



## ASSIGNMENT OF BACHELOR'S THESIS

**Title:** Qt Desktop Google Contacts  
**Student:** Taras Petrychkovych  
**Supervisor:** Ing. Robert Pergl, Ph.D.  
**Study Programme:** Informatics  
**Study Branch:** Software Engineering  
**Department:** Department of Software Engineering  
**Validity:** Until the end of summer semester 2016/17

### Instructions

Perform analysis, design, and implementation of a desktop GUI application using the Qt library. It should be able to manage Google Contacts using its API. The required features are:

- synchronisation of all contact information,
- fields editing,
- command line operation variant for finding, adding and editing,
- seamless offline mode with synchronisation when online,
- incremental search, including substrings,
- export and import functionality in a suitable format,
- lightweight, low-resources, fast app,
- effective UI, configurable keyboard shortcuts for all actions,
- reliability.

Document your solution and tests.

### References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.  
Head of Department

prof. Ing. Pavel Tvrdík, CSc.  
Dean

Prague February 4, 2016



CZECH TECHNICAL UNIVERSITY IN PRAGUE  
FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF SOFTWARE ENGINEERING



Bachelor's thesis

## Qt Desktop Google Contacts

*Taras Petrychkovych*

Supervisor: Ing. Robert Pergl, Ph.D.

17th May 2016



---

## **Acknowledgements**

I would like to thank my supervisor for valuable advices and support during the development process and writing the thesis text. I also would like to thank my friends Firuz Ibragimov and Alzhan Turlybekov for their support and help in testing the program.



---

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 17th May 2016

.....

Czech Technical University in Prague

Faculty of Information Technology

© 2016 Taras Petrychkovych. All rights reserved.

*This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

### **Citation of this thesis**

Petrychkovych, Taras. *Qt Desktop Google Contacts*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.



---

# Abstrakt

Cílem této bakalářské práce je navrhnout, implementovat a otestovat multiplatformní program pro operační systémy Windows a Linux. Daný program, prostřednictvím Google Apps servisu, umožňuje uživateli pracovat s Google Contacts v on-line/off-line režimu a také podporuje synchronizaci s Google databází kontaktů uživatele. Aplikace je vytvořena pomocí multiplatformní knihovny Qt a je napsána v jazyce C++.

**Klíčová slova** Google Contacts, Google Apps, Qt, C++, QxOrm, OAuth 2.0.

---

# Abstract

The aim of this bachelor's thesis is to design, implement and test cross-platform application for Windows and Linux operating systems. The program, by using the Google Apps service, allows users to work with the "Google Contacts" in online/offline mode and also supports synchronization with user's Google Contacts database. The application was developed by using a cross-platform Qt library and was written in C++.

**Keywords** Google Contacts, Google Apps, Qt, C++, QxOrm, OAuth 2.0.



---

# Contents

<b>Introduction</b>	<b>1</b>
<b>1 Goal and methodology</b>	<b>3</b>
<b>2 Review</b>	<b>7</b>
2.1 Google Apps . . . . .	7
2.2 OAuth 2.0 . . . . .	9
2.2.1 How does the OAuth 2.0 work? . . . . .	9
2.2.2 Pros . . . . .	10
2.2.3 Cons . . . . .	10
2.2.4 Summary . . . . .	10
2.3 Qt . . . . .	10
2.4 QxOrm . . . . .	12
2.5 Boost . . . . .	13
2.6 Qt Creator . . . . .	13
<b>3 Analysis</b>	<b>15</b>
3.1 Requirements specification . . . . .	15
3.1.1 Functional requirements . . . . .	15
3.1.2 Non-functional requirements . . . . .	16
3.2 Domain Model . . . . .	16
3.3 Application processes . . . . .	16
3.4 Use cases . . . . .	18
<b>4 Design</b>	<b>23</b>
4.1 Wireframes . . . . .	23
4.2 Database . . . . .	23
<b>5 Implementation</b>	<b>27</b>
5.1 Application architecture . . . . .	27

5.1.1	Application layer . . . . .	27
5.1.2	Data layer . . . . .	28
5.1.3	Presentation layer . . . . .	29
5.1.4	Realization of application logic . . . . .	30
5.1.4.1	Synchronization of contacts . . . . .	31
5.1.5	Command line version of the application . . . . .	33
<b>6</b>	<b>Testing</b>	<b>35</b>
6.1	Unit tests . . . . .	35
6.2	Manual testing on different operation systems . . . . .	37
<b>7</b>	<b>Adaptation for other platforms</b>	<b>39</b>
<b>8</b>	<b>Features</b>	<b>41</b>
	<b>Conclusion</b>	<b>43</b>
	<b>Bibliography</b>	<b>45</b>
<b>A</b>	<b>Acronyms</b>	<b>47</b>
<b>B</b>	<b>Contents of enclosed CD</b>	<b>49</b>

---

## List of Figures

1.1	Google Contacts old view . . . . .	3
1.2	Google Contacts preview . . . . .	4
1.3	GooBook . . . . .	5
2.1	Using OAuth 2.0 for Installed Applications . . . . .	11
3.1	Domain Model . . . . .	17
3.2	Search contacts application process . . . . .	18
3.3	Login process in application . . . . .	19
3.4	Create contact application process . . . . .	20
3.5	Synchronize use case . . . . .	20
3.6	Contact use case . . . . .	21
4.1	Main window wireframe . . . . .	24
4.2	Add new contact wireframe . . . . .	25
4.3	Database model . . . . .	26
5.1	Application overview diagram . . . . .	28
5.2	Main window of Application . . . . .	30
5.3	Edit contact dialog . . . . .	31
5.4	Google Contacts options dialog . . . . .	32
6.1	Assertion failed . . . . .	35
6.2	Unit tests . . . . .	36



---

# Introduction

What is Google Contacts you may ask? Google Contacts is Google's contact management tool that is available in its free email service Gmail, as a standalone service, and as a part of Google's business-oriented suite of web apps Google Apps [1]. So in other words your contacts from Gmail, Google+ circles, your mobile phone and other Google's services are stored and synchronized in one database and they all are called Google Contacts.

The main purpose is to create reliable desktop application with effective and similar to Contacts preview user interface and possibilities of adding, editing and searching contacts. Application also should allow user work with contacts in offline mode with synchronization when online, import and export contacts to XML or CSV formats and support command line operation variant for adding, editing or searching contacts. For more convenience it should also support keyboard shortcuts for all these actions.





## Goal and methodology

Nowadays for working with Google Contacts you might use standard Google Contacts view web page <https://www.google.com/contacts/#contacts> (see Figure 1.1) or the new one <https://contacts.google.com/u/0/preview/all> (see Figure 1.2) named Contacts preview.

Next thing that can help you to manage your Google Contacts is Thunderbird add-on named "Google Contacts". This extension accesses to Google contacts and synchronize them with Thunderbird address books. After you installed the extension, it detects all the Gmail accounts which have already set up on Thunderbird and accesses to the contacts for them. When you update cards in the address books, the changes are immediately applied to the contacts in Google; if you delete a card, the contact will be remove from

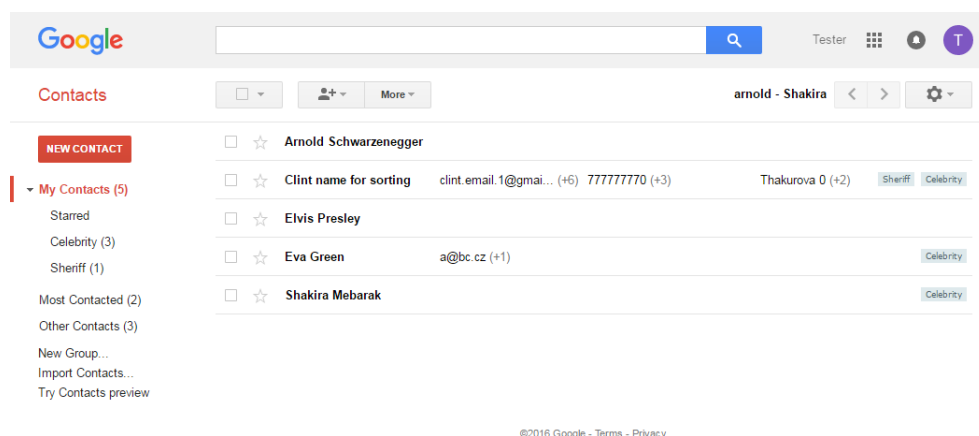


Figure 1.1: Google Contacts standard view

## 1. GOAL AND METHODOLOGY

---

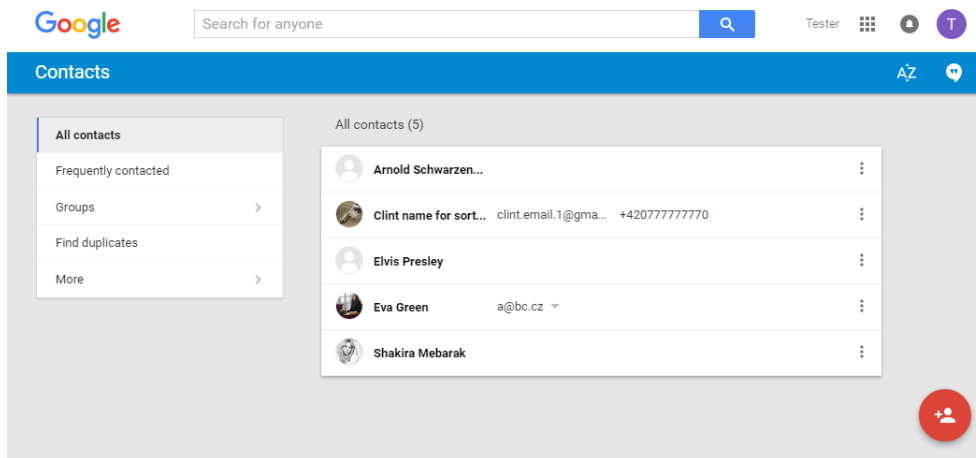


Figure 1.2: Google Contacts preview

Google; if you add a new card to the address book, a contact is added to Google. If you delete the address book, the contacts for the account are no more synchronized [2]. But unfortunately it is not managed by Google alone it was developed by author "H.Ogi" which warns you that it may involve serious bugs and since last updated was in June 20, 2013 it is not supported well anymore and what is worse it is not compatible with Thunderbird v38.1.0 or more and Google new security policies (i.e., OAuth 1.0 stopped being supported).

