is recommended to follow one of the basic layout templates. What is strict defined is mandatory functions some of the controls. For example, how back button should behave. By definition, Microsoft's design focuses mainly on content. Windows phone don't have chrome elements and application design is formatting by content and alignment.

It's possible to develop Windows Store apps using C++, C#, Microsoft Visual Basic, and JavaScript. JavaScript uses HTML5 markup for UI layout, and the other languages use a markup language called Extensible Application Markup Language (XAML) to describe their UI.Here's a list of recomended languages for information type of application (displaying information such as weather, stock price, news, or social network updates) [18]:

- C# and XAML
- JavaScript and WinJS

Although this thesis is focusing on C#, the other languages offer unique benefits.

XAML

Extensible Application Markup Language (XAML) is a declarative language. Specifically, XAML can initialize objects and set properties of objects, using a language structure that shows hierarchical relationships between multiple objects, and using a backing type convention that supports extension of types. The XAML language supports interchange of sources between different tools and roles in the development process without information loss, such as exchanging XAML sources between Visual Studio and Blend for Visual Studio. XAML is the primary format for declaring a Windows Phone UI and elements in that UI [13].

WinJS

WinJS is Windows Library for JavaScript. WinJS provides high quality infrastructure like page controls, promises, and data-binding. Polished UI features like virtualizing collections and high performance Windows controls [16].

4.1 Windows Phone tools

The development environment is clearly defined and must use Microsoft Visual Studio 2013. They are in it to develop Windows Store applications, and Windows Phone applications. It includes project templates, code editor, and visual designer. There is also possible to find the test functions. The main tools are:

- Windows Phone Emulator
- Blend for Visual Studio
- Application Deployment tool
- Windows Phone Developer Power Tools
- Isolated Storage Explorer

Windows Phone Emulator

Hardware virtualization platform for testing applications on multiple devices. Windows phone emulator is dependent on the support of Hardware. For the run it requires a processor with Hyper-V technology. It is able to emulate the resolution and screen size, memory size, display settings, network settings, and the language of the region, debugging when application is in inactive state (application development lifecycle), local storage, microphone and lock screen. Functions as a compass, gyroscope, vibration control, backlight, display video in a resolution greater than VGA emulator, do not support.

Blend for Visual Studio

It is a collection of useful design tools for visual creation of Windows Store applications based on JavaScript, VB, C#, or C++. For applications built on JavaScript automatically apply tools for creating user interfaces in HTML5 and CSS3. For other languages it uses XAML.

Windows Phone Developer Power Tools

Use this tool to monitor the response, resource utilization and debugging in a crash. It consists of three powerful testing and debugging tools:

- **Application Verifier** - Detect subtle programming errors in native code
- **Performance Monitor** - Capture real-time performance metrics and visualize them graphically
- **Performance Recorder** - Collect system-wide logs and analyse them on your computer

```
<UserControl x:Class="MyWindowsPhone.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
  <Grid Background="OldLace">
  </Grid>
</UserControl>
```

Figure 4.1: Windows phone XAML example [13]

```
(function () {
    "use strict";

    var app = WinJS.Application;
    var activation = Windows.ApplicationModel.Activation;
    WinJS.strictProcessing();

    app.onactivated = function (args) {
        if (args.detail.kind === activation.ActivationKind.launch) {
            if (args.detail.previousExecutionState !== activation.ApplicationExecut
                // TODO: This application has been newly launched. Initialize
                // your application here.
            } else {
                // TODO: This application has been reactivated from suspension.
                // Restore application state here.
            }
            args.setPromise(WinJS.UI.processAll());
        }
    };

    app.oncheckpoint = function (args) {
        // TODO: This application is about to be suspended. Save any state
        // that needs to persist across suspensions here. You might use the
        // WinJS.Application.sessionState object, which is automatically
        // saved and restored across suspension. If you need to complete an
        // asynchronous operation before your application is suspended, call
        // args.setPromise().
    };

    app.start();
})();
```

Figure 4.2: Windows phone XAML example [16]

All tools are available for debugging applications in both the emulator and on the device with Windows Phone 8.1 . Testing continues even after you disconnect your phone from the computer and after reconnecting the output of tracking applications downloaded to your computer.

While **Application Verifier** is mainly used for debugging memory and determination of critical vulnerabilities, **Performance Monitor** shows real-time hardware metrics of resources such as CPU, memory, and input/output operations. **Record Performance** is a deeper analysis of the performance and resource consumption. Does not record only performance test applications, but also other services and jobs for detecting the impact of their influence on the running application.

Isolated Storage Explorer

With this tool it is possible to ensure that the application files are deposited in the right place, or to copy them for or to device.

4.2 Push Notification

This is an asynchronous, best-effort service that offers third-party developers a channel to send data to a Windows Phone app from a cloud service in a power-efficient manner. Depending on the format of the push notification and the payload attached to it, the info is delivered as raw data to the app, the app's Tile is visually updated, or a toast notification is displayed. MPNS returns a response code to the cloud service after a push notification is sent indicating that the notification has been received and will be delivered to the device at the next possible opportunity [15].

1. The application requests a push notification URI from the Push client service.
2. The Push client service negotiates with the Microsoft Push Notification Service (MPNS), and MPNS returns a notification URI to the Push client service.
3. The Push client service returns the notification URI to the application.
4. The application can then send the notification URI to the cloud service.
5. When the cloud service has info to send to the application, it uses the notification URI to send a push notification to MPNS.
6. MPNS routes the push notification to the application.

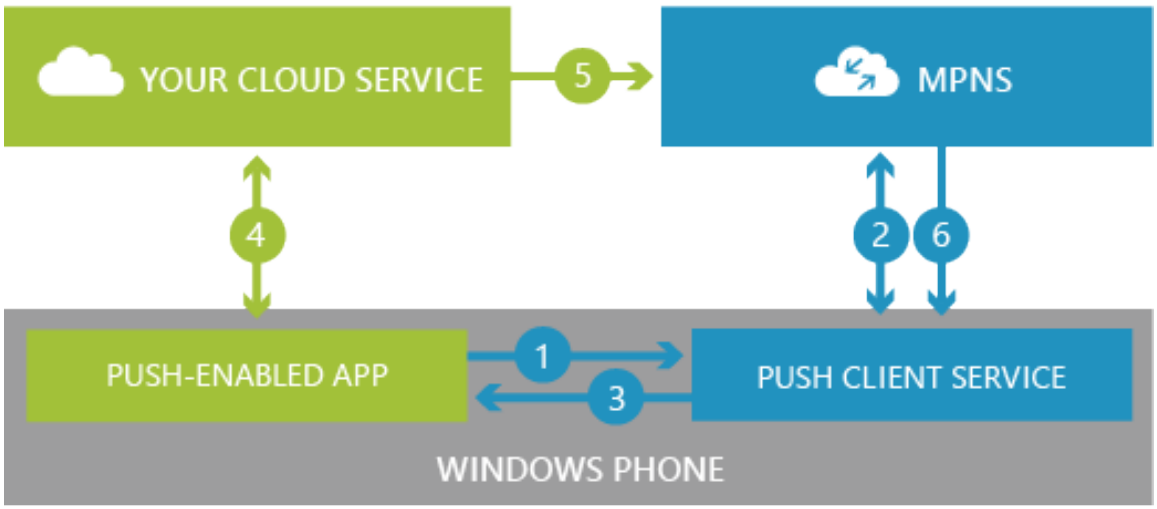


Figure 4.3: Push Notification diagram for Windows Phone [15]

Chapter 5

Current situation and work plan

This thesis is focused on client platform and final application is developed for Windows Phone 8.1 operation system. The main thing for client is which components are located and how to split some of them between him and server platform as described in Section 2.1. Client can be defined as Thin and Thick client. Thin client has Presentation component and eventually part of Application component, while database and main application components are on server. The Thick client contains Presentation and part or entire Application component and part or entire duplicated Database. In Table 5.1, pros and cons of both clients are compared side by side. For the user it is better if application is runnable in offline mode and the application does not have to communicate with server frequently. I chose the thick client because of many benefits.

