Backend As A Service Using the Example of Enginio a Cloud Service for Qt

Wilfried Kohl
wilfried.kohl@uni-ulm.de

Abstract—Cloud computing becomes more common every day. Mobile applications and devices become more and more network oriented. This means a fast way to develop network applications is demanded. Given that many network applications use backends, Backend as a Service (BaaS) is an approach to speed up the backend creation. Every backend system needs to be set up, managed and maintained. Not every developer wants to be limited by the backend or have high costs setting the backend up. So there was the need for a frontend developer friendly cloud service. This paper describes the resulting backend offerings and exemplifies the service using the example of Enginio—a BaaS offering by digia.

1 Introduction

Since the existing cloud service types—Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS)—did not meet the needs of mobile application developers, there was a need for another cloud service type Backend as a Service (BaaS). The main focus of mobile application developers is a usable, appealing and effective frontend application. For this reason they do not like to have the duty of maintaining or laboriously planning and setting up the backend. Of course there are other developers than mobile developers who feel the same way about backends. This is the main reason why another backend service type was established.

One example for a BaaS service is Digias BaaS service for Qt. Under the name *Enginio* the company Digia wants to provide a simple way to create a backend service for Qt/C++ applications. Enginio is described as BaaS which means *Backend as a Service*. Digias backend service provides cross platform delivery in the combination Qt and Enginio. The features are described as "... backend data storage, user management, backend actions ..." and "... fully managed backend infrastructure ..." [15].

This paper describes the structure of Enginio, the basic structure of Qt and at least a part of what is possible with Enginio 1.0. It is important to notice that Enginio was in the beta phase until 12.12.2013 and is still under further development, so some features like backend actions are so far not available.

Section 2 of this paper contains a short overview of the most accepted cloud service types and the according classification of backend as a service. Subject of Section 3 is the general structure of backend as a service and a catalog of a few BaaS offerings. Subsequently the main section—Section 4—contains an introduction to Qt. The main focus of this section is on the Enginio backend for Qt with detailed descriptions of the Enginio dashboard (the graphical user interface of the backend). These descriptions contain Enginios object types and its options. Afterwards follows an introduction of the Enginio API (Section 5). It contains the provided classes, examples of the REST API, short code examples for the basic operations and a short paragraph about security of Enginio. The last section contains a short conclusion and a prospect.

2 Cloud service types

This section is short explanation of the three types of cloud services and their correlation. Figure 1 shows a classification of the cloud service types. According to

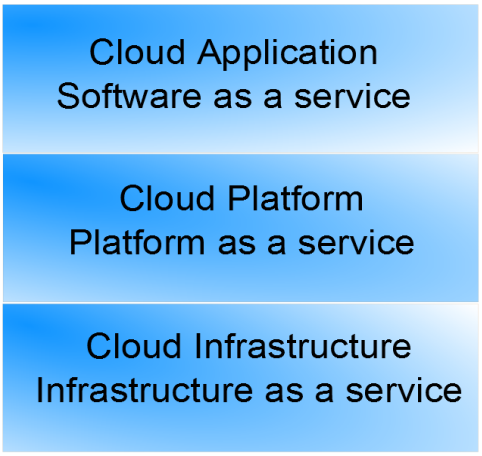

Figure 1. Cloud computing structure according to the National Institute of Standards and Technology (NIST)

an article of the National Institue of Standards and Technology (NIST) [20] the three cloud service types

can be roughly distinguished by the control the cloud subscriber has or alternatively the control the cloud provider has. There is of course other criteria and there are other classification appraches as e.g. described in "Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking" [22].
Backends or generally computer sytems can be subdivided in different levels. In the next sections this levels—originally utilized by NIST—are used to describe differences between the different service types:

1) application
2) platform architecture
3) virtualized infrastructure
4) hardware
5) facility

As one might assume the different levels can be construed as abstraction levels for example a developer who only uses the application level does not need to know all the specifics of the underlying levels (the levels with higher numbers in the list above).