One more useful way to work with your Google Contacts, especially if you are a big fan of the command-line, is "GooBook" which purpose is to make it possible to use your Google Contacts from the command-line and from MUAs<sup>1</sup> such as Mutt<sup>2</sup> (see Figure 1.3). It is very primitive, but you can manage Google Contacts without the need for a browser as an intermediary. Which is always a good thing. Created by "Christer Sjöholm" who still holds it up and adds new features.

So the goal is to create desktop application using Qt libraries. It should combine existing ways of using Google Contacts and make Google Contacts managing more comfortable. The required features are:

- Synchronisation of all contact information.
- Editing Google Contacts.
- Command line operation variant for finding, adding and editing Google Contacts.

---

<sup>1</sup>MUAs (Mail User Agent) — is a computer program in the category of groupware environments used to access and manage a user's email.

<sup>2</sup>Mutt (The Mutt E-Mail Client) — is a small but very powerful text-based mail client for Unix operating systems.

A terminal window titled 'urxvt' with standard window controls. The terminal shows the following text:

```
06:34:24 petrytar@6m47421: ~$ goobook -v reload
INFO:goobook.config:Reading config: /home/petrytar/.goobookrc
INFO:goobook.goobook:Retrieving contact data from Google.

06:34:32 petrytar@6m47421: ~$ goobook query petrytar

petrytar@gmail.com      petrytar      other groups: "Examples"

06:34:37 petrytar@6m47421: ~$ █
```

Figure 1.3: GooBook

- Seamless offline mode with synchronisation when online.
- Incremental search, including substrings.
- Export and import functionality in a suitable format.
- Effective user interface, configurable keyboard shortcuts for all actions.

Creating this application is based on using Google Contacts API and OAuth 2.0 authorization to communicate with Google Contacts, Qt libraries to create program with GUI, QxOrm to store data into database and Boost. The program must be written using Qt Creator.



---

# Review

## 2.1 Google Apps

API stands for Application Programming Interface. Basically it is doorway through which only people with the right key can pass. For example this application needs to get Google Contacts data, so it appeals to Google's API and the API lets it to come in and checks to make sure that it have a right key if it don't the API kick application back, but if this application have a right key it will resend the request of application to Google Contacts and then will send back the reply in the way that application can understand it. Application can use key just for reading or it can use key for reading and modifying Google Contacts.

Because user's contacts are mostly private, the application can access them only by using an authenticated request. That is, the request must contain an authentication token for the user whose contacts application wants to retrieve [3]. The application uses OAuth 2.0 protocol with authentication option for installed applications to get the token.

Steps to get authentication token:

1. Redirect a browser to a Google URL. The URL query parameters indicate the type of Google API access that the application requires.
2. Google handles user authentication and consent, upon success it responds with authorization code as a query string parameter to the local web server.
3. The application creates POST request with the authorization code, its client ID and client secret (obtained from the Google Developers Console) and send it to Google API to obtain access and refresh tokens.
4. Application uses the access token to make calls to a Google API and stores the refresh token for future use.

## 2. REVIEW

---

To see these steps as overview diagram, (see Figure 2.1)

But before application can receive the token it needs to be register as a client on the Developers Console or in other words Google APIs Console. Once application is registered, it will cover all users. Each user must authorize the application to access their individual data.

Functionality of Google Contacts API with contacts:

- To retrieve contacts application send an authorized GET request to the URL<sup>3</sup> with special parameter which depends on the required data:
  - Retrieving all contacts
    - \* parameter is set to "null";
    - \* upon success, the server responds with a HTTP 200 OK status code and the requested contacts feed.
  - Retrieving contacts using query parameters
    - \* parameter is set to "updated-min=2007-03-16T00:00:00";
    - \* the server returns an HTTP 200 OK status code and a feed containing any contacts that were created or updated after the date specified.
  - Retrieving a single contact
    - \* parameter is set to "contactId";
    - \* upon success, the server responds with an HTTP 200 OK status code and the requested contact entry.
- To create a new contact the application send an authorized POST request to the user's contacts feed URL with contact data in the body. Upon success, the server responds with an HTTP 201 Created status code and the created contact entry with some additional elements and properties (shown in bold) that are set by the server, such as id, various link elements and properties.
- To update a contact the application first retrieve the contact entry, modify the data and then send an authorized PUT request to the contact's edit URL with the modified contact entry in the body.
  - parameter is set to "contactId";
  - upon success, the server responds with an HTTP 200 OK status code and the updated contact entry.
- To delete a contact the application send an authorized DELETE request to the contact's edit URL.

---

<sup>3</sup>[https://www.google.com/m8/feeds/contacts/default/full?\[parameter\]](https://www.google.com/m8/feeds/contacts/default/full?[parameter])

- parameter is set to "contactId";
- upon success, the server responds with an HTTP 200 OK status code.

Managing photos and groups is similar to working with contacts [3].

## 2.2 OAuth 2.0

OAuth is a HTTP-based security protocol that allows you, the User, to grant access to your private resources on one site (which is called the Service Provider), to another site (called Consumer, not to be confused with you, the User). While OpenID is all about using a single identity to sign into many sites, OAuth is about giving access to your stuff without sharing your identity at all (or its secret parts) [4]. Instead of unsafe password-sharing, OAuth offers a much more secure delegation protocol. OAuth 2 is the must-know security protocol on the web today and it's the worldwide standard.

### 2.2.1 How does the OAuth 2.0 work?

Like the first version, OAuth 2.0 is based on the basic web technologies: HTTP-requests, redirects, etc. Therefore, the usage of OAuth is possible on any platform with access to the Internet and browser: on websites, mobile and desktop-applications, browser plug-ins ...

The major difference from the OAuth 1.0 is simplicity. The new version has no large circuits signature, reduced the number of requests required for the authorization.

The general scheme of the application that uses the OAuth, is as follows:

- obtaining authorization
- request protected resources

The result of authorization is access token - a key (usually just a set of characters), which represents live pass to protected resources. In the simplest case it is appealed over HTTPS set in titles or as one of the access token parameters.

There are several authentication options that are suitable for different situations:

- Web server applications.
- Installed applications.
- Client-side (JavaScript) applications.
- Applications on limited-input devices.

Authentication of this application is based on using authentication option for installed applications. This is the most complex authentication option, but only it allows the server to uniquely identify application that is currently applying for authorization. In all other cases, authorization is totally on the client side, what is fine, because sometime you might need mask one application under another [5].

### 2.2.2 Pros

There are two main advantages of being a trusted client [6]:

- **More security:** — The key is shared only between the service provider and server-side of the client application. It never gets sent to the browser, and so has much less of a chance of being intercepted.
- **Long-term and offline access:** — Because the client is able to securely store information, they can store the keys and properties necessary for long-term, and even offline, access to a user's data.

### 2.2.3 Cons

Unfortunately, there is a disadvantage associated with this [6]:

- **More complexity:** — To achieve the added security features that make this workflow so beneficial, a more complex infrastructure must be in place to facilitate the more complex key exchange that this workflow utilizes.

### 2.2.4 Summary

OAuth - simple authentication standard based on the basic principles of the Internet, which makes possible to use authorization on almost any platform. The standard has the support of major platforms and it is clear that its popularity will only grow. If you are thinking about an API to your service, the authorization using OAuth 2.0 is a good choice.

## 2.3 Qt

Qt (/kju:t/ 'cute') — is a comprehensive C++ application development framework for creating cross-platform GUI applications using a "write once, compile anywhere" approach. Qt lets programmers use a single source tree for applications that will run on Windows 98 to Windows 10, Mac OS X, Linux, Solaris, HP-UX, and many other versions of Unix with X11 [7].

One of the things that makes Qt a pleasure to use is its online documentation. You can find describing of all classes with sample examples online or