The instance of application supports only one user who is login to the system. This user can observe his data and create new data in application. His data are data created by him or data which are shared with him. It is not right to copy the whole database to client platform but duplicate only some rows from tables. This approach requires to create same tables and relationships as on server. The server should decide which data belong to the user, and served them to the client. To use SQL database on Windows Phone 8.1 platform, SQLite library enables which allows to store data in SQLite database. This SQLite library also supports Object-relational mapping (ORM) for converting data between incompatible type systems in object-oriented programming languages.

The last main part of client/server model is communication between the client and the server. For communication can be used some of defined protocols or architecture, or a new one can be designed. SOAP and XML-RPC have defined message structure and there are many libraries to use them properly but they use XML to structure the messages. A message defined by XML is long and its processing takes a long time. XMPP architecture has same problems. To prevent problems with XML, the solution is to use JavaScript Object Notation (JSON). JSON is a lightweight data-interchange format and language independent. JSON can be simply deserialized to a Dictionary structure or to an Object. We chose to define custom protocol in JSON. The application does not need to communicate with server real-time. Information can be present on client after some time then is not necessary to use Long Polling but only Polling.

The work plane

1. Split application component between client platform and server platform
2. Design database for store user information

3. Define communication protocol and use JSON data-interchange format
4. Define polling policy
5. Implement application and use SQLite database
6. Apply Windows Phone tools for testing and debugging application
7. Show application to users and collect feedback

Table 5.1: Thin and Thick client comparison

Thin Clients	**Thick Clients**
Easy to deploy as they require no extra or specialized software installation	More expensive to deploy and more work for IT to deploy
Needs to validate with the server after data capture	Data verified by client not server (immediate validation)
If the server goes down, data collection is halted as the client needs constant communication with the server	Robust technology provides better uptime
Cannot be interfaced with other equipment (in plants or factory settings for example)	Only needs intermittent communication with server
Clients run only and exactly as specified by the server	More expensive to deploy and more work for IT to deploy
More downtime	Require more resources but less servers
Portability in that all applications are on the server so any workstation can access	Can store local files and applications
Opportunity to use older, outdated PCs as clients	Reduced server demands
Reduced security threat	Increased security issues

JSON Example

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Figure 5.1: JSON vs. XML example defines an employees object, with an array of 3 employee records

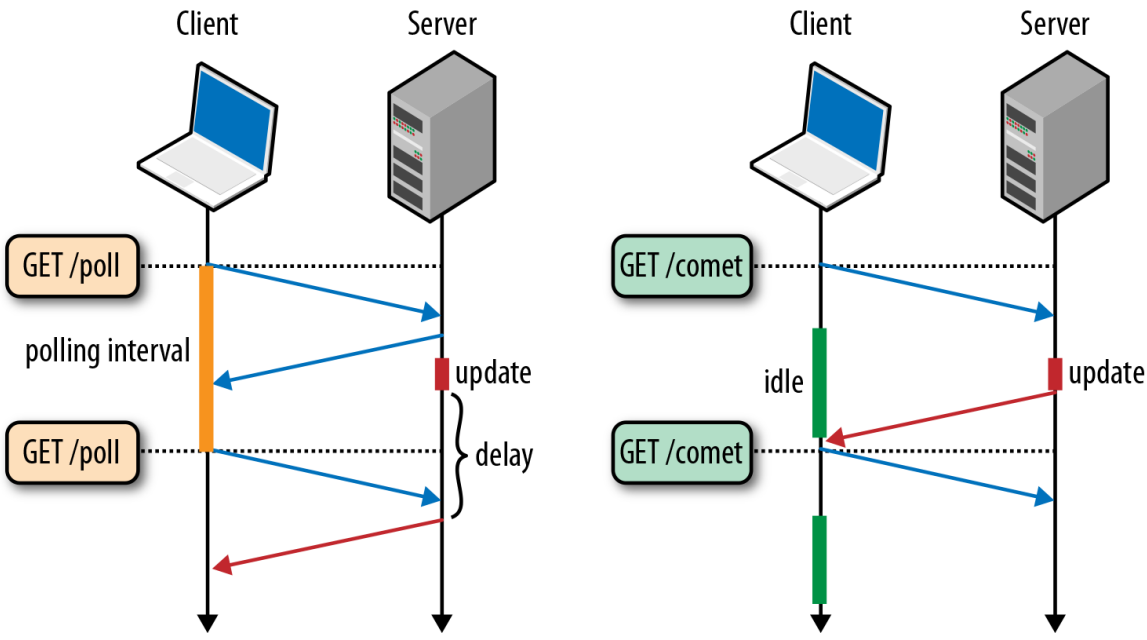


Figure 5.2: Polling (left) vs. long-polling (right) latency

Chapter 6

Proposal

The application communicates with Colla server which is the server storing the data for the construction supervision and the construction company. This data contains Projects construction company. The project has items which represent defects on construction. Defects can be recorded by images and texts. Each item has also discussion section where the defect can be discussed. It's possible to share the Project and the Item between users. The user who creates the Project or the Item can also edit this record and assigned to the other users. These users can add new images and discuss, what type of defect it is. Every record is stored on server and client has onlysome records.

The client controls assignment picture to items and assignment messages to items. User can take a photo of defects on client´s application. The client and the server share some part of application component as editing the Project detail and the Item detail. The client controls this by the User Interface and allow edit the Project only to users who have permission for this. The server controls the same action but informs client and response to the request. This duplication application part is mainly for security on the server and right User´s experience of the client.

6.1 Decomposition

The application is decomposed to three extensive blocks: *User Interface and Controls*, *Communication*, *Data* block and one *Polling Manager (Figure 6.1)*. The *User Interface* block communicates with the user by showing structured data and controlling inputs from the user. This block contains application logic for control what user can do and what he can not. The block creates instances of *Data Classes* which are used for storing in database. These data are passed down to the *Data* block to the *Data Provider* which saves these objects to the database. The *Data Classes* contains classes of database tables. The *Data Classes* are used for objective communication with the database. The *Data Provider* deserialize them and save into the database. The *User Interface* fetching the object from the *Data Provider* to show them and is given to the user. The *Data Provider* sends notifications to the *User Interface* about the data and what kind of data was changed in the database. The *User Interface and Controls* communicates only with the *Data Provider*.