2.1 Platform as a Service (PaaS)

According to NIST PaaS would leave the cloud subscriber control in the platform architecture and the application while the cloud provider has control over the virtualized infrastructure hardware and the facility of the service. PaaS is build upon the cloud infrastructure stated by Geoffrey Raines and Lawrence Pizett [19] and also shown in the Figure 1. PaaS also includes storage (e.g. databases) and computing power. The range of services is wide spread for example online environments for programming and computation as well as business process integration. An example for PaaS is the Google App Engine. The Google App Engine provides development tools and a host service in the cloud. This means developers can create their applications in the cloud and also host them via the same cloud service. Another example for PaaS is COSCA [21]. COSCA is a cloud service which implements 11 requirements e.g. scalability and isolation. COSCA executes applications which consist of components in a distributed system.

2.2 Infrastructure as a Service (IaaS)

The National Institut of Standards and Technology (NIST) [20] puts the control of the cloud subscriber in the IaaS to application, platform architecture and virtualized infrastructure while the cloud provider only controls the hardware and facility. Through the use of IaaS subscribers bypass providing or buying the basic hardware and software structure as well as housing and the associated managing tasks. An example for IaaS is Amazon Elastic Cloud Compute (Amazon EC2). Amazon EC2 provides scalable computational power so that developers only have to pay for the backend power their application needs.

2.3 Software as a Service (SaaS)

In SaaS the cloud subscriber has not much control or tasks to do before using the cloud. The subscriber has only control in the application while the cloud provider has to manage the facility, hardware, the virtualized infrastructure, the platform architecture and possibly also a part of the application. An example for SaaS is GoogleDrive or Google Docs. Google Drive is a data storage which allows users to store their data in the cloud and therefore allowes worldwide access. Google Docs allows edit their files e.g. Microsoft Word documents in the cloud.

2.4 Classification of Backend as a Service (BaaS)

The classification of BaaS is somewhere in the region of Software and Platform as a Service (SaaS and PaaS). On the one hand some BaaS providers allow subscribers to choose their own database and use preimplemented push-services or other applicational support and on the other hand some providers only offer one database and no preimplemented applicational support. Since frontend applications are fully independent of BaaS offerings and BaaS mostly provides universally important backend options, the BaaS offerings allow developers to only concentrate on the applications data.
Under the control aspect the user controls the application and the BaaS provider controls everything else. The provider has to provide availability and other requirements important for cloud services.

3 Backend as a Service (BaaS)

According to siliconindia.com[24] most existing IaaS offerings were too difficult or inconvenient to use and most PaaS offerings were too much work for their utility. BaaS offerings should be easy to use and not too much work for their purpose and security. Although the urge for BaaS offerings mostly came from mobile app developers these offerings can of course be used for non-mobile applications as well.
The structure of the backend as a service offerings likely looks like the Figure 2. Different Frontend applications can communicate with the backend service. The backend service is able to store, alter and delete objects in the database. The service can be able to execute functions written by the developer, depending on the offering.
This structure allows BaaS providers to adjust their offerings exactly to the needs of frontend developers. The developer can define or use backend functions and objects he/she wants to store in the database. The developer does not need to define and maintain backend properties he/she actually does not care about (e.g. the used database). The decision which database, which operating system and similar decisions are in most examined offerings predetermined by the service provider.

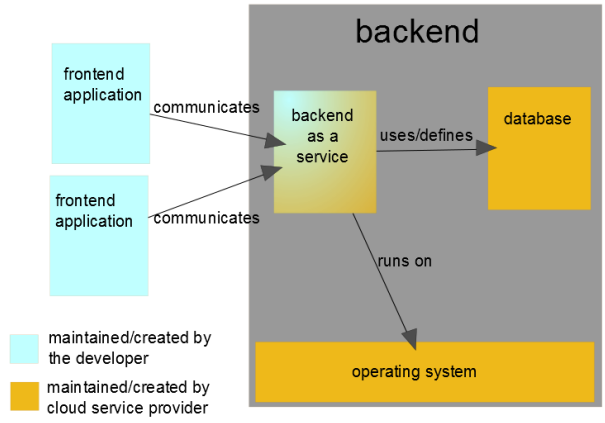

Figure 2. The structure of BaaS

3.1 BaaS offerings

Most of the following offerings provide services like push notifications and social network integration.