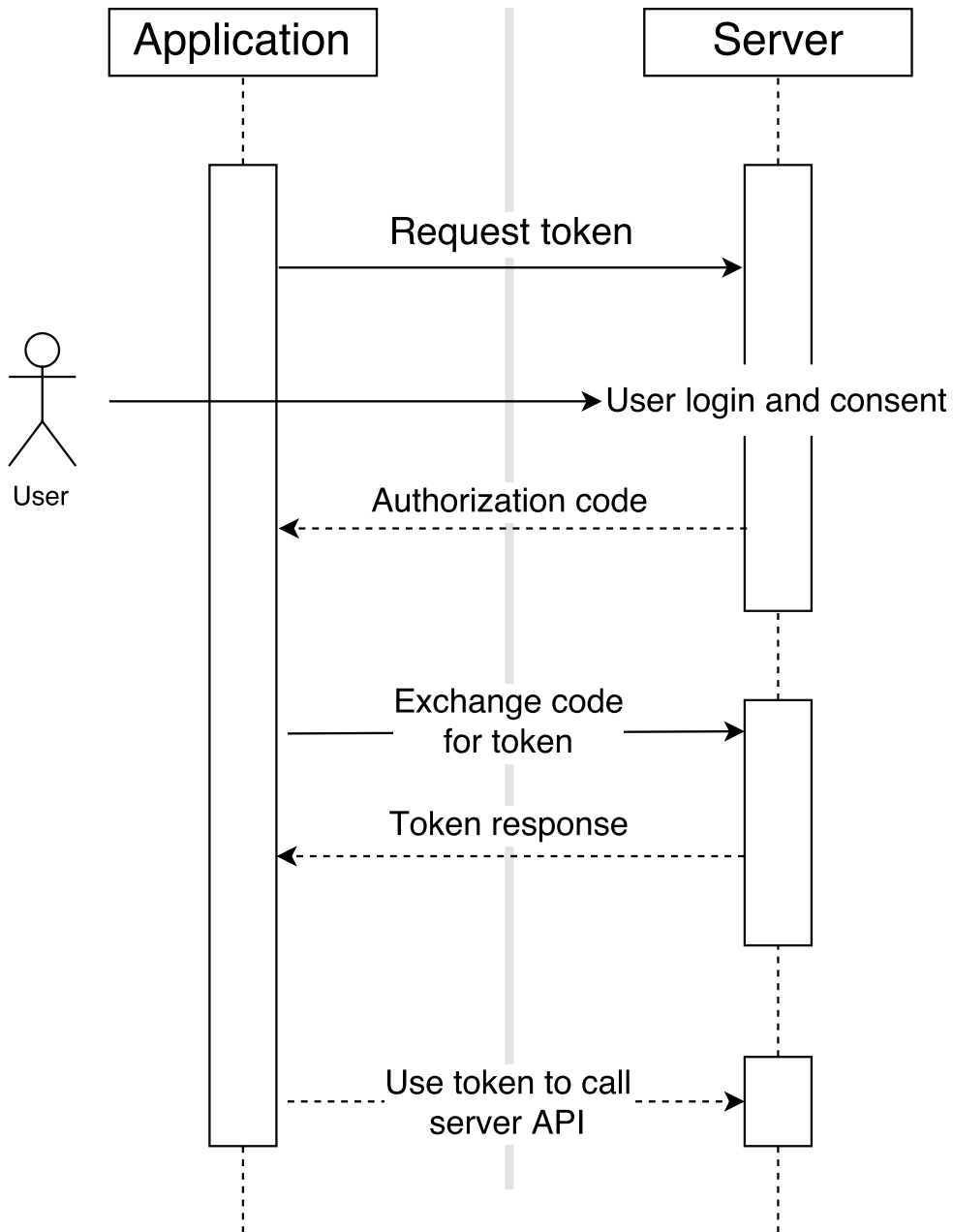


Figure 2.1: Using OAuth 2.0 for Installed Applications

you can download source code and browse it manually. Qt creator also allows you to see all documentation just pressed key F1 on needed class, signal, slot, method etc. The documentation is so good that if you have a basic knowledge of C++, Java, or C# programming languages you can easily start write program without reading any Qt tutorial book.

Next big advantage of Qt is signals and slots system which are used for communication between objects. Qt uses a code generator (the Meta-Object Compiler or moc) to implement flexible signals/slots. Classes can mark themselves as moc'eable with the Q\_OBJECT macro (and must inherit QObject), then indicate that some functions in the class are slots, and some are signals. Slots have declarations and definitions just like normal functions; signals are essentially just a function prototype, and have no definitions (the moc provides them) [8].

### 2.4 QxOrm

QxOrm library is an Object Relational Mapping database library for C++/Qt developers. With a simple C++ setting function per class (like Hibernate XML mapping file in Java), you have access to the following features:

- **persistence** — communication with databases (support 1-1, 1-n, n-1 and n-n relationships)
- **serialization** — binary, XML and JSON format
- **reflection (or introspection)** — access dynamically to classes definitions, retrieve properties and call classes methods

QxOrm library is designed to make easier C++ development and provides many functionality. Advantages of QxOrm library :

- The C++ setting function per class doesn't modify class definition, QxOrm can be used in existing projects.
- No XML mapping file.
- Classes doesn't need to inherit from a 'super object'.
- Template meta-programming.
- Works with Visual C++ on Windows, GCC on Linux, Clang on Mac OS X, and MinGW on Windows.
- Only one file <QxOrm.h>to include in precompiled-header.

For modelling entities and relations for the QxOrm framework there is a graphical interface named QxEntityEditor. It is multi-platform (available for Windows, Linux and Mac OS X) and generates native code for all environments : desktop (Windows, Linux, Mac OS X), embedded and mobile (Android, iOS, Windows Phone, Raspberry Pi, etc.). QxEntityEditor is based on plugins and provides many ways to import/export your data model :

- generate C++ persistent classes automatically (registered in QxOrm context)
- generate DDL SQL script automatically (database schema) for SQLite, MySQL, PostgreSQL, Oracle and MS SQL Server
- manage schema evolution for each project version (ALTER TABLE, ADD COLUMN, DROP INDEX, etc.)
- transfer your data model over network and create quickly client/server applications, using QxService module
- import existing database structure (using ODBC connection) for SQLite, MySQL, PostgreSQL, Oracle and MS SQL Server databases
- because each project is different, QxEntityEditor provides several ways to customize generated files (especially a JavaScript engine and an integrated debugger)

## 2.5 Boost

Boost is a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing. Boost gives you a lot of the tools you need to compose stuff easily. Smart pointers, functions, lambdas, bindings, etc. It was not necessarily to use Boost except QxOrm which actually depends on it, but it's truly become helpful.

## 2.6 Qt Creator

Why Qt Creator? Well I have a half year experience with working on Qt Creator and since I have known it I truly like it. Qt Creator has great object support (help, navigation, etc.), GUI editor, nice features like mouse navigation, really fast IDE, very great debugger and fantastic possibility of using keyboard shortcuts so it is really simply and comfortable to use. Also Qt Creator is a lot more Qt-oriented. Every common operation while developing with Qt is bound to be easier in Qt Creator since it's primary goal was to be a tool for developing Qt applications.



---

# Analysis

## 3.1 Requirements specification

### 3.1.1 Functional requirements

Functional requirements are the statements of services that the system delivers. These statements describes how the system should react to inputs and how it should behave in particular situations. They also are known as capabilities or features [9].

Google Contacts functional requirements:

- Adding, deleting and editing contacts.
- Synchronization of all user's contacts.
- Substring searching.
- Creating groups and adding contacts to them. The application should have four groups defined by Google: Family, Friends and Coworkers.
- Displaying all data in offline mode.
- Command line versions for all previous described functional requirements.
- Keyboard shortcuts.
- Status bar displaying actual status of synchronization process.
- Importing and exporting contacts from file.
- Validation of user's entered data.

#### 3.1.2 Non-functional requirements

Nonfunctional requirements are sometimes known as restriction or quality requirements. They do not affect the functionality of the system [9].

Application non-functional requirements:

- Intuitive graphical user interface.
- Command line manual.
- Cross-platform application.
- Handle errors during synchronization process.

#### 3.2 Domain Model

The domain model (see Figure 3.1) for the application is represented by class diagram. The purpose of the diagram is to show and explain program structure, staff, relationships with users, and user contacts.

The ContactEntry entity describes Google Contact Entry and contains ID, Google ID, name, nickname, organization name, organization title and flag to mark whether the contact was locally deleted. On the diagram a ContactEntry could has different ContactProperty entities. ContactProperty must have only one ContactEntry object. The ContactProperty class on the diagram describes the interface with ID, label, value and type members for representing email or telephone number.

Contact could be assign to any number of groups. Diagram groups are represented by the ContactGroup class contains ID, Google ID, title, updated time, flag to mark whether the group is system and flag to mark whether the group was locally deleted. The ContactGroup could has any number of contacts represented by the ContactEntry entity.

To communicate with the server are required access token, refresh token and user email. For these purposes there is a User class contains all needed information.

#### 3.3 Application processes

This section describes processes in the application.

- **Search process** (see Figure 3.2) — user set cursor to the searching line edit field and type substring of user emails by which he wants to find needed contact. The application will react on every key pressed and will immediately display the results.

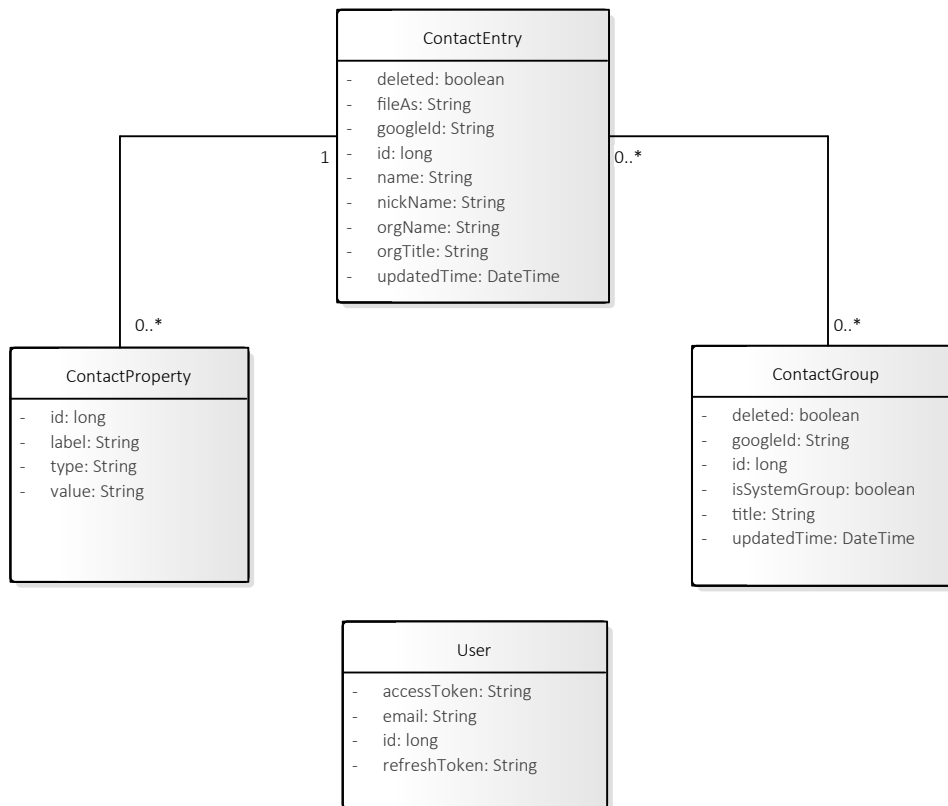


Figure 3.1: Domain Model

- **User login process** (see Figure 3.3) — when the application starts user can log in using either already saved Google account or can log in using new one. It is a usual web browser authentication process. If authentication was succeeded Google Contacts asks user to confirm the application access to his private data. Which is Google profile, email address and Google contacts. If authorization was succeeded too the application starts communication with Google Contacts API and main window dialog will be shown.
- **Create contact process** (see Figure 3.4) — user clicks on the "Create new contact" button and "Create new contact" window will be shown. After that user has to fill the required fields, such as telephone and email, and other contact information. When editing is finished he can save new contact by clicking on the "Save" button or he can discard data by clicking on "Cancel" button. Save button will provoke validation of required fields, i.e., it will check if they are not empty and if so