The *Communication* block has *Api Interface* with collection of method for communication with server. *JSON API* implements this method and is connected with *Request Serializer* and *Response Deserializer*. *Polling Manager* initiates request to server by *JSON API* and plans next request. If in response from server is some data *Response Deserializer*

deserialize this data and send them to Data Providet. If the user creates new data the *Data Provider* sends them to *JSON API*. *Request Serializer* create proper message for server. *JSON API* sends data the server and the server responses if the data was stored or error if data was wrong or a different error message.

This structure is easy to change. If the communication between the client and the server will change and the server will start to use XML instead JSON it's neccessery only change the *Request Serializer* and *Response Deserializer*. This situation is the same with the Databse. The Data Classes will remain the same or with only small changes. The change in one of block shouldn't ovlivnit the other block.

Next think is to propose application screent flow *(Figure 6.2)*. Storyboard has all screens which should appeare in application. It shows how application can be used and how can be using. Defined flow is example what user see whan turn on application and where he can go. Also define screen following some action. The first start is defined as *Start I.*. User login him self or if he don't have an account he can created it. Follows *Loading Dialog*. When data are downloaded appeare *Project List Page*. User can create new Project or chose from some of existing. If is project select appeares *Item List Page* belongs to project. Each Item has chat section and information section. Item detail in *Item Page*. From *Item Page* is possible take a photo and add this photo to Item. *Start II.* is application start if user is already logged but still does not select Project. Once user select Project the application start with *Start III.* action.

6.2 Communication

It is used HTTP protocol POST method for the communication with the server. The Body of request has *application/x-www-form-urlencoded* content type with three parameters *cmd*, *token* and *input*. The client has to ask server to generate new token at first. The Token is used as identifier in communication with the server. The Token is unique for different users and different devices. If one user uses more devices then each device has allocated different token. Asking token is the first message to all clients. The Client can send new data and request old data from the server when has token as show Figure 6.2. All communication has user context.

A Prameter cmd contains string with command name. The Prameter token specifies user *token* and *input* contain JSON string with parameters for command. Each request has to include this three parameters except command *get_token*. This command makes a request server to generate new token for the user. Communication format is shown on Figure 6.4. Server return JSON contetn with parameters *cmd*, *results* with number of results in response, *data* with data and *error*. Parrameter *error* content *message* where is specify type of error in string and *id* where is error code. List of commands and JSON *input* parameters for request and JSON HTTP BODY response in *data* parameter is shown in Table 6.1 and Table 6.2. The bolt cell is name of *input* parameter. The errors generated by databse are specify in Table 6.3.

6.3 Database

Database is similar to database on server but not same. In Figure 6.5 is databse diagram. Each table corresponds with *Data Classes* from Section 6.1. Table *User* is main table which has relationship with almost all tables. Tables has `Foreign keys` for linking rows.

