Syncing Mobile Applications with Cloud Storage Services

Paul POCATILU, Cătălin BOJA, Cristian CIUREA
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ppaul@ase.ro, catalin.boja@ie.ase.ro, cristian.ciurea@ie.ase.ro

Cloud data storage is an option available almost on any mobile platform. Nowadays, there are multiple solutions for syncing data in mobile applications. The aim of the paper is to analyze mobile application developers' possibilities for syncing content using major free cloud storage providers. The paper describes the cloud computing in mobile context and highlights cloud providers APIs. Experimental results are analyzed in order to identify the best cloud storage solution for syncing mobile applications, depending on the operating system on which they are implemented.
Keywords: *Cloud Computing, Mobile Application, Data Synchronization, Application Programming Interface, REST, OAuth*

1 Introduction

We are living in the era of agile and always-available data storage [1], where it is very important to have instant and permanently access to the data, personal and private, with which we are operating at work or at home. The development of mobile technologies and the spectacular growth of mobile devices users created this opportunity to quickly read our emails, to view our documents from shared folders, to access all the data saved in the cloud directly from the personal smart-phone or tablet.

The mobile devices that we are taking with us every day represents mobile clients for our cloud storage subscriptions that we have to main providers, such as Dropbox, SkyDrive, Google Drive, Box, and so on. It is very important to have the same or a similar user experience on each mobile device, independently by the operating system.

The choice of a certain cloud storage provider (CSP) depends on the facilities provided, the user experience and the storage amount space that he offers. Some mobile operators provides also cloud storage solutions, such as Orange Cloud, which allow to store your digital content, secure in the cloud, available anytime and anywhere [8].

Figure 1 presents the storage and backup settings for iCloud solution, which is integrated in every iOS operating system. Apple provides by default a 5 GB free storage plan for every iOS user.

Fig. 1. Storage and backup settings for iCloud

The paper is structured in five sections, as follows. The section *Cloud Computing in Mobile Context* describes the main types of cloud computing facilities and their advantages when they are used in mobile environments.

Section *Comparative Analysis* analyzes experimental results and findings for main cloud providers on the market, such as Dropbox, SkyDrive and Box, in the mobile context.

Section *Cloud providers API* presents all related information to use cloud providers

DOI: 10.12948/issn14531305/17.2.2013.08

APIs in order to develop mobile applications, on different operating systems, which can store their data in the cloud.
The paper ends with *Conclusion and future work* section that summarize important research results of this analysis and identifies future development possibilities.

2 Cloud Computing in Mobile Context

Because many hardware and software manufacturers have invested a lot in cloud computing solutions, the evolution of public and private cloud has increased in terms of users, security, infrastructure and data storage, [14]. In few years we will assist at the moment when a user will go at work or at home with the same tablet that will be docked in a keyboard and will access all the data from the cloud. The desktop computers will disappear slowly and will be replaced with simple monitors or mobile tablets that will use the desktop virtualization technology. When data storage in the cloud will be cheaper, companies will replace all the hardware equipment with these simple monitors in order to use platform as a service (PaaS), software as a service (SaaS), storage as a service (STaaS), security as a service (SECaaS), data as a service (DaaS), database as a service (DBaaS) or test environment as a service (TEaaS) [2].
Figure 2 below presents the cloud computing architecture integrating all related cloud solutions.