### 3. ANALYSIS

---

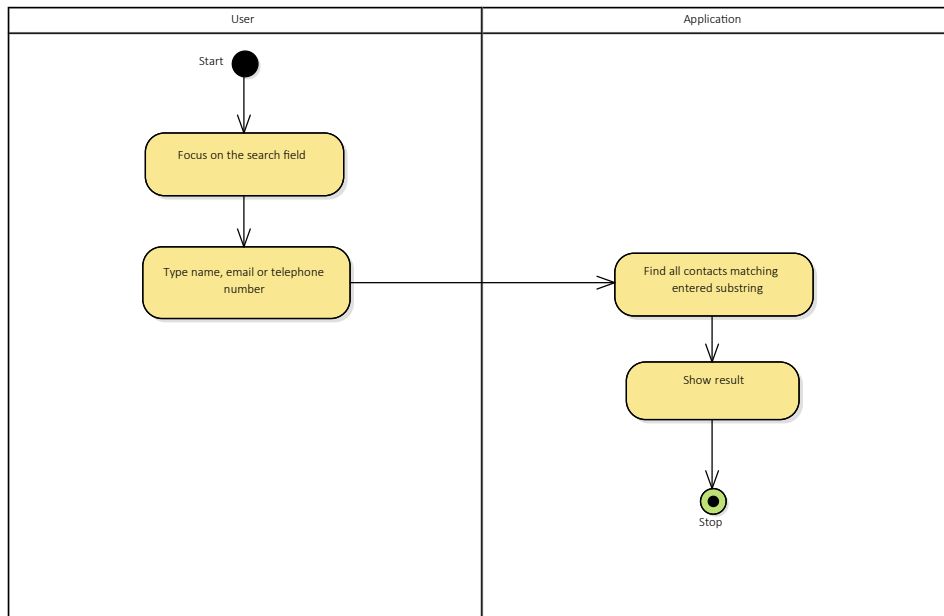


Figure 3.2: Search contacts application process

then check if they are filled in the correct way. If validation was successful then new contact is saved to a local database and will be pushed to Google Contacts with the next synchronization . Otherwise error message describing the problem will be shown and line edits containing invalid data will be red marked. "Cancel" button will show dialog with the warning message and two buttons, "Cancel" and "Discard changes". "Cancel" button will allow user to continue creating new contact and "Discard changes" button will discard changes and stop creating new user. Clicking anywhere outside the dialog is equal to pressing 'Cancel' button.

### 3.4 Use cases

This section describes Google Contacts use cases.

- **Synchronization use case** (see Figure 3.5) — synchronizes contacts and groups with Google.
- **Contact use case** (see Figure 3.6) — user can create, edit, merge, delete, import or export selected contacts.