3.1.1 Anypresence

Anypresence [1] offers a BaaS solution. They advertise it as "*Revolutionary* **Enterprise Backend-as-a-Service** Platform". The list of features[10] is quite long and includes features like a *Guided Starter App, Source Code Management, database selection, SMS Messaging , Email Notification*. So far Anypresence supports SDKs for IOS Objective-C, Android Java, Windows Phone, Javascript Backbone.Js, Java, Test Scripts and Documentation.

3.1.2 ApiOmat

ApiOmat [2] offers a service for Android, Objective C, Phyton, PHP, Java and Javascript. Creating the first backend is very easy. On their website they offer a dashboard in which you can define classes and integrate modules e.g. a chat module. Every backend has the modules *User* and *Role* for authentication and authorization purposes. Defining access privileges works via ACL (Access Control Lists). Unfortunately the use of ACL is not included in the free "Basic" account. The backend is identified by a Base URL and an ApiKey. Calling Backend functions works with code injections[11]. The callable javascript functions can be defined in a website integrated javascript code editor.

3.1.3 Parse

Parse [8] provides SDKs for OS X, iOS, Javascript, Android and .Net. The REST (Representational State Transfer) API allows every service which can send HTTP Requests to comunicate with the backend. Setting up a backend with Parse is easy and intuitive. Additionally Parse provides Third Party Libraries on their website. These libraries are provided by the community of Parse developers and provide a variety of programming languages e.g. Corona, ActionScript, Ruby and also Qt. Backend functions are written in javascript.

3.1.4 Cloudmine

Cloudmine [5] provides Android/ Java, iOS and a JavaScript library as well as a REST API. Server Code can be written in JavaScript but also in Java. Cloudmine also offers mobile analytics and also offline workflow which means even if a mobile device loses the internet connection the changes will be applied in the backend when the connection is reestablished.

3.1.5 BAASBOX

BAASBOX [4] is an open source BaaS. On the one hand this means in order to use your backend functionality you need a server, on the other hand you do not have to pay a third party for the service. This service is written in Java so there is no need for a specific operating system on the server. BAASBOX provides SDK features for iOS SDK and AndroidSDK. It also provides a REST API.

3.1.6 App42 Cloud API

App42 Cloud API [3] provides many SDKs e.g. for Windows Phone, Android iOS, J2ME, C#, Java, Marmalade. The backend uses a NoSQL Storage to store the transfered Json objects. App42 provides backend services like sending emails or messages, but also allows developers to create custom server side functions.

4 Qt and Enginio

Preceding to the deeper analysis of the backend a short introduction to Qt.

4.1 About Qt

Qt was developed to provide cross-platform development for C++ applications and easy Graphical User Interface (GUI) creation. Qt established itself and is now also available for other programming languages (e.g. Phyton, Ruby[16]).

Among others the Qt Creator—the main integrated development environment (IDE) for Qt—allows to create Qt widgets and Qt Quick applications. The widgets use the standard C++ structure with header-files and class files in which Qt classes can be used. Qt Quick applications use additionaly to the class and header files also qml-files. Qml-files are written in the QML (Qt Meta-object Language)[17].

The first Qt version was developed "by Haavard Nord (Trolltech's CEO) and Eirik Chambe-Eng (Trolltech's president)"[25]. In May 1995 Qt became publicly available. In 2008 Trolltech was bought by Nokia [23] and Nokia took on the Qt development. "Digia [acquired] Qt commercial licensing business from Nokia" [18] in March 2011. In 2012 digia took over the full development of Qt from Nokia [9].

4.2 The Enginio backend

Enginio [12] is the BaaS approach for Qt developed by Digia. The structure of Enginio is overall similar to the general structure of BaaS offerings (Figure 2). Developers can communicate with the backend via an API in Qt/C++ or via the REST API from every programm which can send/receive HTTP-Requests/Responses.
A short usecase to illustrate the process of standard requests is shown in Figure 3.