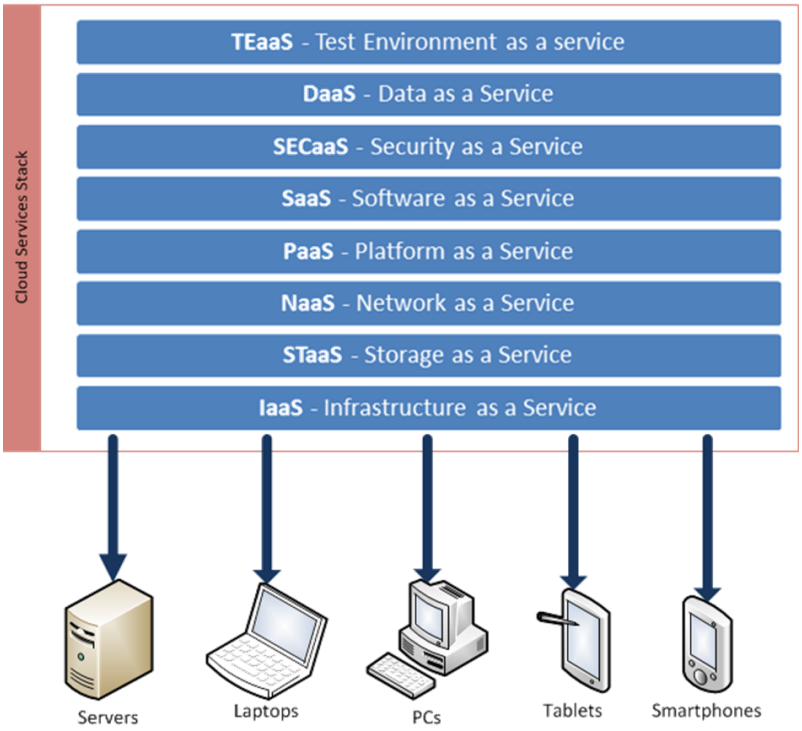

Fig. 2. Cloud services

In [1] is presented a set of challenging storage issues for researchers and engineers. One of these problems is how the storage infrastructure is ensured to be scalable, efficient, and reliable, without any access disruptions, even for upgrades and maintenance periods.
It is very important to have all our data available on the personal smart-phone, but it is crucial to ensure their security and their integrity. Imagine the situation when a user lost the smart-phone or someone stole it. If that device was connected to all the shared folders available in the cloud, the user can say goodbye to his privacy and maybe to his career. Taking these hypotheses into consideration, it is obviously that we must ensure high data security and integrity to all data that can be accessed from different devices connected in the cloud.

DOI: 10.12948/issn14531305/17.2.2013.08

Hardware and software producers for mobile devices have implemented intelligent solutions to solve these issues, such as:

- the "Find my phone" facility, which allows to localize a lost or stolen smart-phone on the map;
- the password to access the smart-phone when unlocking the screen;
- the backup facilities of personal data in the cloud; if we consider Apple, they have the iCloud solution, if we speak about Android devices, Google allows to store all the personal data, and also the Windows Phone devices allows to back up the data in Microsoft cloud solutions (SkyDrive).

Figure 3 presents the "Find My Phone" tool accessed from Windows Phone user webpage.

Fig. 3. The "Find My Phone" tool accessed from Windows Phone

In both cases, iOS and Windows Phone, a user can locate his device on the map and can send different commands to lock it or to erase all the data stored on it. An important disadvantage of mobile devices is related to their portability that can represent a vulnerability when the user loses the mobile device. It is very important to ensure data security on each mobile solution that can represent an entry point for the entire data storage account of a user that save his data to the cloud.

Figure 5 presents the user interface of Google Drive mobile application installed on an iPhone device. The user can acces his files and documents by categories and also he can save them offline to reduce data traffic and access them even when he does not have an internet data connection.

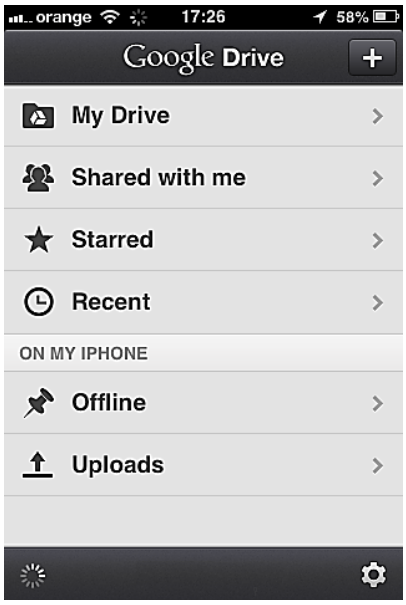

Fig. 4. Google Drive interface on iOS

The cloud serves not only for data storage, but also for testing an application on multiple devices. There are many services that enable developers to test their desktop or mobile applications on multiple real devices through a web interface [4].

3 Comparative Analysis of Cloud Storage

Beginning with the 1997 Dropbox start-up, [19], new storage cloud services [16] have been made available for both business and public users. These services are offered by independent providers for whom this is the main business or by large companies that are adding this new service to their portfolio, like Apple, Amazon, Microsoft and others.

As a business model, cloud storage has proved a growing success [15], Figure 5, as it offers many advantages like:

- solution for disaster recovery and data backup;
- centralized data management;
- data storage costs saving;
- virtualized storage resources;
- collaborative working and user shared resources;
- scalability;
- business flexibility;
- synchronization over different devices.

DOI: 10.12948/issn14531305/17.2.2013.08

Disadvantages and user concerns for public cloud data storage:

- Security, privacy and ownership; recent concerns are highlighted by Bring Your Own Device (BYOD), [13] companies policies as is opens new security threats based on users unprotected and uncontrolled devices that are integrated in secure environments;
- latency over WANs;
- little or none data control regarding; how and where data is stored;
- future performance issue.

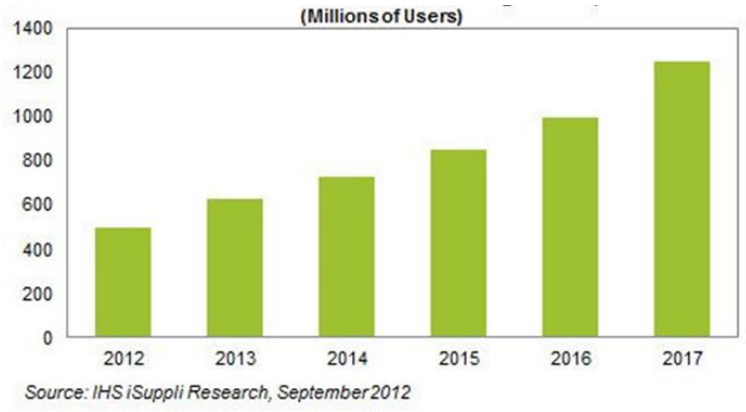

Fig. 5. Worldwide forecast regarding cloud storage subscription 2012-2017. Source [15]