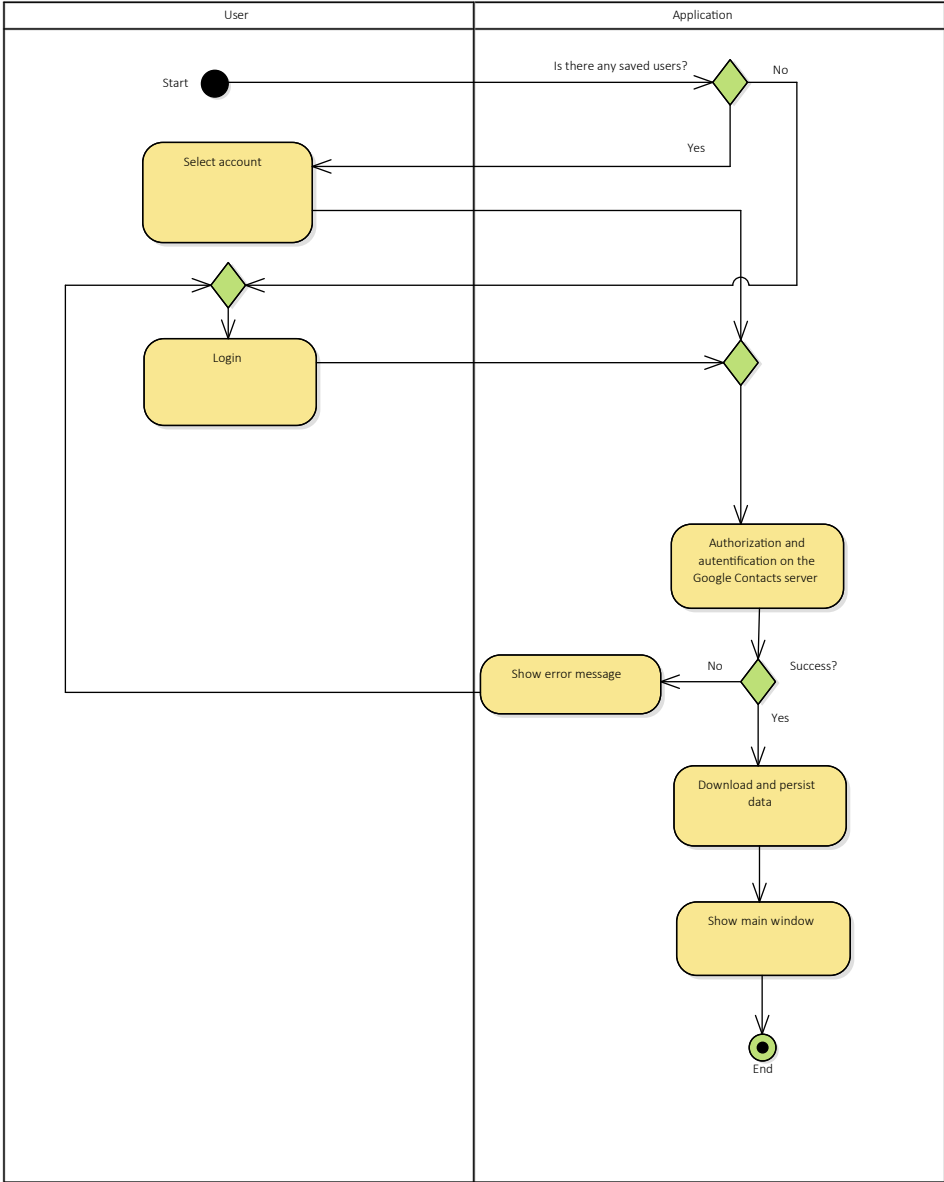


Figure 3.3: Login process in application

### 3. ANALYSIS

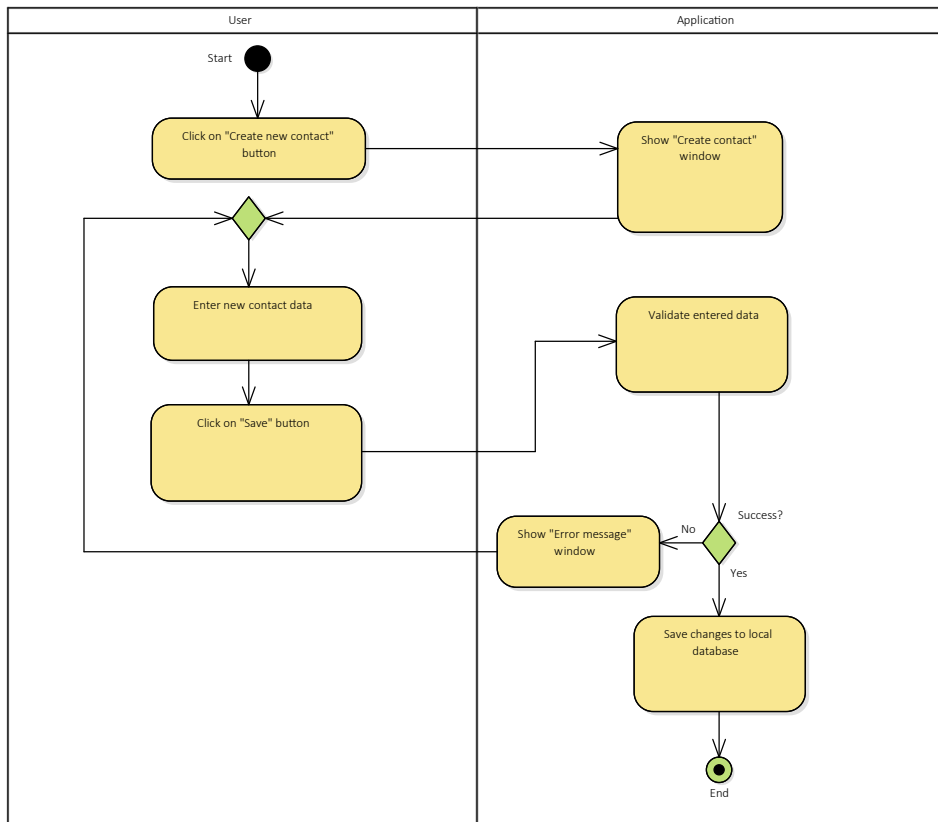


Figure 3.4: Create contact application process

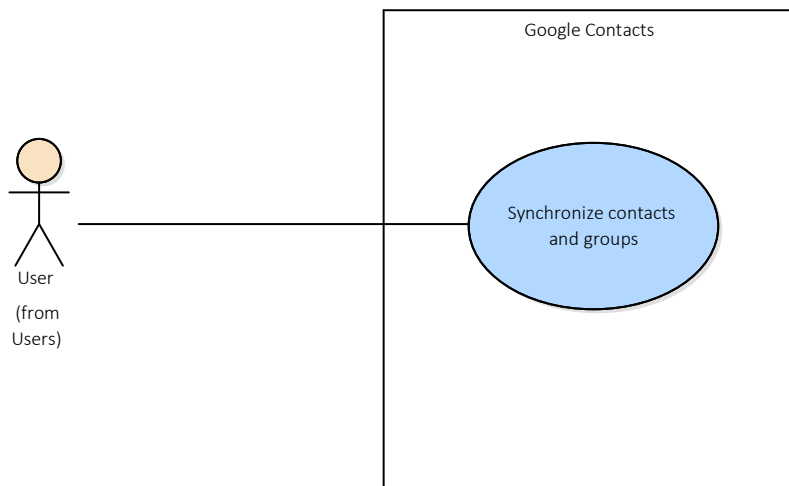


Figure 3.5: Synchronize use case

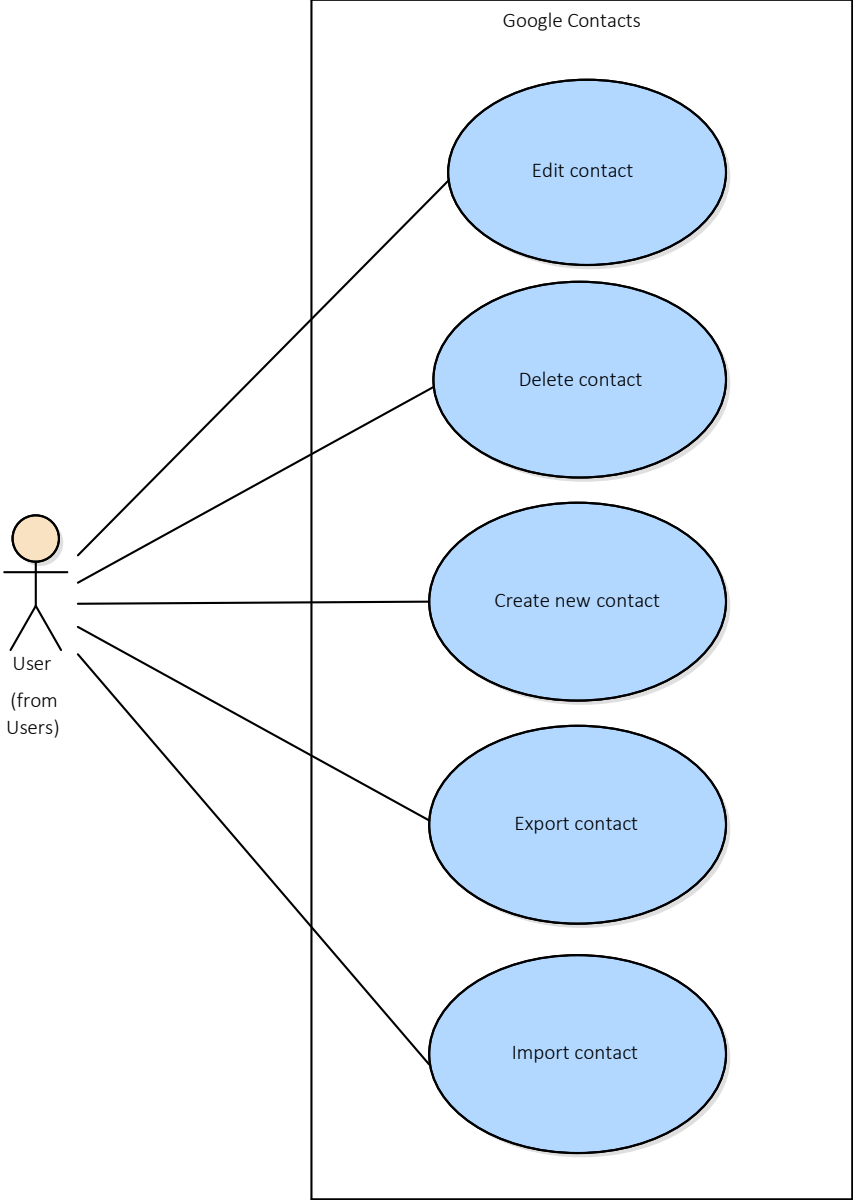


Figure 3.6: Contact use case



---

# Design

This chapter contains description of design Google Contacts application.

## 4.1 Wireframes

Wireframes is a standard technology that is used to design user interface. Main window wireframe (see Figure 4.1) includes table view with names, emails and phone numbers of user contacts. By checking radio button in the left side of main window it is possible to see all contacts specified by group. Add new contact wireframe (see Figure 4.2) defines layout of the functional elements such as input name, nickname and other fields. Field "File as" gets string which will be used for sorting contacts. New contact can be added to the selected groups by check boxes.

## 4.2 Database

Database is an organized collection of data. Database model determines the logical structure of a database and determines in which manner data can be stored, organized and manipulated.

Google Contacts database model (see Figure 4.3) is created by QxEntityEditor. QxEntityEditor is a graphic editor for QxOrm library. The editor provides a graphic way to manage the data model. The price of it's license key is 300 € per developer and 12 months of free updates. But thankfully, it is free up to 5 entities per project and Google Contacts database has just 4 entities [10].

Relation many-to-many between ContactEntry and ContactGrop will not be effective. Because database relation many-to-many is implementing by usage third table. So relation between ContactEntry and ContactGroup is unidirectional many-to-many relationship.

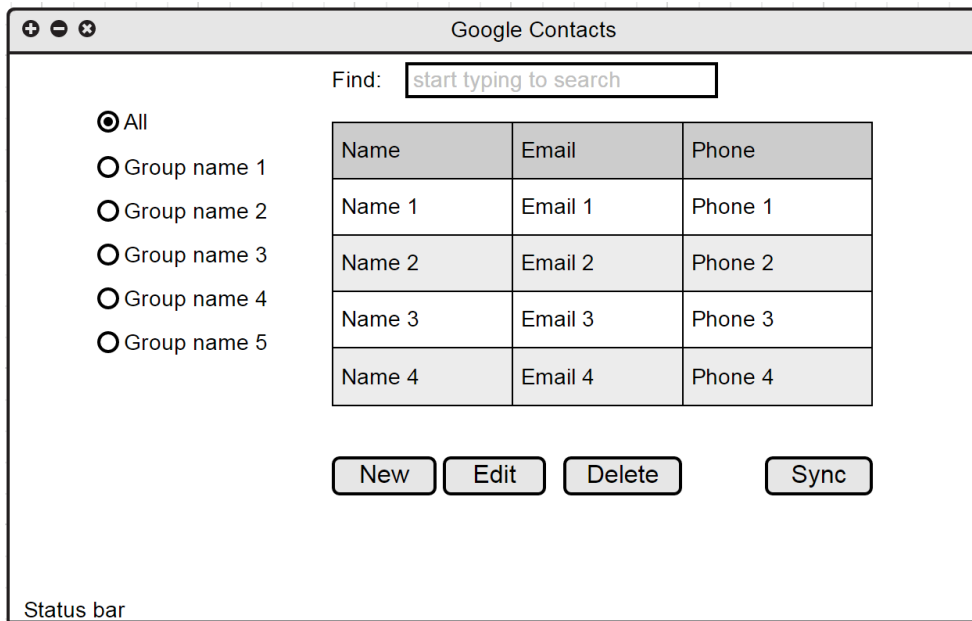


Figure 4.1: Main window wireframe

Google Contacts add new contact

Group name 1  
 Group name 2  
 Group name 3  
 Group name 4  
 Group name 5

Name:   
Nickname:   
File As:   
Company:   
Job title:

- Emails
- Phones

Figure 4.2: Add new contact wireframe

## 4. DESIGN

---

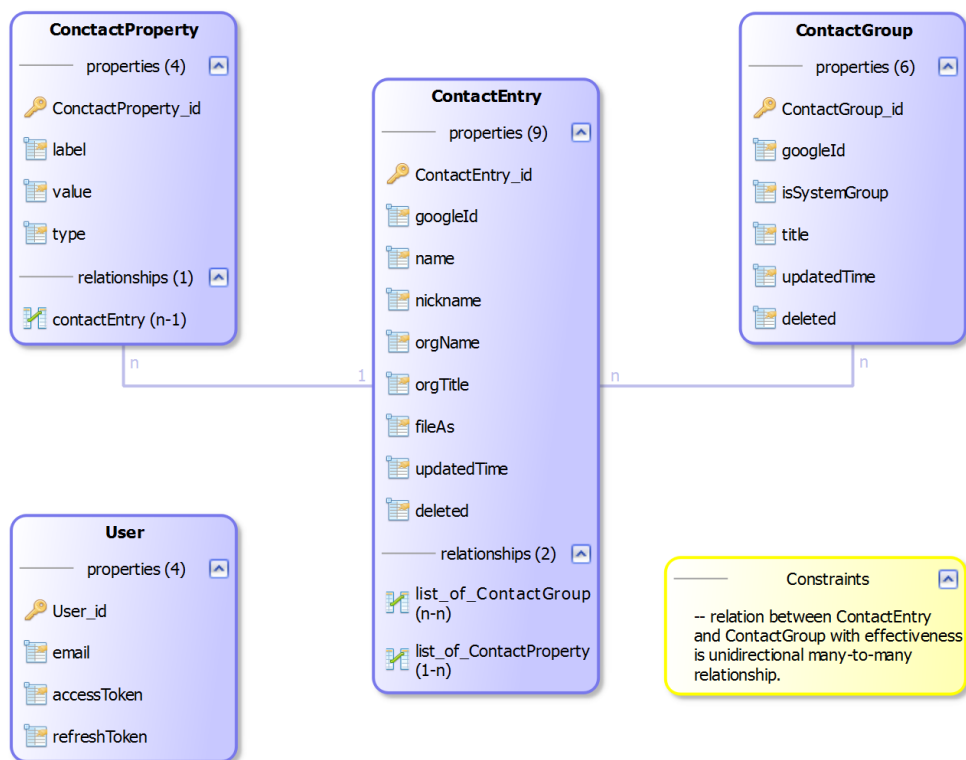


Figure 4.3: Database model



---

# Implementation

## 5.1 Application architecture

Application architecture divides into presentation, application and data layers. To explain how the application works I have designed the following overview diagram (see Figure 5.1). Communication between different objects in the application are based on Qt signals and slots. So there was no need to use multithreading.