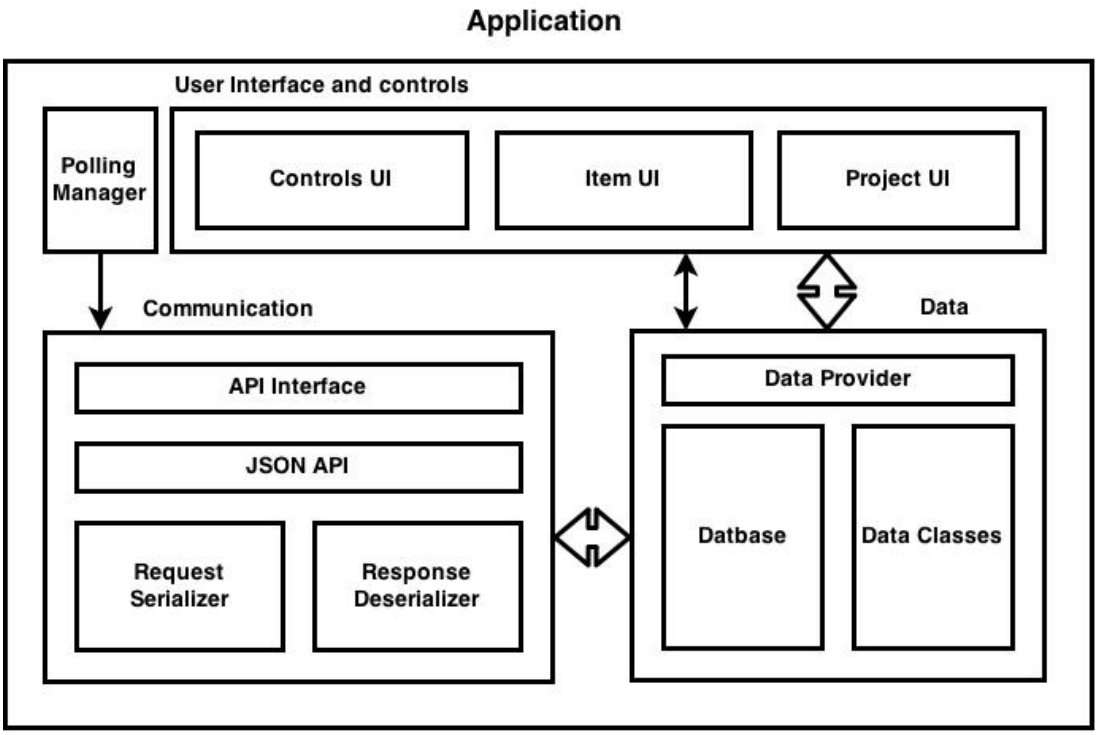


Figure 6.1: The application block model

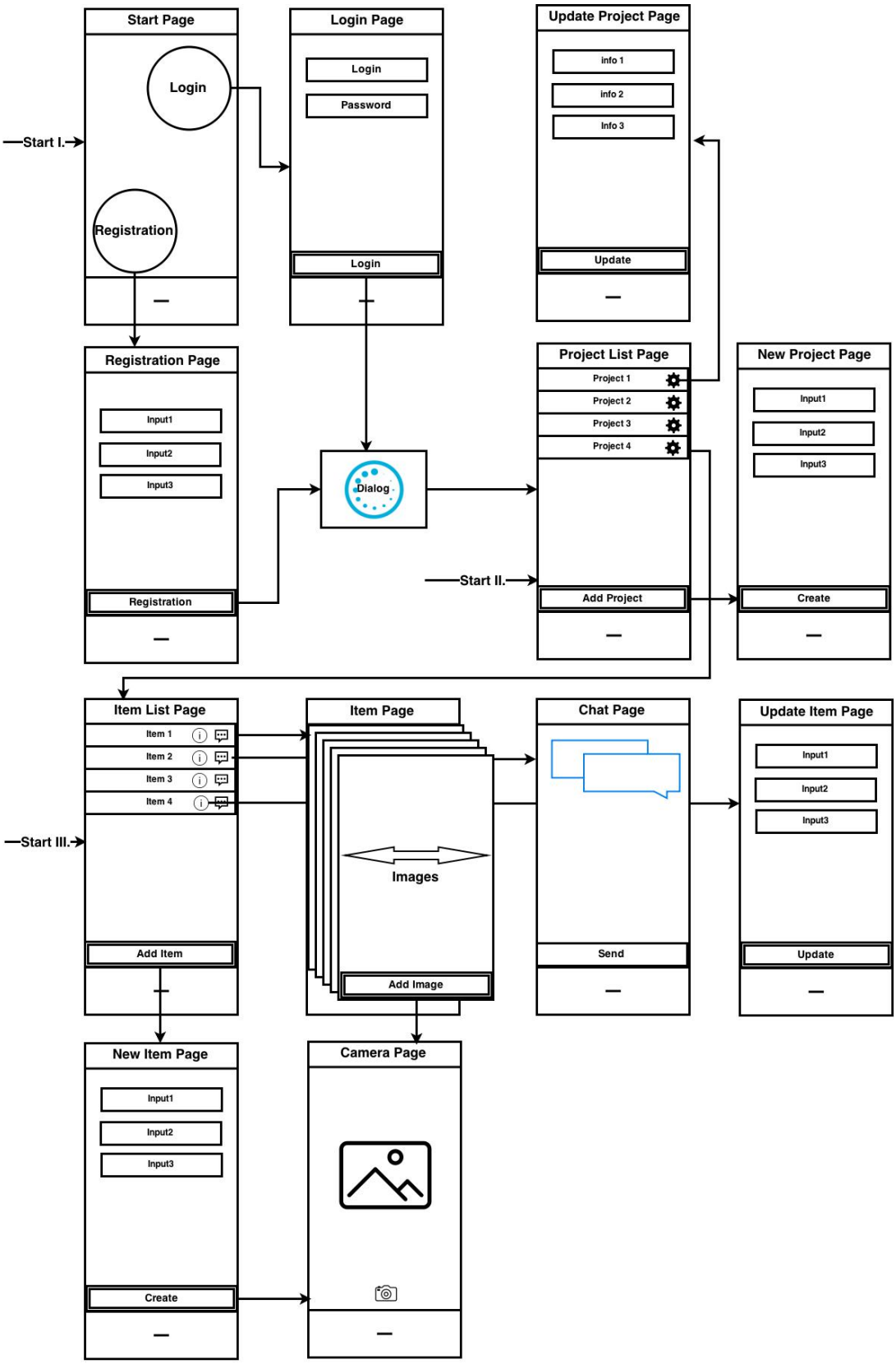


Figure 6.2: The application storyboard

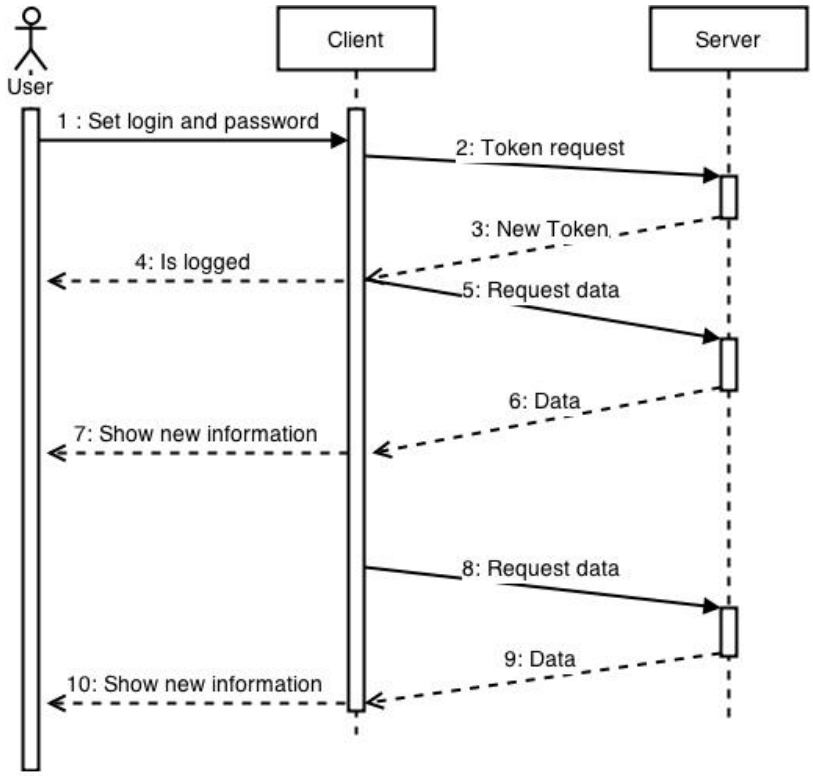


Figure 6.3: Sequence diagram of first communication

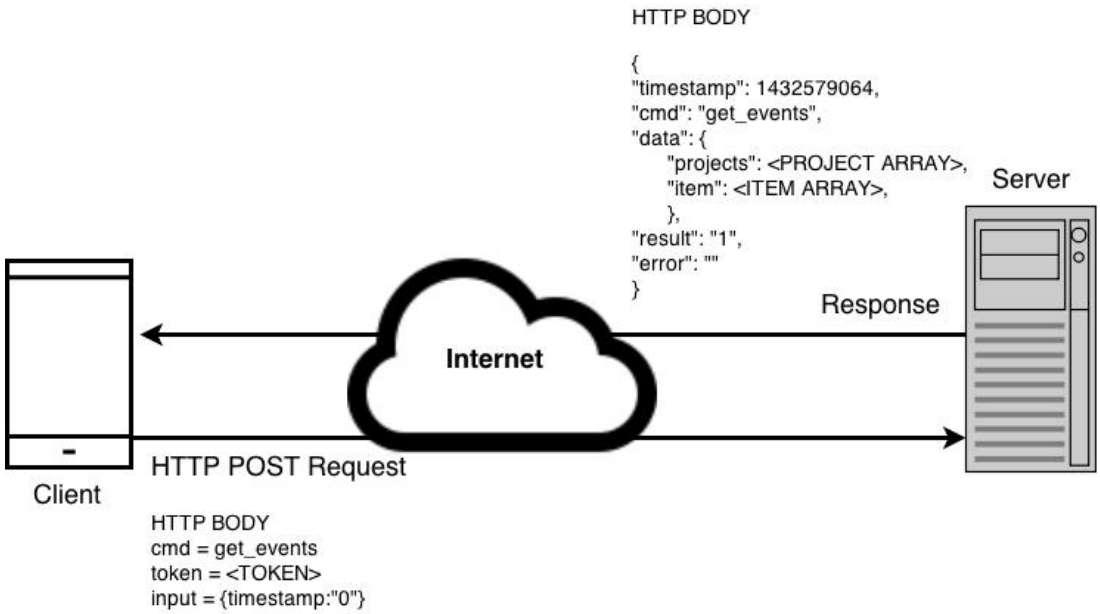


Figure 6.4: Communication example

get_token command	
request:	cid - device indentifier login - user identifier (aid or login name) password type - 0 (loging by aid) / 1 (loging by login name)
response:	token

add_project command	
request:	name - project nameaccount_id - account id which for is project connect address - project address (not required) description - project descriotion (not required) finish_time - timestamp of finish time (not required)
response:	pid - new project id

add_sheet command	
request:	pid - project id which for belong name - sheet name (not required) POST.file - binary data (image or PDF)
response:	sheet_id

add_item command	
request:	pid - project id which for belong name - item name (not required) sheet_id - sheet id (not required)
response:	iid - item id changeTime - timestamp of last change

update_item command - user has to be the item owner	
request:	name - item new name (not required) sheet_id - sheet id (not required)
response:	iid changeTime - timestamp of last change

add_multimedia command	
request:	iid - item id which belong POST.file - binary data (image or pdf)
response:	mid - multimedia id

get_multimedia command - return image of multimedia or sheet	
request:	mid - multimedia id or sheet_id - sheed id
response:	binary data