A comparative analysis of cloud storage services must take into consideration a set of measurable criteria [21]. For this research we considered that in terms of syncing efficiency, the next criteria are considered important for the comparison:

- **availability** as the number of different mobile platforms that have a native or independent client sued to access the storage. The iCloud is an IOS native application and it allows only Apple mobile clients to access the cloud.
- **content type diversity** as the number of file types allowed to be stored in the cloud. For example the Amazon Cloud Drive mobile client allows only the upload of photos or music files.
- **ease of use** in terms of provided functionalities; these allows users to manage their content by syncing multiple local folders, collaborating with other users, tracking and recovering file versions;
- **security** functions used by users to password protect files, encrypt files, private and public share of files**;**

Table 1 below presents a comparative analysis of cloud storage solutions on different operating systems.

Table 1. Public cloud storage solutions on different platforms. Source [18].

	Mobile OS				
CSP	**Android**	**Windows Phone**	**iOS**	**BlackBerry**	**Public API**
Dropbox	yes	no*	yes	yes	yes
SkyDrive	yes	yes	yes	no	yes
Box	yes	yes	yes	no	yes
Ubuntu One	yes	no	yes	no	yes
Google Drive	yes	no	no	no	yes
iCloud	no	no	yes	no	yes
Sugarsync	yes	yes	yes	yes	yes
Spideroak	yes	no	yes	no	yes

DOI: 10.12948/issn14531305/17.2.2013.08

As seen from Table 1, the Dropbox solution is not implemented as a native application on Windows Phone platform. Third-party developers created a Dropbox client application that can run on Windows Phone operating system.

Analysis of specific target users groups can describe other usage patterns. A Strategy Analytics Report [21] done on cloud media usage has highlighted a significant link between media providers that offer also integrated cloud storage, Figure 6.

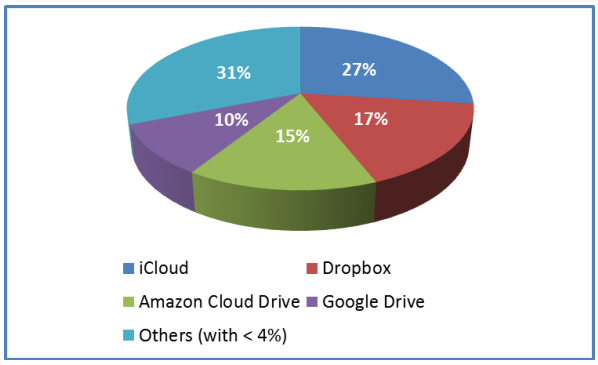

Fig. 6. Market share of cloud storage providers for media content. Source [22]

In this case the media provider integrates in the primary service a secondary cloud storage service and thus having for this segment a greater market share. Taking into account criteria defined for this research, the same cloud storage provider, Apple with its iCloud, doesn't qualify as it doesn't provide a cross platform public API.

From a business perspective, Nasuni [17] has conducted a cloud storage survey based on:

- functionality;
- price;
- performance based on a high level of writes, reads and deletes operations for files that varies in size from 1 KB to 1GB, on data availability and also scalability.

The survey has testes the service provided by five CSPs: Amazon S3, Microsoft Azure Blob Storage, Google Cloud Storage, HP Cloud Object Storage, and Rackspace Cloud Files.

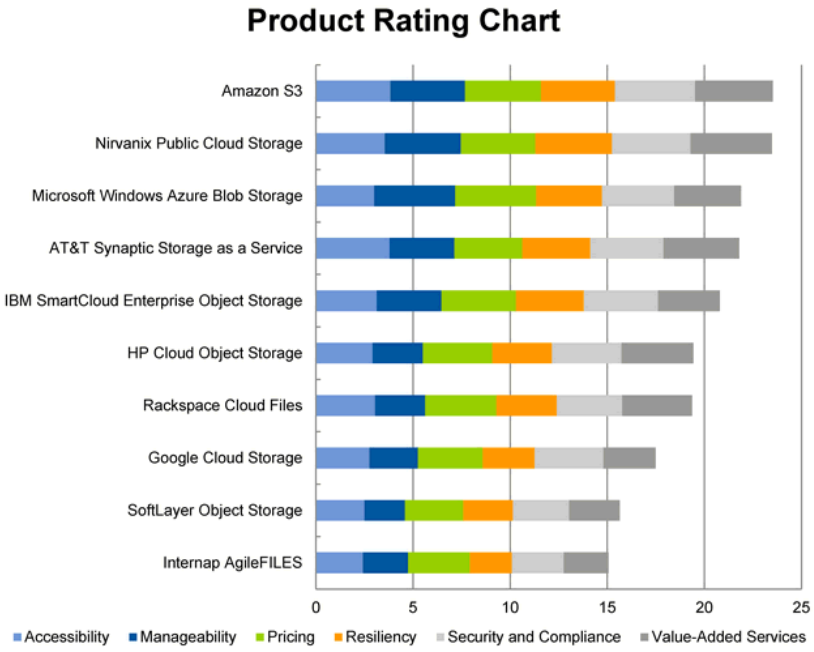

Fig. 7. Gartner public CSP's rating based on six criteria. Source [12]