### 5.1.1 Application layer

There are many client libraries that make easier to use Google APIs. And if your program is on Java, .Net or PHYTON it is much more easier for you to communicate with Google Contacts API. Unfortunately, there are no such libraries for C++ or even for C which would be still supported. So to communicate with Google Contacts API the application uses it's own classes.

The application layer contains three classes that are responsible for communication with Google Contacts API:

- **AuthManager** - class responsible for access and refresh tokens. It gets refresh token for new user and update the access token when it expires.
- **AuthServer** - helper class inherited from QTcpServer. It helps to configure the connection between the application and Google Contacts API.
- **AuthSettings** - helper class used to parse and store data from credential file.

AuthManager is a main object in communication with Google Contacts API.

AuthManager generates authorization URL and creates AuthServer that starts listening on some free port for reply of Google API. When AuthServer gets the reply it will parse it and emit signal with authorization code it got.

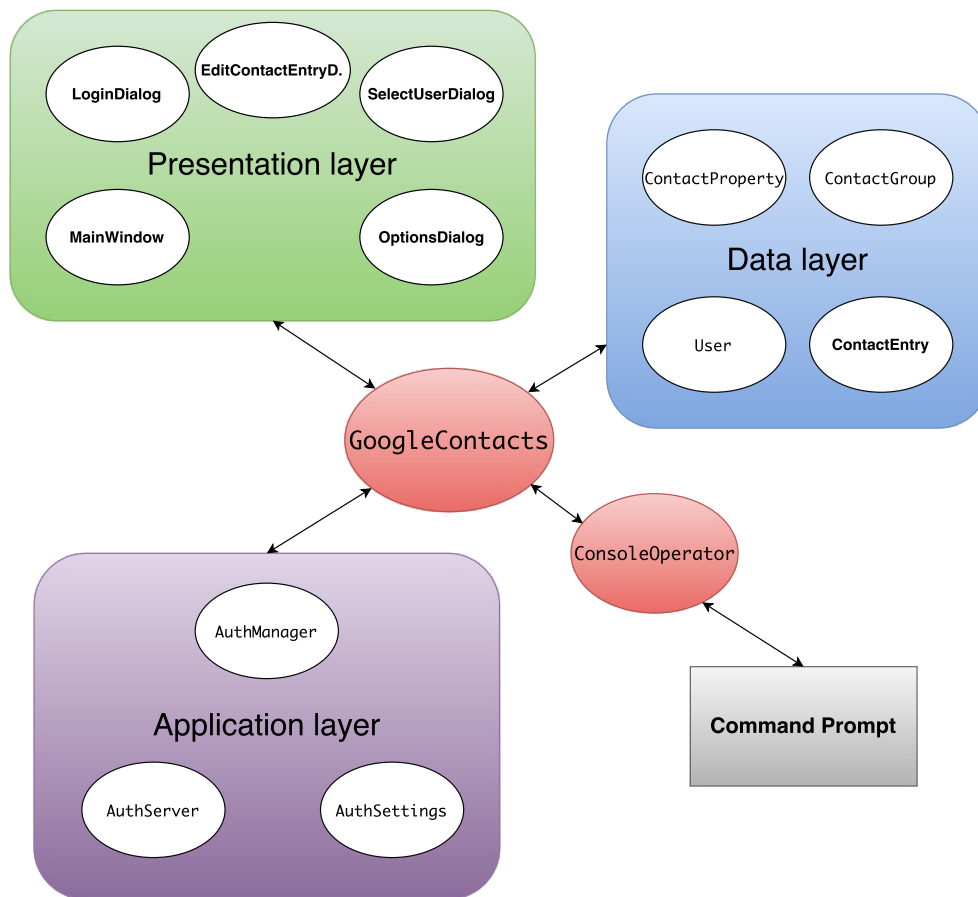


Figure 5.1: Application overview diagram

Then AuthManager generates GET request which consists the authorization code and some credentials data. On reply AuthManager gets access token and refresh token and it sends first request to Google Contacts to get email of authorized user. Upon success AuthManager creates new user with taken tokens and email. And emit signal newUserInitialiazed with new created user sent as parameter. So basically, its purpose is to initial new user and refresh access token.

During program life MainWindow asks AuthManager to refresh access token when it is expired.

### 5.1.2 Data layer

This layer is represented by four classes:

- **ContactEntry** - class represented one entry in contact list. This class holds user to whom it belongs.

- **ContactGroup** - class represented group.
- **ContactProperty** - helper class. It is a container for contact list values such as telephones, emails, dates etc.
- **User** - class represented one user. It holds access, refresh tokens and email of user.

All these classes are saved in database. The database is represented by Database class which consists all needed methods for saving, reading, searching and updating objects. For example, Database class provides method for getting all users which are in database. This method uses QxOrm `fetch_all` function that gets all users and returns them as container of pointers to User (see Listing 5.1).

When application needs to manipulate with objects it uses Database class. Inside of each four classes there is another layer properly QxOrm. So in other words Database class hides this complex logic and allows application works with objects as themselves, but inside objects are using QxOrm library.

```
QList<boost::shared_ptr<User>> Database::getUsers()
{
    QList<boost::shared_ptr<User>> users;
    qx::dao::fetch_all(users);
    return users;
}
```

Listing 5.1: Getting all saved users in database

### 5.1.3 Presentation layer

This layer is represented by dialogs:

- **MainWindow** — the main window of the application and all other dialogs are called from this dialog. It works with GoogleContacts class to get needed information from Google API and works with Database class to get or save contacts. User's contacts are displayed in the table. User could filter contacts by substring or by selected group. At the bottom of window is status bar which displays actual status of synchronization process.
  - **green** - synchronization was successful;
  - **red** - synchronization failed;

Status bar also displays date of last synchronization (see Figure 5.2).

- **SelectUserDialog** — dialog for selecting existing users. This dialog will be shown only if database contains some users.

## 5. IMPLEMENTATION

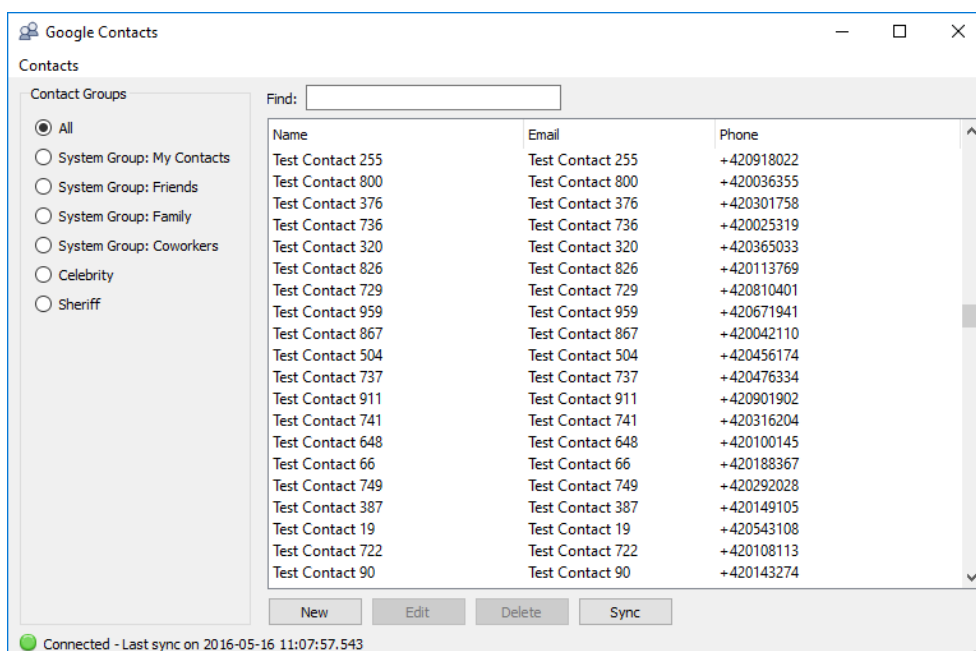


Figure 5.2: Main window of Application

- **LoginDialog** — simple dialog which contains only QWebView and displays web page by specified URL. This dialog is shown when new user needs to be authorized.
- **EditContactEntryDialog** — dialog for editing existed or creating new contact (see Figure 5.3).
- **OptionsDialog** — dialog that allows user to manage keyboard shortcuts (create, edit, delete, export, import, synchronize contacts) and automatic synchronization interval (see Figure 5.4).

### 5.1.4 Realization of application logic

Application logic is represented by class `GoogleContacts`. It does all the work of reading data from database, manipulations on them and communicates with Google API.