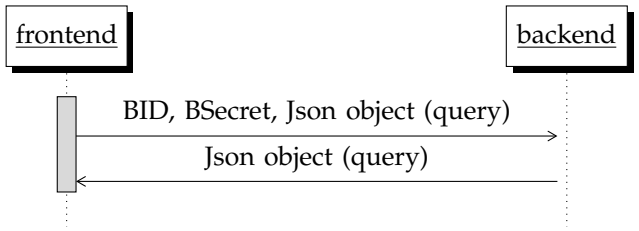

Figure 3. Standard communication with the backend via Enginio

The Figure 3 shows that requests must contain the backend id (BID) and the backend secret (BSecret) as well as a query (a Json object) which specifies which components to get, alter, add or delete from/in the backend. The reponse also contains a Json object which contains the requested components (assuming the requesting user has the appropriate rights for the request).
Users, usergroups as well as object types and access policies can be defined via the dashboard. The dashboard is the graphical user interface available via the Enginio website. In the dashboard developers can manage their projects (/backends). As mentioned in the introduction, Enginio is not fully implemented yet, thus not all promised functionality is usable at the moment (5.11.2013).
Available functionality of the dashboard (so far):

- define *object types* with their properties and relationships
- add *data validation* to properties of a object type
- edit *reference constraints*
- specify *access* to object types
- create and alter *objects* of an object type
- administrate *users* and *usergroups*

The above mentioned functionality is defined more specificly in the following subsections-

4.3 Object type

Every object type has a name which the developer defines.

4.3.1 Properties

There are some internal properties which the backend creates automatically cf. Figure 4
Thus the meaning of these properties is mostly clear. The description on the right-hand side of the colon should help to understand the internal representation.

- id : objectid
- createdAt : time
- creator : reference to a user
- objectType : String
- updatedAt : time
- updater : reference to a user

Figure 4. Internal properties of Enginio

Of course it is possible to define your own properties, they are marked as custom instead of internal. These properties always need a name and need to have one of the following datatypes:

- String
- Number
- Boolean
- Datetime
- Hash
- Array
- Geolocation
- Ref (standing for “reference”)

When choosing “Ref” the target object type needs to be selected in the selection list. There are self defined object types as well as users, usergroups and files. Every custom property can be indexed which “... improves [the] query performance and enables fulltext-search capabilities”. In case the custom property is a reference to a file, a file processor can be selected. The default selection is the ”Image processor”. So far there are no other processors choosable.

4.3.2 Data validators

Data validators can be added to a custom property. Every type of validator has a message which can be altered.
Validator-Types:

Type	Options
Required	
Length	minimum:Number, maximum:Number
Format	with :Regular expression
Uniqueness	caseSensitive:Boolean
Inclusion	in: Array
Exclusion	in: Array

4.3.3 Reference constraints

Under the headline “reference constraints” developers can define dependencies between their object type and the referenced object types. The options are:

- Do nothing
- Delete object
- Nullify Key

”Do nothing” is the default dependency.

4.3.4 Permissions

There are three different kinds of permissions: collection permissions, default object permissions and dynamic permission rules.

Collection permissions allow users or usergroups to read or create the corresponding object type. These permissions are object type specific.
Default object permissions differentiate between users/usergroups and the creator. These permissions are object specific. Selectable permissions are *read*, *update*, *delete* and *admin*.
dynamic permission rules define dependencies to other object types and objects. The object type can inherit permissions from referenced object types or inherit permissions from the object type and the target object.

4.4 Users

Many applications need access rights. In order to provide access rights the object type *Users* is needed. The dashboard has an extra tab for this object type. In this tab new users can be added, edited or deleted. When adding a user the developer can specify the properties *name*, *password*, *email*, *firstName* and *lastName*. When editing the developer can change all properties except for the password. The password ist not visible to the developer. The developer can also grant a user object permission. This means the user can get the permissions to *read*, *update*, *delete* or *admin* a user or a usergroup.
In the tab Usergroups the developer can also add a user to existing usergroups.

4.5 Usergroups

To group user e.g. to give those users the same rights. The Enginio dashboard provides tab for Usergroups. Usergroups can be added, edited or deleted. The usergroup data only consists of a name. The developer can add object permissions which grant the selected users or usergroups permissions to *read*, *update*, *delete* or *admin* this specific user group object.
Member Management Permissions allow developers to define users or usergroups who are allowed to *read*, *create*, *delete* or *admin* members of this usergroup.
It's also possible to view all members of a usergroup via the tab called "Members".