A market research done by Gartner, [12], has taken into account CSPs that:

- offer APIs for data access and protocols that include Internet APIs, such as REST;
- offer transparent on-demand accessibility and scalability;
- offer definable data security, reliability and availability as part of an SLA;

DOI: 10.12948/issn14531305/17.2.2013.08

- offer pay-as-you-go pricing for capacity and data transfer at a granular level;
- have an established market presence.

They rated ten public CSPs, figure 7, by measuring measured six critical criteria:

- accessibility as the ease of accessing the service and its performance;
- manageability;
- pricing;
- availability and fault tolerance;
- security;
- value-added services.

on four different use cases:

- primary storage;
- backup;
- archive;
- content distribution.

4 Case Study: Cloud Storage APIs Usage in Android Applications

The purpose of this case study is to implement a native application that uses the API provided by several cloud services. The mobile application targeted the Android platform because of its important coverage.

APIs were used from the following cloud services: Dropbox, Google Drive and Box.

The application focuses on the following basic services:

- authentication and authorization;
- file upload;
- file download.

File versioning and revisions are discussed in context.

The file content is taken from an input box, stored in a file, and then uploaded. When downloading files, the remote file content is stored in a local file and then it is stored in a widget. This is a simple usage scenario for such type of dynamically storing data in a cloud.

Each cloud provider requires developers to have an account on their Web site.

All native classes, for all tested cloud services, use REST API. The authorization is based on OAuth 1.0 standard or OAuth 2.0 framework.

For Android, the APIs include activities for simple tasks like authorization and authentication, folder and file selection etc.

4.1 Dropbox API

In order to use the API, the developer is required to register an application that will receive a unique key. The key is used for all API access.

There are three types of APIs that can be used in applications [9]:

- Dropbox Chooser, for file selection in Web pages;
- Sync API, for simple file synchronization;
- Core API, for full access to Dropbox features.

In order to use the Core API, the SDK has to be downloaded. Core API SDK is available at: https://dropbox.com/developers/core/sdk.

The SDK includes several *jar* libraries that will be linked in the Android application's project.

The Dropbox authentication is based on OAuth 1.0 protocol.

The initialization and authentication initiation is presented in Listing 1.

The application key and application's secret are stored in Constants class.

Listing 1. Dropbox authentication process

```
AppKeyPair appKeyPair = new  AppKeyPair(Constants.DROPBOX_APP_KEY,
Constants.DROPBOX_APP_SECRET);
AndroidAuthSession androidAuthSession = new AndroidAuthSession(appKeyPair,
AccessType.DROPBOX);
//the type of dropboxApi is DropboxAPI<AndroidAuthSession>
dropboxApi   = new DropboxAPI<AndroidAuthSession>(androidAuthSession);
//start authentication process
dropboxApi.getSession().startAuthentication(MainActivity.this);
```

The *dropboxApi* member will be used to access all the provided methods for file management. *AndroidAuthenticationSession* class is used to store records on currently logged user and provides methods for authentication using a dedicated Android activity or a web page.

DOI: 10.12948/issn14531305/17.2.2013.08

The method *startAuthentication()* will begin the authentication and authorization process by providing screens for sign in and for application authorization. The user interface depends on the existence of the Dropbox client. The Dropbox client application for Android is not required to be installed on the mobile device.

If the client application is not installed on the device, the user will be redirected to a Web page were the user can authenticate using the email address and the password. Figure 8 (left) presents the *Sign in* page and the acceptance page for the application to access the account (right).

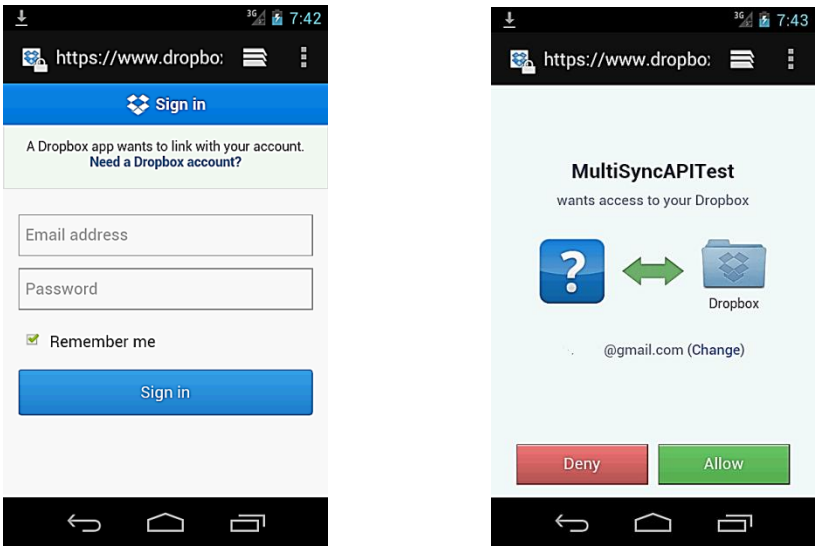

Fig. 8. Dropbox's client authentication window in a Web page

If on the mobile device is installed the Dropbox client, the authentication and acceptance are controlled using native Android activities. In this example, the client was previously authenticated by the Dropbox client application so the sign in activity is not displayed. The application authorization activity is displayed as in Figure 9.