`GoogleContacts` class is created at the beginning of program with the user entry we will work with. Then it reads data from Database, i.e, contact groups, entries. User works with contacts but changes are not promptly synchronized, they are stored in list, and will be applied with the next synchronization with Google Contacts. Synchronization call the `syncGroupsAndContacts` method. Which synchronize both sides (local database and Google Contact database). Synchronization process looks as follow:

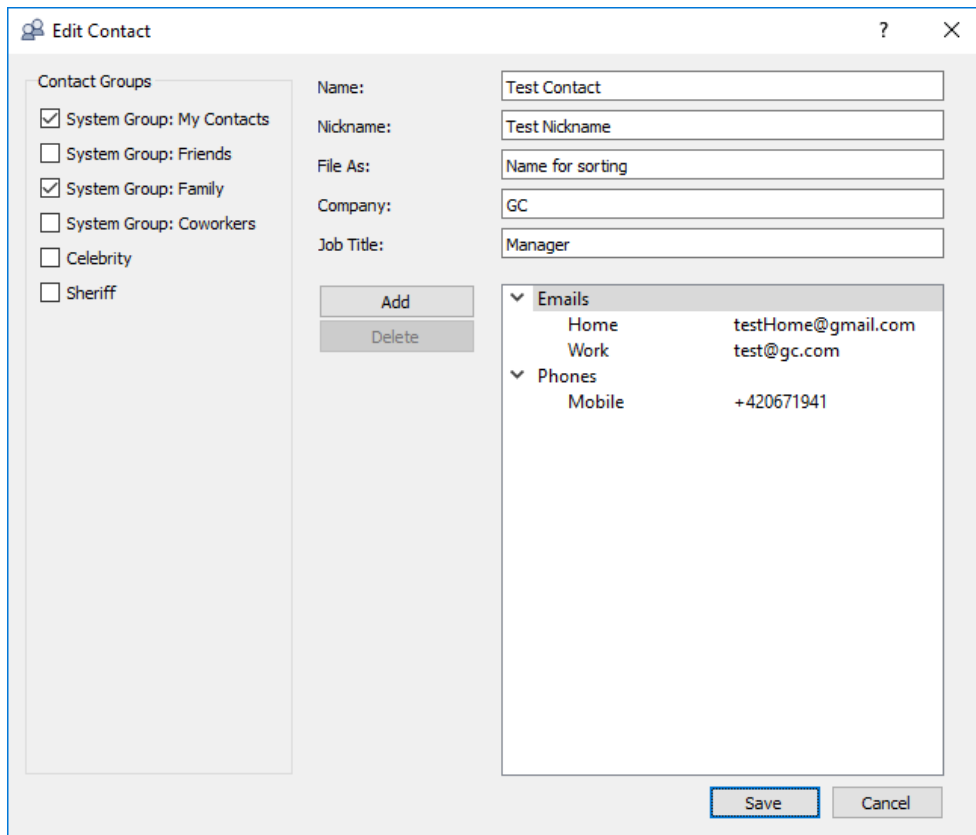


Figure 5.3: Edit contact dialog

#### 5.1.4.1 Synchronization of contacts

- application sends request to get the contacts;
- upon success, it gets reply and parses it;
- check if some of the contacts are marked as deleted, and if so then delete the contact in the local database;
- check if the contact is new, and if so then create the new contact and save it to the local database;
- while all user changes that were saved to the list are not fulfilled, the application will do the next:
  - sends request to Google API;
  - gets reply;
  - analyze reply;
  - react on the reply.

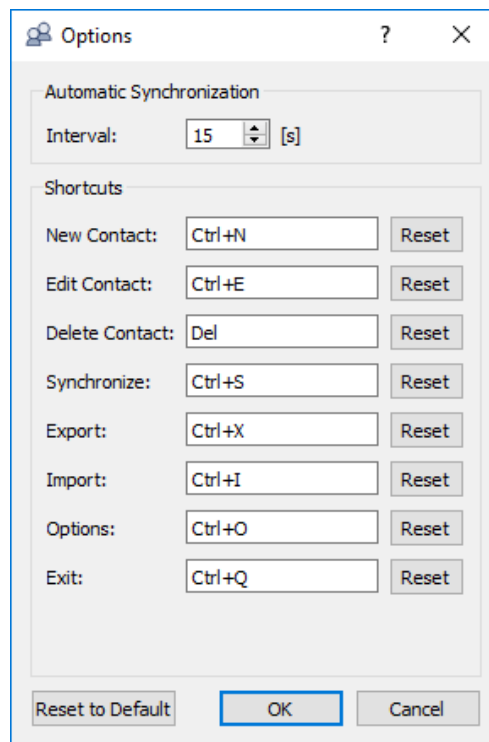


Figure 5.4: Google Contacts options dialog

- GoogleContacts class emits one of these four signal types:
  - **void groupsSyncSuccessful()** - groups synchronization passed successful;
  - **void contactsSyncSuccessful()** - contacts synchronization passed successful;
  - **void authorizationError()** - authorization error, could be emitted when access token is no more valid;
  - **void otherError(QNetworkReply::NetworkError error)** - other type of error.

These signals will provoke to some actions on presentation layer. For example the table with contacts will be updated or some error message will be shown and so forth.

GoogleContacts class suggests that authorization is already pass and all needed data for communication is stored in the user. If it turns out that user is not authorized, GoogleContacts emits authorization error. Such kind errors are handled in application layer namely by AuthManager. AuthManager will processed the problem and emit result. Upon success it's signal will awake syncGroupsAndContacts method. So this synchronization method just trying

to load data with valid token but if such token does not exist, it will not try to resolve the problem.

The ConsoleOperator class performs a similar role to GoogleContacts class but it communicates with command line.

### 5.1.5 Command line version of the application

This version of the application is a separate program which can be launched from command line with one of the arguments.

Commands and their arguments:

- **-f "substring"** - search contacts.
- **-c "name"** - create new contact.
- **-d "substring"** - delete contact.
- **-e "fileName"** - export contacts.
- **-i "fileName"** - import contacts.

Program is cross-platform same as GUI version of the application. The main function of this program works with ConsoleOperator class providing functions for commands described above. These functions in ConsoleOperator class do not work directly with application or data layers, instead they work with GoogleContacts class.





# Testing

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. It is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

To catch bugs during debugging the application uses its own asserts.

`ASSERT` (condition, "Assert message.");

If condition is false it shows warning dialog with error message (see Figure 6.1).

## 6.1 Unit tests

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually [11]. Executing unit tests is very fast and therefore can be run after any major change of code and verify that maintaining functions work correctly. Unit tests are not appropriate for testing parts of code, which require changing data in the database. Before you can run the tests again data in the database must be returned to its original state, and hence the difficulty of tests is much more higher.

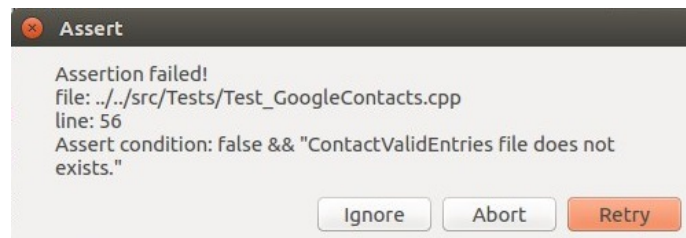
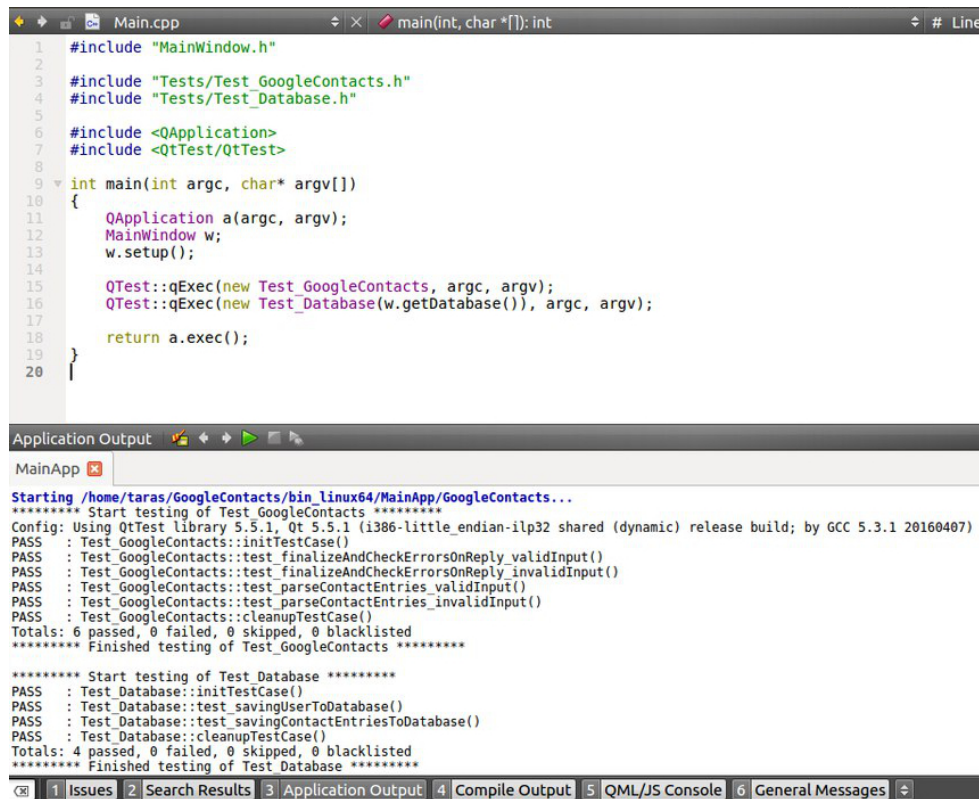