4.6 Environments

Developer want to create new backend versions in one environment while running their backend in another environment. For this reason dashboard offers two environments:
The **production** and the **development** environment. A created backend in the development environment can be deployed to the production environment. In this step the deploy mechanism will copy current object types as well as settings to the production environment. To reach the production environment from client applications developers need to update the key values (backend id and backend secret) in their client application. Developers need to be careful while deploying newer versions because the object types in the production environment will be overridden.

4.7 Using Enginio in Qt

To use Enginio in Qt a newer Qt 5.1 or newer is needed. The Enginio packages are going to be integrated in Qt with Qt 5.2 [7]. The next requirement is for OpenSSL to be installed. With those two requirements met, the Enginio Qt library can be downloaded and installed.

5 The Enginio API and its availability

Enginio is available in the form of C++ classes for Qt widget or Qt Quick applications or in the form of QML types Qt quick applications.
The **C++ classes** are:

- **The EnginioBasicAuthentication class,**
  which represents a by the backend authenticated user.
- **The EnginioClient class,**
  which handles the communication with the server.
- **The EnginioIdentity class,**
  which is an abstract base class which is used for different authentication methods
- **The EnginioModel class,**
  which is a QAbstractListModel that represents the Enginio data
- **The EnginioReply class,**
  which contains the data to previously sent request.

The **QML types** are:

- **The Enginio type,**
  which is responsible for the Enginio access
- **The EnginioModel type,**
  simplyfies the access of collections
- **The EnginioReply type,**
  is a reply to a previously sent request

Enginio is also available via the REST API on any development framework or operating system via HTTP-Requests as described in the Features of Enginio [14]. The REST API is described more accurately in this course of this section.

5.1 Using Enginio in Qt-Gui applications

After creating a Qt-Gui project, Enginio can be used as follows[6]. At first the Project file (.pro) must be altered so that it includes:
QT += enginio
The Enginio Client must be imported and initialized.
#include<Enginio/Enginio> and

Listing 1. Initialize the Client

```
QByteArray EnginioBackendId = "OWN_BACKEND_ID";

QByteArray EnginioBackendSecret =  "OWN_BACKEND_SECRET";

m_client = new EnginioClient(this);
m_client->setBackendId(    EnginioBackendId);
m_client->setBackendSecret( EnginioBackendSecret);
```

or alternatively with

Listing 2. Initialize the Client

```
QString backendId(    "YOUR_OWN_BACKEND_ID");
QString backendSecret(    "YOUR_OWN_BACKEND_SECRET");
EnginioClient *client =     new EnginioClient(backendId,
      backendSecret);
```

Now the backend can be used.
Lets assume there is an object type called lecturer with the only custom attribute name of the type String. Than the backend can be queried as follows:

Listing 3. Query the backend

```
QJsonObject json= QJsonObject();
json["objectType"]= QString(      "objects.lecturer");

m_client->query(json);

QObject::connect(m_client,
      SIGNAL(finished(EnginioReply*)),
      this,SLOT(replyFinished( EnginioReply*)));
```

This query returns a QJsonObject which contains a query and a result. The requested data can be found in the results QJsonObject.
Saving a new lecturer works like this:

Listing 4. Add a new object

```
QJsonObject query;
 query.insert("objectType", QString("objects.lecturer"));

query.insert("name",QString("name_obj"));
m_client->create(query);

connect(m_client,SIGNAL(finished(EnginioReply*)),
  this, SLOT(uploadFinished(EnginioReply*)));
```

A QJsonObject is created and gets the Attributes *objectType* and *name*. Afterwards the EnginioClient create-function is called, which will create the object in the backend. Connecting the signal finished with the slot uploadfinished results in an instant call of uploadFinished(EniginioReply*) when the signal finished(EnginioReply*) is emitted.
To update an object in the backend, the id of the object must be known. This can be done as shown in Listing 3.

Listing 5. Update an object

```
QJsonObject query;
query.insert("objectType", QString("objects.lecturer"));

query.insert("id",QString("obj_id"));
//set the new name:
query.insert("name",QString("new_obj_name"));
m_client->update(query);
```

Like updating an object id is needed when deleting an object from the backend.

Listing 6. Delete an object

```
QJsonObject query;
query.insert("objectType", QString("objects.lecturer"));
query.insert("id",QString("obj_id"));
client.remove(query);
```

5.2 Example of the REST API

The REST (Representational State Transfer) API allows users or more likely developers to use HTTP-Requests to communicate with the backend.