The results of user interaction are controlled by *authenticationSuccessful()* method, that returns true or false, depending on several actions, such as user's selection, correct inputs, network availability, etc. The tokens resulted after authorization are stored by calling *finishAuthentication()* method. The above methods are available from the *AndroidAuthenticationSession* class and will be called within *onResume()* or *onActivityResults()* callbacks from current *Activity* class.

After the user authorizes the application to access the Dropbox account, all the file operation options can be used by it.

Fig. 9. Dropbox's integrated client authentication activity

The application uses classes that extend *AsyncTask* for file handling operations. This is required because these operations are made over a network and these requests have to be implemented in a separate thread [7]. The

DOI: 10.12948/issn14531305/17.2.2013.08

parameters required by the background task are encoded in the *params* variable.

Listing 2 presents the operations used to store a file on the cloud.

In this example, the *putFile()* method takes as non-null parameters the remote file name, the input stream associated to the local file and the content length (in bytes). The remote file is updated to the newest version.

Listing 2. Excerpt from *doInBackgound()* function for file upload using Dropbox API

```
//take data (params[1] and write to a temporary local file (params[0])
BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(new File(params[0])));
bufferedWriter.write(params[1]);
bufferedWriter.close();

//read the content of the local file (params[0]) and write the remote file(params[2])
File file = new File(params[0]);
FileInputStream inputStream = new FileInputStream(file);
dropboxApi.putFile(params[2], inputStream, file.length(), null, null);
```

Retrieving a file from the cloud is done using the remote file name, including the path starting from the root. The application implements this operations as in Listing 3.

Listing 3. Excerpt from *doInBackgound()* function for file download using Dropbox API

```
File localFile = new File(params[0]);
//if the local file doesn't exists it will be created
FileOutputStream outputStream = new FileOutputStream(localFile);

//take the content from remote file (param[1]) to local file
dropboxApi.getFile(params[1], null, outputStream, null);
outputStream.close();
//read the first line of the saved file (as example, to check the content)
BufferedReader bufferedReader = new BufferedReader(new FileReader(localFile));
result = bufferedReader.readLine();
bufferedReader.close();
```

The method *getFile()* receives the remote file name and the output stream associated to the local file were the remote file content will be written.

4.2 Google Drive API

Google Drive SDK allows application integration in browser, access to files, folder and other features of Google Drive from user applications [11].

In order to access these services the developer has to enable the APIs on the *Google APIs Console*, Figure 10. Drive API and Drive SDK services has to be turned on.

In order to use the API for Google Drive, the Android project requires the installation and use of Drive API and Google Play services.

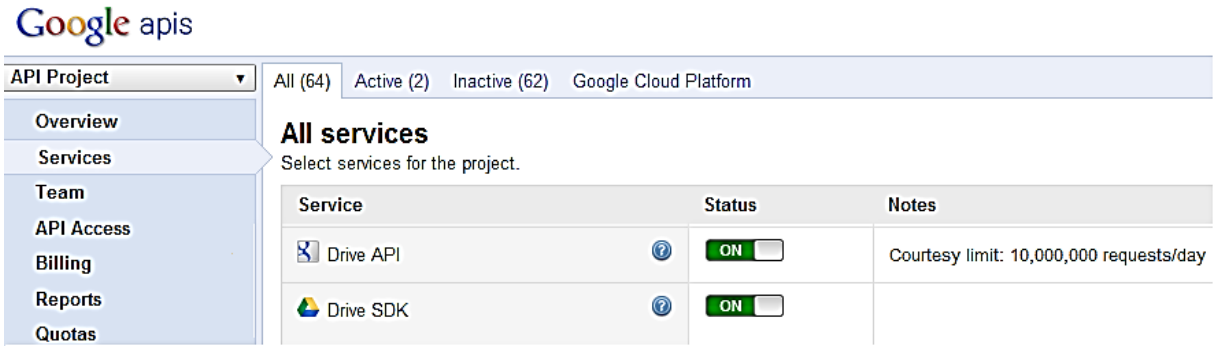

Fig. 10. Drive services activation on Google account

The developer has to register the application with the Google APIs Console. The registration implies the use of the same application certification SHA-1 fingerprint. After the registration process, the developer

DOI: 10.12948/issn14531305/17.2.2013.08

will receive the keys required for authorization using OAuth 2.0.
Listing 4 presents the authentication in authorization sequence.