Figure 6.1: Assertion failed

## 6. TESTING



```
1 #include "MainWindow.h"
2
3 #include "Tests/Test_GoogleContacts.h"
4 #include "Tests/Test_Database.h"
5
6 #include <QApplication>
7 #include <QtTest/QtTest>
8
9 int main(int argc, char* argv[])
10 {
11     QApplication a(argc, argv);
12     MainWindow w;
13     w.setup();
14
15     QTest::qExec(new Test_GoogleContacts, argc, argv);
16     QTest::qExec(new Test_Database(w.getDatabase()), argc, argv);
17
18     return a.exec();
19 }
20
```

Application Output

```
MainApp
Starting /home/taras/GoogleContacts/bin_linux64/MainApp/GoogleContacts...
***** Start testing of Test_GoogleContacts *****
Config: Using QTest library 5.5.1, Qt 5.5.1 (i386-little_endian-ilp32 shared (dynamic) release build; by GCC 5.3.1 20160407)
PASS : Test_GoogleContacts::initTestCase()
PASS : Test_GoogleContacts::test_finalizeAndCheckErrorsOnReply_validInput()
PASS : Test_GoogleContacts::test_finalizeAndCheckErrorsOnReply_invalidInput()
PASS : Test_GoogleContacts::test_parseContactEntries_validInput()
PASS : Test_GoogleContacts::test_parseContactEntries_invalidInput()
PASS : Test_GoogleContacts::cleanupTestCase()
Totals: 6 passed, 0 failed, 0 skipped, 0 blacklisted
***** Finished testing of Test_GoogleContacts *****

***** Start testing of Test Database *****
PASS : Test Database::initTestCase()
PASS : Test Database::test_savingUserToDatabase()
PASS : Test Database::test_savingContactEntriesToDatabase()
PASS : Test Database::cleanupTestCase()
Totals: 4 passed, 0 failed, 0 skipped, 0 blacklisted
***** Finished testing of Test Database *****
```

Figure 6.2: Unit tests

GoogleContacts class is tested using Qt Test framework which provides classes for unit testing Qt applications and libraries. All public methods are in QTest namespace. In addition, the QSignalSpy class provides easy introspection for Qt's signals and slots [12].

Test\_GoogleContacts is a testing class that consists unit tests to test GoogleContacts class methods:

- Parse ContactEntry.
- Validate Google API reply.

Test\_Database is a testing class that consists unit tests to test Database class methods:

- Save User to database.
- Save ContactEntry to database

All unit tests are automatically run before every program launch (see Figure 6.2).

## 6.2 Manual testing on different operation systems

The application was tested by many users on some major platforms, i.e., Windows 7, Windows 8, Windows 10, Mac OS X El Capitan and Ubuntu 16.04. Found bugs was fixed and now Google Contact application works on these OSs' more smoothly, without errors and really quickly.

For manual tests was created test Google account with one thousand contacts.

**username:** petrytar.testers.0@gmail.com

**password:** testPassword



---

## Adaptation for other platforms

Because Qt is a cross-platform application framework the program can be built and run on the different platforms. Before building, you will have to go through the next steps:

1. Download QxOrm library (for example from: [http://www.qxorm.com/qxorm\\_en/download\\_details.php](http://www.qxorm.com/qxorm_en/download_details.php)).
2. Download Boost library (for example from: <https://sourceforge.net/projects/boost/files/boost/1.60.0>).
3. In QtCreator add variable BOOST\_INCLUDE with path to downloaded Boost library.
4. In QtCreator build release version of QxOrm library.
5. Add just compiled QxOrm libraries to project.

Now you can build and run the program. Depending on the operation system that you have you might will have to add some libraries or install utilities. To create independent release version just build program in release mode and copy some of standard Qt libraries into the output folder. You can find inspiration in attached Compact Disc.



---

## Features

These are new features that will be added in Google Contacts 1.0:

- Copy, cut, paste and merge selected contacts.
- Create, edit, delete and merge groups.
- Delete all contacts from selected group.
- Seeing contacts from multiple accounts.
- Multi-editing contacts.
- Add user photo.
- Change user without closing the program.
- Auto log in possibility.
- Improve getting entries from Google Contacts.





---

## Conclusion

The result of this bachelor thesis is an application that allows user to work with Google Contacts. It is cross-platform application, but the application compiled for one Unix flavor will probably not run on a different Unix system, so I decided do not add release version for Unix system. In repository there is only release version for Windows. The application has GUI and command prompt single request versions. It's design is nice and simple, easy to understand and use. Configurable keyboard shortcuts improve contacts managing. The application also remember users that was logged on. It allows switching between users without going through the authorization process. With available import and export to XML format function it is now possible to create backups of contacts, transfer contacts between users and so on. User can also operate the database directly. It is a standard SQLite database and it is created with the first application start.

It wasn't all romantic. Because I didn't have experience on working with APIs and there are no helping libraries that make communication easier for C++ I had had to spend a lot of time on studying and implementing application layer. But now I am really glad to have chosen this bachelor's thesis. It was really interesting to implement and I am happy that I was able to create a stable program which can be improved in future. Some functionality that wasn't required but might be useful for users are added to features which I am planning to implement.

The application was tested on different platforms and showed good performance in reliability and speed. Even specially created Google test account with around 1000 contacts does not slow filtering contacts. But getting them from Google Contacts was long enough.

The application is available in the public repository on GitHub.



---

## Bibliography

- [1] Wikipedia. *Google Contacts [online]*. March 2016, [Cited 2016-04-03]. Available from: "[https://en.wikipedia.org/wiki/Google\\_Contacts](https://en.wikipedia.org/wiki/Google_Contacts)"
- [2] h.ogi blog. *Synchronizing Google contacts and Thunderbird address books [online]*. May 2008, [Cited 2016-05-01]. Available from: "<http://hogiblog.blogspot.cz/2008/05/synchronizing-google-contacts-and.html>"
- [3] Google Developers. *Google Contacts API version 3.0 [online]*. March 2016, [Cited 2016-04-03]. Available from: "<https://developers.google.com/google-apps/contacts/v3/>"
- [4] OAuth. *What is it For? [online]*. September 2007, [Cited 2016-04-03]. Available from: "<http://oauth.net/about/>"
- [5] Google Developers. *Using OAuth 2.0 for Installed Applications [online]*. February 2016, [Cited 2016-04-03]. Available from: "<https://developers.google.com/identity/protocols/OAuth2InstalledApp>"
- [6] Bihis, C. *A Bird's Eye View of OAuth 2.0*, volume 205. PACKT, 2015, ISBN ISBN 978-1-78439-540-7, 29 pp.
- [7] Blanchette, J.; Summerfield, M. *Foreword*, volume 752. Prentice Hall, 2008, ISBN ISBN-10: 0-13-235416-0, 7 pp.
- [8] Qt Documentation. *Signals & Slots [online]*. January 2016, [Cited 2016-04-29]. Available from: "<http://doc.qt.io/qt-5/signalsandslots.html>"
- [9] Abran, A.; Bourque, P.; Dupuis, R.; et al. *Software Requirements*, volume 200. Angela Burgess, 2001, ISBN ISBN 0-7695-2330-7, 35 pp.
- [10] Marty, L. *QxEntityEditor [online]*. 2016, [Cited 2016-05-02]. Available from: "[http://www.qxorm.com/qxorm\\_en/download.html](http://www.qxorm.com/qxorm_en/download.html)"

## BIBLIOGRAPHY

---

- [11] Rouse, M. *What is unit testing?* [online]. January 2007, [Cited 2016-05-10]. Available from: "<http://searchsoftwarequality.techtarget.com/definition/unit-testing>"
- [12] Company, T. Q. *Qt Test 5.6* [online]. April 2016, [Cited 2016-05-10]. Available from: "<http://doc.qt.io/qt-5/qttest-index.html>"

## Acronyms

<b>API</b>	Application Programming Interface
<b>CSV</b>	Extensible Markup Language
<b>DDL</b>	Data Definition Language
<b>GUI</b>	Graphical User Interface
<b>HTTP</b>	Hypertext Transfer Protocol
<b>IDE</b>	Integrated Development Environment
<b>JSON</b>	JavaScript Object Notation
<b>MUA</b>	Mail User Agent
<b>Mutt</b>	The Mutt E-Mail Client
<b>ODBC</b>	Open Database Connectivity
<b>OS</b>	Operating system
<b>SQL</b>	Structured Query Language
<b>SSL</b>	Secure Sockets Layer
<b>URL</b>	Uniform Resource Locator
<b>XML</b>	Extensible Markup Language



## **Contents of enclosed CD**

## B. CONTENTS OF ENCLOSED CD

---

readme.txt .....	the file with CD contents description
release .....	the directory with application release
├─ platforms .....	the directory with platforms
├─ sqldrivers .....	the directory with sql drivers
├─ gcontacts.exe ..	the directory with command line release executables
├─ GoogleContacts.exe .....	the directory with gui release executables
├─ *.dll .....	the files with shared library
resources .....	the directory with resources
├─ OAuth2.0_client_ID.json .....	the file with credentials
├─ resources.qrc .....	the file with application resources in qrc format
├─ executables	
├─ *.png .....	the icons that are used in program in png format
src .....	the directory of source codes
├─ 3rdParty .....	the 3rd part helping libraries
├─ Data .....	the directory of data source codes
├─├─ Auth .....	the part responsible for application layer
├─├─ Model .....	the part responsible for data layer
├─ MainApp .....	the part responsible for presentation layer
├─ MainAppConsole .....	the part responsible for command line app
├─ Tests .....	the part responsible for unit tests
├─ thesis .....	the directory of $\text{\LaTeX}$ source codes of the thesis
├─ GoogleContacts.pro	the file with configurations for the project codes
text .....	the thesis text directory
├─ thesis.pdf .....	the thesis text in PDF format