To use the REST API in Enginio any tool, which is aplicable to use HTTPS-connections and send/receive HTTP-Requests/-Responses can be used. The backend is identified via its backend id. The general backend access permission is granted through the backend secret. To communicate with the backend we need to send a HTTP-Request to the URL: *https://api.engin.io/v1/_RESOURCE_PATH_*

Examples:

To get the lecturers from our backend the URL: *https://api.engin.io/v1/objects/lecturer"* is needed for the HTTP GET Request.

Listing 7. HTTP-Request example

```
[...]
Enginio-Backend-Secret:  YOUR_OWN_BACKEND_SECRET
Enginio-Backend-Id:  YOUR_OWN_BACKEND_ID
[...]
Cookie: [...]
```

The update operation can be realized with the REST API like this:
Send a PUT-Request to the URL: *https://api.engin.io/v1/_RESOURCE_PATH_/_ID_* with _ID_ being the objects id.
The HTTP-Request Header has the same additional information as in Listing 7 (the BackendID and Backend-Secret) and another header field for the content type *Content-Type: application/json*
Furthermore the payload needs to be added:

Listing 8. Update object payload

```
{
"name":"_New_Name_"
}
```

Of course there are more operations like delete which are all structured similarly. Here is a short overview of used HTTP-Request types used for REST API [13] and their association to different functionalities:

GET	Retrieve data from the backend
POST	Create new data object
PUT	Replace an object
DELETE	Remove/Delete an object

The REST API also supports atomic updates which allows developers to use commands like e.g. “increment” and “decrement” to circumvent unwanted concurrent side-effects. Here is an example for an unwanted side-effect without the increment functionality in the backend:
The involved users are user A and user B. Every user who reads the variable x has to increment x, let x be 10. Here is the chronological order of operations the users perform:
A reads — B reads — A overrides x — B overrides x
The described scenario is an example for the **lost update problem**. What happens is A reads x=10, B reads x=10. Afterwards A and B increment their read values and send them to the backend to update them. In the backend

the value of x is twice overridden with 11, but actually the value should be 12 because A and B both read the variable x. In contrast when both users use the atomic backend functionality x is 12 because the backend increments x twice.
To login a user a HTTP POST Request is send to *https://api.engin.io/v1/auth/identity* containing the Backendid, BackendSecret as well as the username and password.

5.2.1 Atomic updates

In order to understand the commands it is important to remember thatn every command sends a Json-object to the backend. This object contains the command, the value which should be altered and the value the command needs to alter the backend value. Here is a short list of so far available atomic update commands:

- inc
  increments a value in the backend by a assigned value
- dec
  decrements a value in the backend by a assigned value
- push
  appends an assigned value to an array in the backend
- pushAll
  appends values of an array to an array in the backend
- pull
  removes an value of an array in the backend
- pullAll
  removes values of an array in the backend
- addToSet
  appends an unique value to an array in the backend.

5.3 Security

The first impression might be that Enginio is unsecure. Using Enginio in C++ there are no cryptographic functions called and the REST API sends simple HTTP-Requests with user data and password in plain text. However, actually Enginio is secure because it always uses the Secure Socket Layer (SSL). For the use of the Enginio API Open SSL needs to be installed because it's used by the API. Every HTTP-Request sent using the REST API utilizes HTTPS-Protocol

6 Conclusion and Prospect

This paper described the basic structure of BaaS and its vague distinction to PaaS and SaaS. After showing other BaaS offerings it introduced the Enginio backend and how its APIs can be used in frontend applications. Unfortunately the Enginio backend is still in the beta version so not all planned functionality is available at the moment. Concerning this matter it would have been nice to also provide a section about the backend actions and their benefit.
In the future BaaS services will be used and become more commonly than they are right now. This assumption is based on the steadily rising amount of mobile devices and mobile applications. However every big company in the IT business (e.g. Facebook) won't use third party offerings and also needs their own databases, accordingly most BaaS offerings will essentialy be used by smaller companies or private persons.
The Backend as a Service structure and offerings are an easy and fast way to manage data in the backend. In this regard the BaaS offerings meet exactly what they intended to meet—the needs of mobile application developers.