Table 6.1: Communication parameters 1/2

get_preview command - return image preview (256x256px) multimedia or sheet	
request:	mid - multimedia id or sheet_id - sheed id
response:	binary data

get_events command	
request:	timestamp - when was the data created (not required)
response:	users : array of users which client needs projects : array of projects for current user items : array of items for current user multimedia : array of multimedias info for current user contacts : array of contacts for current user project_invitations : array of project invitations from others users (aid, pid, state) item_invitations : array of item invitation from others users (aid, iid, state) accounts : array of available accounts for current user sheets : array of sheets for current user count: response counter (the next response on this command have to be one greater) messages: array of messages

add_message command - add new message	
request:	text - message body iid - item id which message belong
response:	creationTime - timestamp when message hit server msg_id - message id

Table 6.2: Communication parameters 2/2

error code	**error message**	**error description**
666	invalid token	token is invalid or is not specify
2	missing command	command is not specify
789	params error	to many parameters or invalid parameter
1	user does not match	both login and password does not match with any user in database on server
6	unknown command	command is invalide

Table 6.3: Server errors codes and messages

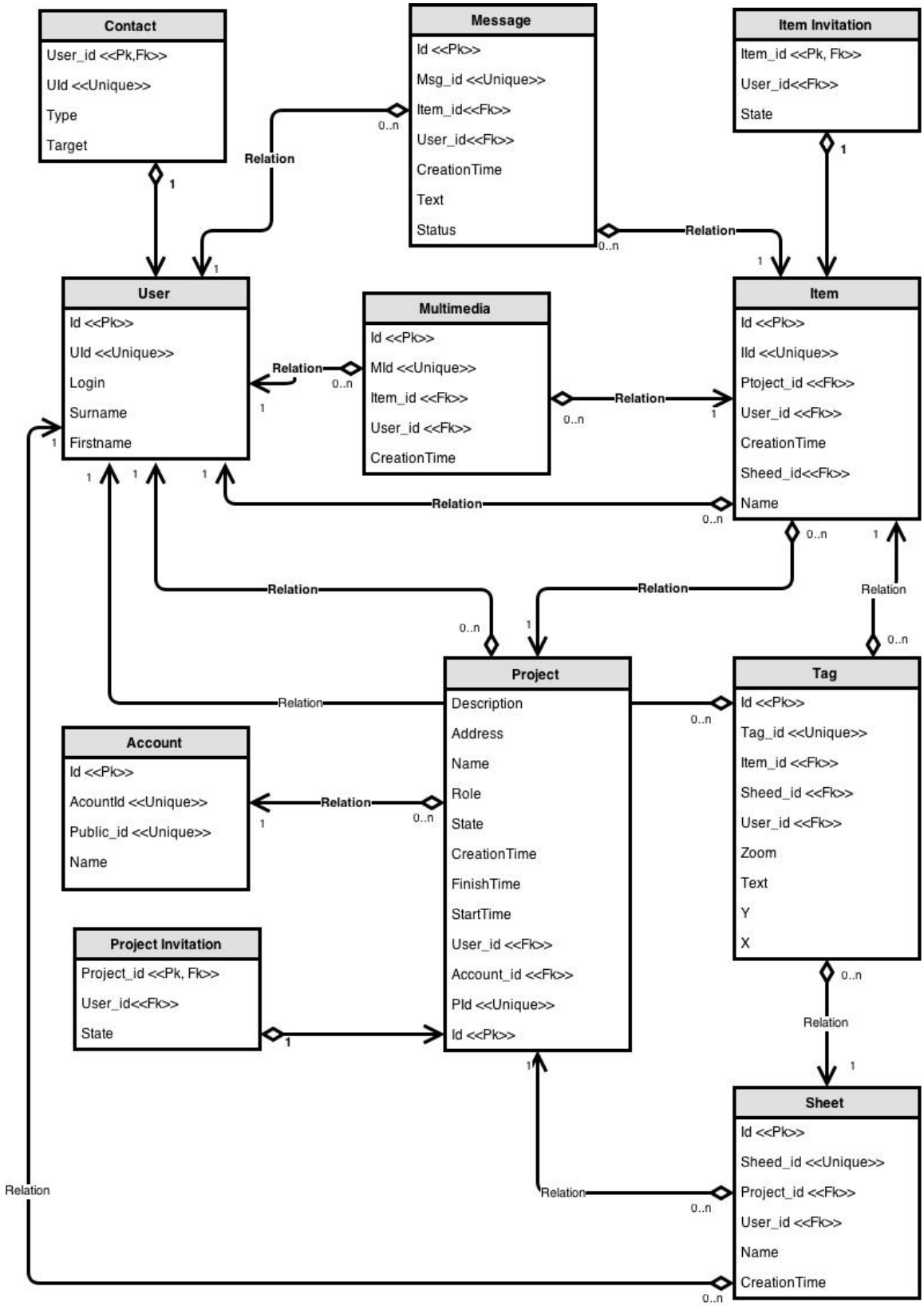


Figure 6.5: Database diagram

Chapter 7

Implementatio

Implementation is using models and diagrams from Chapter 6. Application is implemented in Microsoft Visual Studio 2013 and using his tools. The libraries and notworks are linked to project by NuGet Package Manager. Code is written in C#.

7.1 Database

Diagram form Figure 6.5 is giving model how database should be implemet. In the project was use `SQLite.Net` library for creationg tables and fetching data from database. Communication with database is basically synchronous and `SQLite.Net` use ORM. Calling fuction `conn.Find<Item>(item => item.Iid == message.Iid)` return object of Item classe who has reletionship with object `message`. For defined table is necessary define class with special keywords above attributes. Keywords like `PrimaryKey, AutoIncrement, ForeignKey, Unique`. All tables has defined relationships with other tables. Relationship is defined by `SQLiteNetExtensions` library. This relationship is specified by keywords `ManyToOne, OneToMany, OneToOne`. Each relationships needs `ForeignKey` for addressing connection with other tables.

Tables on client are different than on the server. Tables on client are extended on its own `Id`. User can create new Item but his Id is created by server. After client send data on server is created Item `Id` and return it in response. This action is asynchronous on application run. Client have to create own id for storing Item as soon as is Item created and user can take a photo attached to Item. Client can add photos to Item without storing Item ion server. Client shoul send new Item on server as soon as possible. `DataHelper` class is implement as singleton and handles all action with database.