GoogleAccountCredential class is used for this process. If the authorization is successful, the Drive service is initialized. *Drive* class represent the starting point for interacting with Drive API.

Listing 4. Google Drive authentication

```
//GoogleAccountCredential
credential = GoogleAccountCredential.usingOAuth2(this, DriveScopes.DRIVE);
//acountName is the name of selected Google account
credential.setSelectedAccountName(accountName);
//Drive
driveService = new Drive.Builder(AndroidHttp.newCompatibleTransport(),new GsonFactory(),
               credential).build();
```

Figure 11 presents the confirmation screen used to authorize the application to access Google Drive.

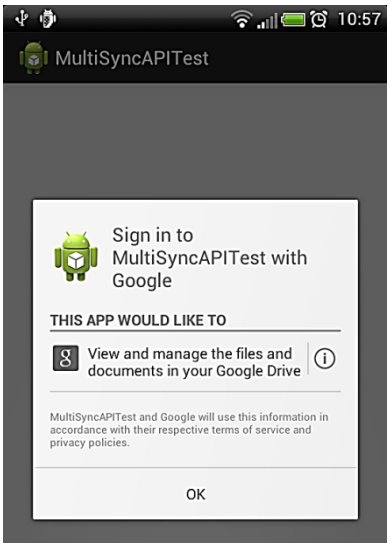

Fig. 11. Drive authorization confirmation

AndroidManifest configuration file requires *android.permission.GET_ACCOUNTS* - permission which is mandatory when user has to select the desired account to be used with Google Drive.
All file operations require the use of MIME types.
The access to *Files* collection is made using *files()* method. *Files* collection includes methods to copy, delete, get and insert files. The *File* class (used in Listing 5 and 6) is defined in *com.google.api.services.drive.model* package and includes file information (metadata) like name (title), creation date, MIME type etc.
Listing 5 represents the sequence used to upload a file using Google Drive. The file metadata are initialized using *com.google.api.services.drive.model.File* class, and the file content is initialized using *java.io.File* class. After the initialization, the file is added to *Files* collection using *execute()* method, which is applied on an *Insert* object created by *insert()* method of the *Files* class.
If the resulting *File* is not null, the operation was successfully.

Listing 5. Excerpt from *doInBackgound()* function for file upload using Google Drive API

```
String mime = "text/plain";

//write data (params[1]) to local file (params[0])
BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(new
java.io.File(params[0])));
bufferedWriter.write(params[1]);
bufferedWriter.close();

Uri fileUri = Uri.fromFile(new java.io.File(params[0]));

// Content initialization
java.io.File file = new java.io.File(fileUri.getPath());
FileContent fileContent = new FileContent(mime, file);

//Metadata initialization
File fileMetadata = new File();
fileMetadata.setTitle(file.getName());
fileMetadata.setMimeType(mime);
```

DOI: 10.12948/issn14531305/17.2.2013.08

```
File resFile = driveService.files().insert(fileMetadata, fileContent).execute();
```

In Listing 6 it is presented the sequence used to download a file from a Google Drive account. In order to download the file, its URI is required. File information can be obtained by executing a search based on the required parameters. In this example, the file name is used as search criteria. This is done by setting the query as *title='filename'*.

Listing 6. Excerpt from *doInBackgound()* function for file download using Google Drive API

```
// get the file by the name; set the query strig
Files.List request = driveService.files().list().setQ("title='" + params[0] + "'");
FileList files = request.execute();

if (files != null)
{
        //get the first file info
        File file = files.getItems().get(0);
        //check the url
        if (file.getDownloadUrl() != null && file.getDownloadUrl().length() > 0)
        {
                //get file content
                HttpResponse resp = driveService.getRequestFactory().buildGetRequest(
                                        new GenericUrl(file.getDownloadUrl())).execute();

                BufferedReader bufferedReader = new BufferedReader(
                                new InputStreamReader(resp.getContent()));
                //read the first line as example
                content = bufferedReader.readLine();
}
```

File downloading is done using a HTTP request based on file's URI. The file content is read using the response's input stream.

4.3 Box API

Box provides several SDKs as open source libraries [10]. The current version of Box for Android is REST API(V2). Box API uses OAuth 2.0 authentication framework.

As on other platforms, developers need to register theirs applications in order to use the API. They will receive the API key. The OAuth 2.0 requires the client's secret key that is generated for each application.

In order to use the Box APIs in a project, an Android library project is provided with full source code.

The library includes Android activities for authentication and for file and folder selection.

Listing 7. Box authentication and authorization

```
Intent intent = OAuthActivity.createOAuthActivityIntent(this, Constants.BOX_CLIENT_ID,
Constants.BOX_CLIENT_SECRET);
this.startActivityForResult(intent, Constants.REQ_CODE_BOX);
...
//BoxAndroidClient
boxClient = data.getParcelableExtra(OAuthActivity.BOX_CLIENT);
```

The user authentication and application authorization activities are presented in Fig. 12. After the user authorizes the application, the API can be used to access the files. An instance of *BoxAndroidClient* is initialized after the authorization.