References

[1] Anyprecence- the revolutionary enterprise backend-as-a-service platform. Effective: 31.10.2013. [Online]. Available: http://www.anypresence.com/architecture.php
[2] Apiomat. Effective: 13.11.2013. [Online]. Available: apiomat.org
[3] App42 cloud api - backend as a service. Effective 13.11.2013. [Online]. Available: api.shephertz.com
[4] Baasbox- open source backend as a service. Effective: 13.11.2013. [Online]. Available: baasbox.com
[5] Cloudmine- rethink mobile apps. Effective 13.11.2013. [Online]. Available: Cloudemine.me
[6] Getting started with enginio and c++. Effective: 02.12.2013. [Online]. Available: https://engin.io/documentation/qt/enginio-cpp.html
[7] Getting started with qt. Effective: 26.11.2013. [Online]. Available: https://engin.io/documentation/qt/enginio-getting-started.html
[8] Parse analytics. Effective: 13.11.2013. [Online]. Available: Parse.com
[9] H. M. G. . (ane) heise.de. Digia uebernimmt nokias restliche qt-aktivitaeten. Effective: 28.10.2013. [Online]. Available: http://www.heise.de/developer/meldung/Digia-uebernimmt-Nokias-restliche-Qt-Aktivitaeten-1663660.html
[10] anypercence. All features. Effective: 31.10.2013. [Online]. Available: http://www.anypresence.com/all-features.php
[11] apiomat. Server code quickstart. Effective: 31.10.2013. [Online]. Available: http://www.apiomat.com/docs/java-script-code-injection-tutorial/
[12] Digia. Effective: 13.11.2013. [Online]. Available: engin.io
[13] digia. Documentation of the rest api. Effective 05.12.2013. [Online]. Available: https://engin.io/documentation/rest
[14] ——. Engin.io/features. Effective: 23.10.2013. [Online]. Available: https://engin.io/features
[15] ——. Engin.io/pricing. Effective: 23.10.2013. [Online]. Available: https://engin.io/pricing
[16] Programming language support & language bindings. Digia. Effective: 21.12.2013. [Online]. Available: http://qt-project.org/wiki/Category:LanguageBindings
[17] digia. Qt quick tutorial. Effective 08.12.2013. [Online]. Available: http://qt-project.org/wiki/Qt_Quick_Tutorial
[18] ——. Quick history. Effective: 28.10.2013. [Online]. Available: http://qt.digia.com/About-us/
[19] L. P. Geoffrey Raines, "Platform as a service: A 2010 marketplace analysis," *MITRE*. [Online]. Available: http://www.mitre.org/sites/default/files/pdf/cloud_platform_service_paas.pdf
[20] W. J. T. Grance, "Guidelines on security and privacy in public cloud computing," *Information Technology Laboratory National Institute of Standards and Technology*, p. 52, January 2011. [Online]. Available: https://cloudsecurityalliance.org/wp-content/uploads/2011/07/NIST-Draft-SP-800-144_cloud-computing.pdf
[21] S. Kächele, J. Domaschka, and F. J. Hauck, "Cosca: An easy-to-use component-based paas cloud system for common applications," in *Proceedings of the First International Workshop on Cloud Computing Platforms*, ser. CloudCP '11. New York, NY, USA: ACM, 2011, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org/10.1145/1967422.1967426

[22] S. Kächele, C. Spann, F. J. Hauck, and J. Domaschka, "Beyond IaaS and PaaS: An extended cloud taxonomy for computation, storage and networking," December 2013.
[23] N. P. Release. (2008, 01) Nokia to acquire trolltech to accelerate software strategy. Effective: 28.10.2013. [Online]. Available: http://press.nokia.com/2008/01/28/nokia-to-acquire-trolltech-to-accelerate-software-strategy/
[24] SiliconIndia. (2012, August) Understanding the basics of backend as a service (baas). Effective 31.10.2013. [Online]. Available: http://mobile.siliconindia.com/news/Understanding-the-Basics-of-Backend-as-a-Service-BaaS-nid-126045.html
[25] J. B. M. Summerfield, *C++ GUI Programming with Qt 4*. Prentice Hall, June 21, 2006. [Online]. Available: http://my.safaribooksonline.com/book/programming/cplusplus/0131872494/a-brief-history-of-qt/pref04