Example of class corresponding to the table in the database:

```
public class Item
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }
    public int Iid { get; set; } //Item id
    [ForeignKey(typeof(Project))]
    public int ProjectId { get; set; }
    [ForeignKey(typeof(User))]
    public int AId { get; set; } //User owner id
```

```
    public uint CreationTime { get; set; }
    public int Sheet_id { get; set; }
    public string Name { get; set; }

    [ManyToOne]
    public Project Project { get; set; }
    [ManyToOne]
    public User User { get; set; }
    [OneToMany(CascadeOperations = CascadeOperation.All)]
    public List<Message> Messages { get; set; }
    [OneToMany(CascadeOperations = CascadeOperation.All)]
    public List<Multimedia> Images { get; set; }
```

7.2 Communication

For communication has been use HTTP protocol with custom API. Api is define by `APIInterface` interface. The interface is part of *Communication* block. The interface is implemented in `JSONApi` class. Other bocks communicate with *Communication* block by this interface. The `JSONApi` class includes *serializer* and *deserializer*. Processed responses are passing to other blocks by callbacks because the communication is asynchronous.

Request message has `application/x-www-form-urlencoded` content type. Application used for serializing data `MultipartFormDataContent` class. This class has key/value list and value can be `string`, `number` or `biteArray` data type. The `biteArray` has been use for sending images and sheets. The `JsonObject` class is serializer for JSON data wich are append to the `input` parameter.The `MultipartFormDataContent` and `JsonObject` are part of `RequestContent` class which processes all data before sending. The `RequestContent` class corresponds with the *Request Serializer* block from Figure 6.1. Communication is asasynchronous and `MultipartFormDataContent` and is ensuring by `HttpClient` class. The result is passed to `ProcessResponse` class wich is implement as singleton. The `ProcessResponse` deserialize the response send them to database. The `ProcessResponse` corresponds with the *Response Deserializer* block from Figure 6.1.

API interface:

```
interface APIInterface
{
    Task getToken(Action<JsonObject> responseAction,
                        string login, string password, int type);
    Task getEvents(Action<JsonObject> responseAction, string timestamp);
    Task getEvents(Action<JsonObject> responseAction, DateTime startDate);
    Task getEvents(Action<JsonObject> responseAction);
    Task getNewNotification(Action<JsonObject> responseAction);
    Task getMultimediaPreview(Action<byte[]> responseAction, int mid);
    Task getSheetPreview(Action<byte[]> responseAction, int sheet_id);
    Task<byte[]> getMultimedia( int mid);
    Task getSheet(Action<byte[]> responseAction, int sheet_id);
```

```
    Task addSheet(Action<JsonObject> responseAction);
    Task addMultimedia(Action<JsonObject> responseAction,
                       StorageFile image, int iid);
    Task addItem(Action<JsonObject> responseAction, Item item);
    Task addProject(Action<JsonObject> responseAction, Project project);

}
```

7.3 User Interface

The User Interface is implemeted in XAML and controlled by C# controllers classes. For data binding is used *observers* wich notifi interface about data the change of data. For implementation User Interface has been used strandarts interface components. The list is `ListView` component and the Item detail is created by `StackPanel`. On Figure 7.1, Figure 7.2 and Figure 7.3 are displayed application pages. On Figure 7.4 is comparison of design on the Android and on the Windows Phone.