File operations are performed using *BoxFilesManager* class. An instance of this class can be obtained calling the *getFilesManager()* method from the *BoxAndroidClient* class.

Uploading a file requires the id of the parent folder. The root folder's id is 0. The files and folders ids can be obtained using dedicated pickers activities (*FilePickerActivity* or *FolderPickerActivity*) or by searching the item (folder or file) by its name using *BoxSearchManager* class.

DOI: 10.12948/issn14531305/17.2.2013.08

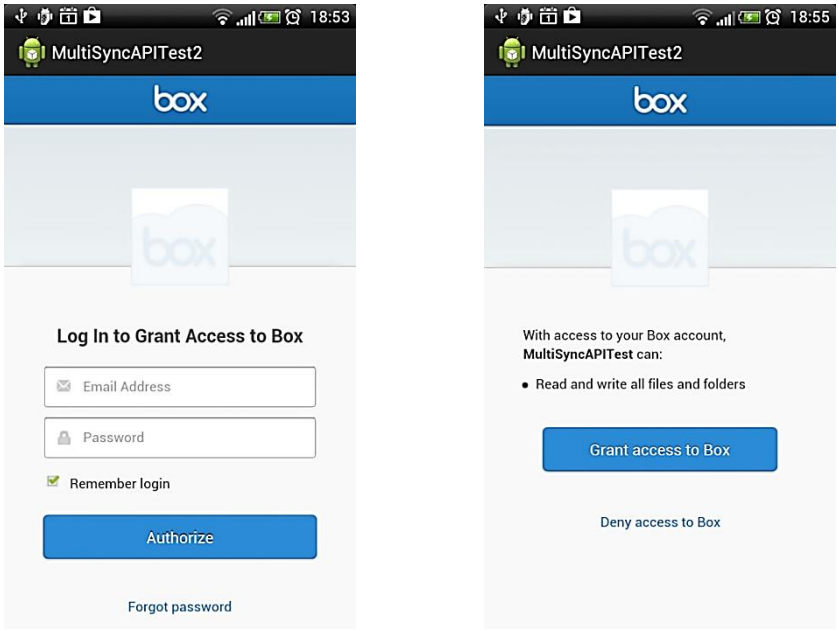

Fig. 12. Box authentication and authorization screens

Listing 8 presents the sequence used to upload a file in the root folder of the Box account. The folder id is initialized in code. The method *uploadFile()* is called to upload the new file. If the file exists and needs to be updated, *uploadNewVersion()* method will be used. This methods requires the id of the existing file.

Listing 8. Excerpt from *doInBackgound()* function for file upload using Box API

```
File localFile = new File(params[0]);

//take data (params[1] and write to a temporary local file (params[0])
BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(localFile));
bufferedWriter.write(params[1]);
bufferedWriter.close();

//root folder
String folderId = "0";
BoxFileUploadRequestObject boxFileUploadRequestObject =
       BoxFileUploadRequestObject.uploadFileRequestObject(folderId, params[2], localFile);

//upload a new file
boxClient.getFilesManager().uploadFile(boxFileUploadRequestObject);
```

Listing 9 presents the code used to download a file using Box API. The method *downloadFile()* requires the id of the file that needs to be downloaded. In this example, the id is obtained by using *search()* method from the *BoxSearchManager* class. Another option is to use dedicated Android picker activities for files and folders.

Listing 9. Excerpt from *doInBackgound()* function for file download using Box API

```
BoxDefaultRequestObject defaultRequest = new BoxDefaultRequestObject();
//search the remote file by name in the default folder
BoxCollection coll = boxClient.getSearchManager().search(params[1], defaultRequest);

if (coll.getEntries().size() != 0)
{
       File localFile = new File(params[0]);
       //get the id of the remote file and store locally the remote file
       boxClient.getFilesManager().downloadFile(coll.getEntries().get(0).getId(), localFile,
null, null);

       //read the first line of the saved file (as example, to check the content)
```

DOI: 10.12948/issn14531305/17.2.2013.08

```
        BufferedReader bufferedReader = new BufferedReader(new FileReader(localFile));
        result = bufferedReader.readLine();
        bufferedReader.close();
}
```

In this example the file used for synchronization is stored in the root folder.

5 Conclusion and future work

Almost all cloud services provides free APIs for developers. As it can be seen from the examples, the code required to use the APIs is intuitive, easy to use and it generally follows the same pattern.

All platforms include an authentication and authorization phase that uses a Web based access or a dedicated Android activity. The user is required to authorize the application. The authorization tokens can be stored so that further use of the application does not require user interaction at this level.

When using smart solutions for syncing mobile applications to the cloud, users save time and money for syncing their files and documents, which represent an important thing in a business process.

A very delicate aspect of this anytime and anywhere advantage that is offered by these solutions is related to data security and users must agree that his files and documents can be automatically scanned by cloud storage providers in order to extract some sensitive information.