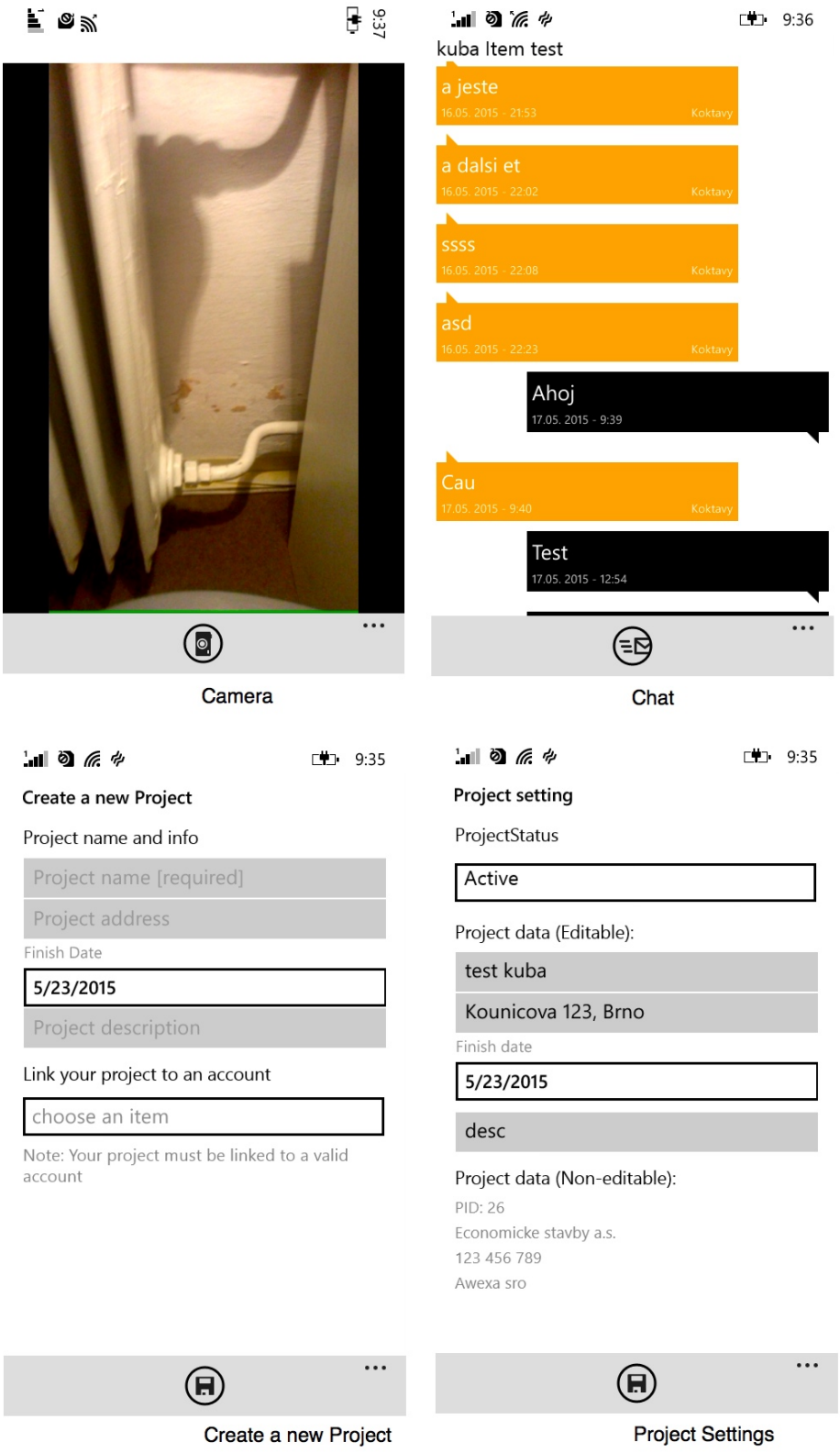


Figure 7.1: Application pages 1/3

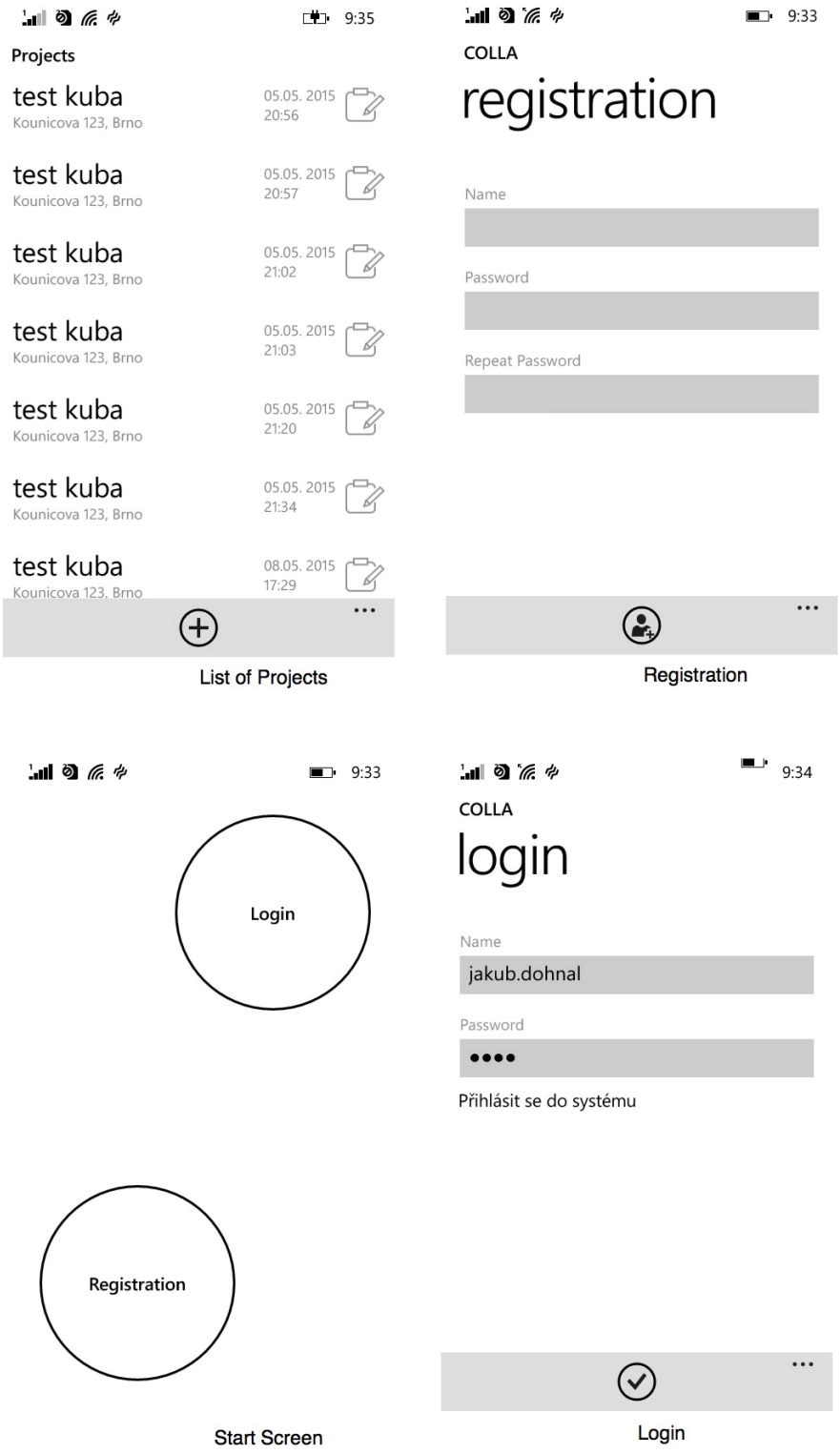


Figure 7.2: Application pages 2/3

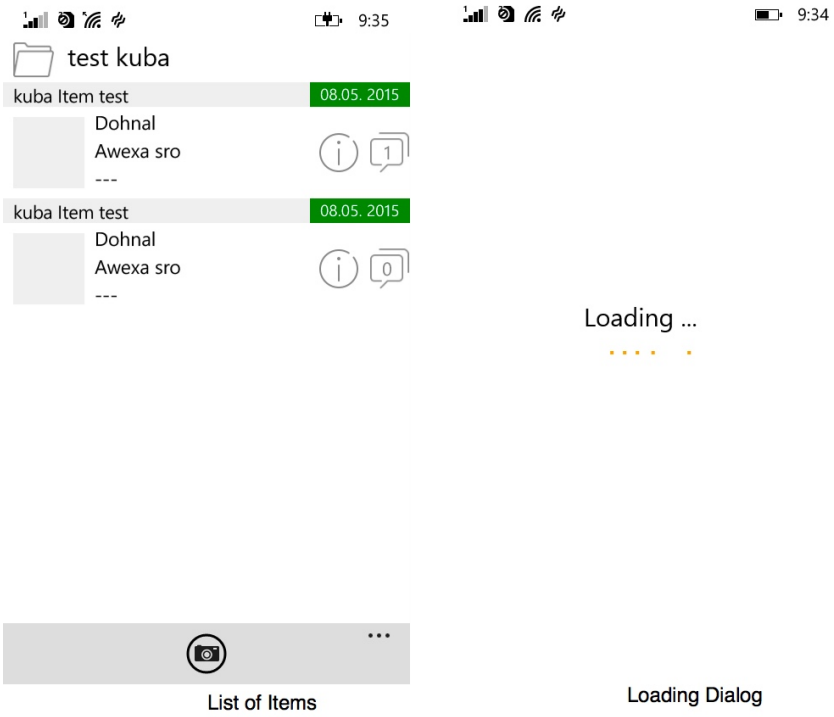


Figure 7.3: Application pages 3/3

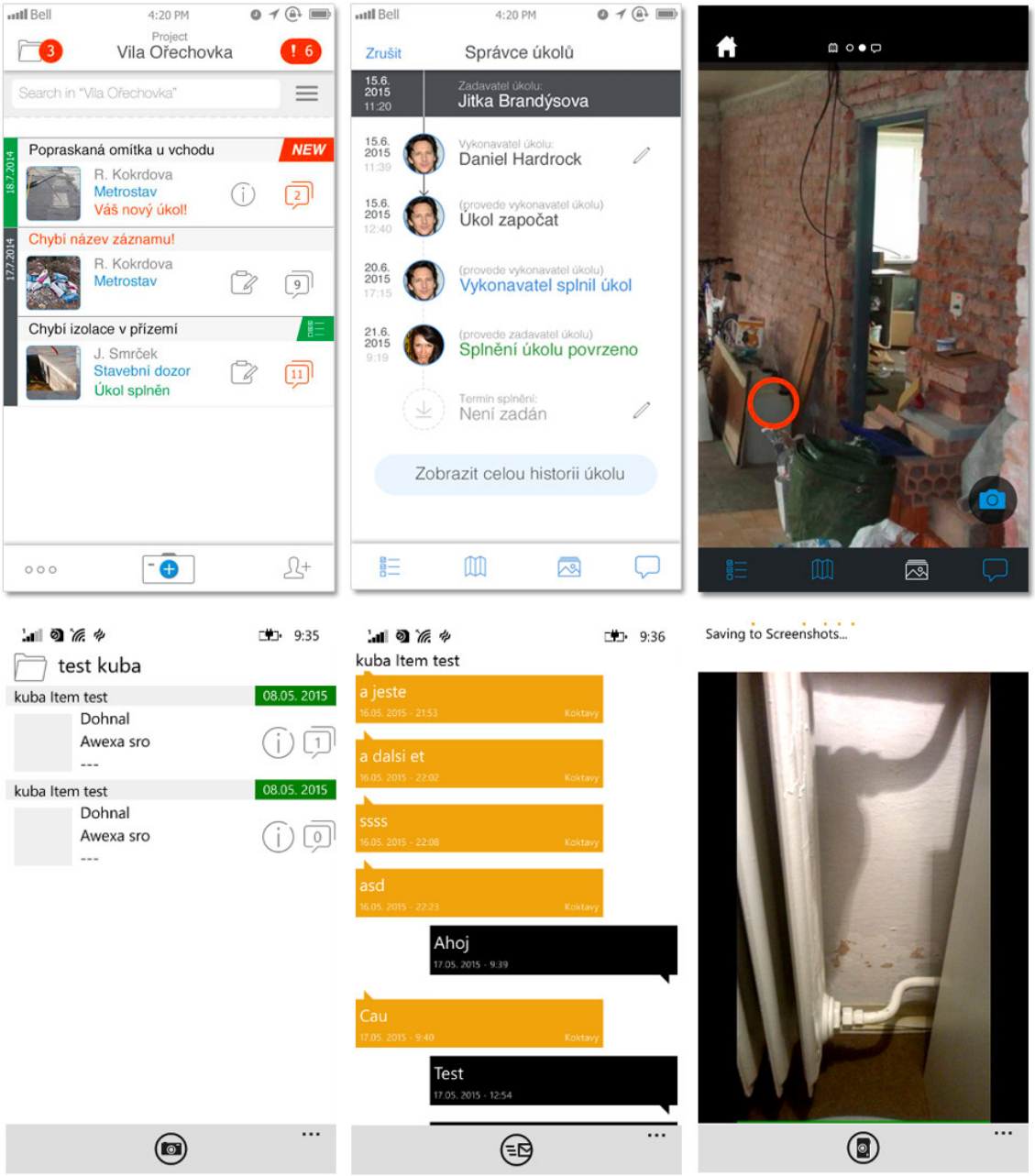


Figure 7.4: The application design comparison with Android platform

Chapter 8

Testing

For debugging and testing has been used standard Visual Studio tools as *Application Verifier*, *Record Performance* for records memory leaks and *Isolated Storage Explorer* for debugging database and stored images.

User experience testing is defined as questionnaire for user. Users are divided into two categories. The first category are users who working as construction supervision who are mainly entering information to the application and assign defects to individual buildings. The second category are users who works as construction manager and observe new information in the application and and with its help manage repair defects. This two different groups have different questionnaire with questions of how they use the application, what are their problems with the application, how often they use the application and What features are missing in the application. This test also requires go with users to work and observe their behavior and if they use the application properly. This process is time consuming and requires a long observation. It has been scheduled two months of testing with users.

Chapter 9

Conclusion

This master's thesis is focused on how to design client-server application and design what procedures and technologies are used for implementation of this type of application on Windows Phone platform. The thesis indicates a problems with pooling method and describes the design of communication between application and server. The available protocols for communication are assessed, including their appropriateness for using in final application.

The proposed application utilizes the local storage and database system for using application without internet connection and shows how to keep the data consistency. The application allow users shared their pictures and maintain their connections. The application allows user to take a photo of defect and pass it to other coworkers. Also allows you to edit individual records and opening and closing defects in the form of items. The application mediates the discussion of individual faults between coworkers in a chat.

It is important to test the application with users. It is the most important work that must be planned. Contact the user to test the application and control the further development of the applications according to their requirement. As required to add more features to the application and finish the application appearance to resemble the appearance of the application on other platforms. Arrange partners for the delivery of applications and monetize applications according to their wishes.

Bibliography

[1] A. Berson. *CLIENT/SERVER ARCHITECTURE. 2nd edition, édition en anglais.* McGraw-Hill, 1996. ISBN 9780070056640.

[2] Clive Evans, David Lacey, David Harvey, David Gibbons and Andy Krasun. *CLIENT/SERVER: A Handbook of Modern Computer Design.* Prentice Hall International UK, 1995. ISBN 0-13-377201-2.

[3] Dave Winer. Xml-rpc specification. `http://xmlrpc.scripting.com/spec`, 1999-06-15 [cit. 2015-04-15].

[4] Jani Ilkka Frederick Hirsch, John Kemp. *Mobile Web Services: Architecture and Implementation.* John Wiley & Sons, 2007. ISBN 047-001-596-9.

[5] Mukesh Singhal Gurdeep S. Hura. *Data and computer communications : networking and internetworking.* CRC Press, 2001. ISBN 0-8493-0928-1.

[6] Leonard Richardson, Sam Ruby. *RESTful Web Services.* O'Reilly Media, 2008. ISBN 978-0-596-52926-0.

[7] PD Dr. Ralf Klamma, AOR. The xmpp experience. `http://dbis.rwth-aachen.de/cms/projects/the-xmpp-experience`.

[8] Kevin Smith Peter Saint-Andre and Remko Troncon. *XMPP: The Definitive Guide.* O'Reilly Media, 2009. ISBN 978-0-596-52126-4.

[9] WWW site. Smart client-side caching. `https://www.componentone.com/Studio/Data-Management/DataSourceSilverlight`.

[10] WWW site. Overview of saaj. `https://docs.oracle.com/javaee/5/tutorial/doc/bnbhg.html`, 2010 [cit. 2015-4-10].

[11] WWW site. The websocket protocol. `http://tools.ietf.org/html/rfc6455/`, 2011 [cit. 2015-4-10].

[12] WWW site. How to build a restful web api on a raspberry pi in javascript. `https://thefloppydisk.wordpress.com/2013/05/08/how-to-build-a-restful-web-api-on-a-raspberry-pi-in-javascript/`, 2013 [cit. 2015-4-10].

[13] WWW site. Xaml for windows phone 8. `https://msdn.microsoft.com/en-us/library/windows/apps/cc189036.aspx`, 2015 [cit. 2015-04-10].

[14] WWW site. Building windows and windows phone apps with shared code. https://dev.windows.com/en-US/develop/build-apps-shared-code, 2015 [cit. 2015-4-10].

[15] WWW site. Push notifications for windows phone 8. https://msdn.microsoft.com/en-us/library/windows/apps/ff402558%28v=vs.105%29.aspx, 2015 [cit. 2015-4-10].

[16] WWW site. Quickstart: Adding winjs controls and styles (html). https://msdn.microsoft.com/en-us/library/windows/apps/hh465493.aspx, 2015 [cit. 2015-4-10].

[17] WWW site. Xmpp technologies overview. http://xmpp.org/about-xmpp/technology-overview/, 2015 [cit. 2015-4-10].

[18] WWW site. Languages, tools and frameworks. https://msdn.microsoft.com/en-us/library/windows/apps/dn465799.aspx, 2015 [cit. 2015-4-15].

Appendix A

CD content