References

[1] S. Nagarajan, *Era of Agile and Always-Available Data Storage*, Computer Now, March 2013, Available at: http://www.computer.org/portal/web/computingnow/archive/march2013?lf1=3634 08692f248216093269c6779825

[2] Wikipedia, Cloud computing, Available at: http://en.wikipedia.org/

[3] Android Developers, Syncing to the Cloud, Available at: http://developer.android.com/training/cloudsync/index.html

[4] Smashing Magazine, Getting to Know the Android Platform: Building, Testing and Distributing Apps, Available at: http://mobile.smashingmagazine.com/2012/06/01/getting-to-know-android/

[5] How to Sync App Data across Android Devices, Available at: http://android.appstorm.net/how-to/synchronization/how-to-sync-app-data-across-android-devices-2/

[6] 12 Awesome Android Apps for Plugging in to the Cloud, Available at: http://android.appstorm.net/roundups/12-awesome-android-apps-for-plugging-in-to-the-cloud/

[7] P. Pocatilu, *Programarea dispozitivelor mobile*, ASE Publishing House, Bucharest

[8] Orange Cloud, Available at: https://cloud.orange.ro

[9] Developers – Dropbox, Available at: https://www.dropbox.com/developers

[10] Box Platform Developer Documentation, Available at: https://developers.box.com/sdks/

[11] Google Drive SDK – Google Developers, Available at: https://developers.google.com/drive/

[12] Gene Ruth, Arun Chandrasekaran, Critical Capabilities for Public Cloud Storage Services, Gartner Report, January 2013, Available at http://www.gartner.com/technology/reprints.do?id=1-1D9C6ZM&ct=121216&st=sg

[13] Lucas Mearian, Mobile devices bring cloud storage -- and security risks -- to work, ComputerWorld, June 2012, available online at http://www.computerworld.com

[14] Han Qi, Abdullah Gani, "Research on Mobile Cloud Computing: Review, Trend and Perspectives," *Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, 2012, Bangkok, pp. 195 – 202, ISBN 978-1-4673-0733-8

[15] Jagdish Rebello, "Subscriptions to Cloud Storage Services to Reach Half-Billion Level This Year," *IHS iSuppli Market Research*, September 2012, Available at: http://www.isuppli.com

DOI: 10.12948/issn14531305/17.2.2013.08

[16] Wikipedia, Comparison of file hosting services, 2013, available online at http://en.wikipedia.org/wiki/Comparison_of_file_hosting_services
[17] Nasuni, *The State of Cloud Storage 2013 Industry Report, A Benchmark Comparison of Performance, Availability and Scalability*, Available at: http://www6.nasuni.com/rs/nasuni/images/2013_Nasuni_CSP_Report.pdf
[18] E. Hamburger, "Google Drive vs. Dropbox, SkyDrive, SugarSync, and others: a cloud sync storage face-off," *The Verge*, April 2012, Available at: http://www.theverge.com/
[19] V. Barret, *Dropbox Hits 100 Million Users Says Drew Houston*, November 2012, Available at: http://www.forbes.com
[20] M. Endler, *Apple, Dropbox Lead Cloud Storage Market*, Available at: http://www.informationweek.com
[21] M. Popa, "Characteristics of the Audit Process for ICT Mobile System," *Proceedings of the Tenth International Conference on Informatics in Economy – Education, Research & Business Technologies*, Academy of Economic Studies, Bucharest, 05 – 07 May 2011, ASE Publishing House, Bucharest, ISSN 2247-1480, ISSN-L 2247-1480.
[22] J. Fingas, *Strategy Analytics: iCloud, Dropbox and Amazon top cloud media in the US*, Available at: http://www.engadget.com/

Paul POCATILU graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 1998. He achieved the PhD in Economics in 2003 with thesis on Software Testing Cost Assessment Models. He has published as author and co-author over 45 articles in journals and over 40 articles on national and international conferences. He is author and co-author of 10 books, (Mobile Devices Programming and Software Testing Costs are two of them). He is associate professor in the Department of Economic Informatics of the Academy of Economic Studies, Bucharest. His current research areas are software testing, software quality, project management, and mobile application development.

Catalin BOJA is associate professor at the Economic Informatics and Cybernetics Department at the Academy of Economic Studies in Bucharest, Romania. In June 2004 he has graduated the Faculty of Cybernetics, Statistics and Economic Informatics at the Academy of Economic Studies in Bucharest. He is a team member in various undergoing university research projects where he applied most of his project management knowledge. His work currently focuses on the analysis of mobile computing, information security and cryptography. He is currently holding a PhD degree on software optimization and on improvement of software applications performance.

Cristian CIUREA has a background in computer science and is interested in collaborative systems related issues. He has graduated the Faculty of Economic Cybernetics, Statistics and Informatics from the Bucharest University of Economic Studies in 2007. He has a master in Informatics Project Management (2010) and a PhD in Economic Informatics (2011) from the Bucharest University of Economic Studies. Other fields of interest include software metrics, data structures, object oriented programming in C++, windows applications programming in C# and mobile devices programming in Java.

DOI: 10.12948/issn14531305/17.2.2013.08