Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

SOFTWARE DE PLANIFICACIÓ DE CLASSES (LESSON PLANNER)

Rubén Boada Navarrete

Director: José Bosch Estrada
Realitzat a: Departament d'Electrònica.
UB

Barcelona, 20 de juny de 2013

1. Introducció i antecedents

1.1.1- Introducció

El desenvolupament d'aplicacions per a plataformes mòbils *Android* és un dels mercats amb més creixement actual, i suposa una oportunitat assequible per poder arribar al màxim nombre de persones amb el disseny, desenvolupament i comercialització de software informàtic. Podem partir d'una idea innovadora o analitzar el mercat actual i intentar millorar alguna idea, ampliar-la o agrupar-ne un conjunt per intentar millorar les propostes realitzades fins aleshores per altres persones o empreses.

Aquest últim cas serà el punt de partida del nostre projecte, desenvolupar un software agrupant les funcionalitats ja implementades en altres aplicacions i alhora afegir-hi alguna d'altra per adaptar-lo a les nostres necessitats.

En el nostre cas, volem desenvolupar un software per a PC i per a tablet *Android* orientat bàsicament al personal docent de col·legis, instituts i universitats per tal que puguin organitzar els seus cursos. El requisits principals són que el professor pugui gestionar els seus cursos i els seus alumnes, definir horaris i adjuntar els apunts que es donaran a classe. Per tal d'analitzar quines funcionalitats implementarem, estudiarem quina és l'oferta existent, i analitzarem possibles ampliacions.

1.1.2- Introduction

The development of applications for Android platforms is one of the fastest-growing markets today, and is an affordable opportunity to reach to many people with the design, development and marketing of computer software. We can start with an innovative idea or analyze the current market and try to improve an idea, expand it or group a set of them to try to improve the proposals made by other persons or companies.

The latter will be the starting point of our project to develop a software, clustering the features already implemented in other applications and adding some others to suit our needs.

In our case, we want to develop software for PC and Android tablet aimed primarily at teachers of schools, colleges and universities so that they can organize their courses. The main requirements are that the teacher can manage their courses and their students, set schedules and attach the notes that will be taught in a class. To analyze which features will we implement, we will study what is on offer and discuss possible extensions.

1.2.1- Estudi de mercat

Amb aquestes premisses, estudiarem quina és l'oferta del mercat en aquest tipus d'aplicació i que podem afegir i millorar:

Aplicacions multi-plataforma

Hellmansoft Planbook

Lloc web: http://www.hellmansoft.com/

Aplicació per a PC *Windows*, *Mac* i *Ipad*. Permet crear cursos, assignatures i adjuntar apunts i links a pàgines web per a cada classe. La interfície d'usuari ens permet tenir una visió global del curs i un seguiment detallat setmana a setmana. Permet fer *back-ups* de la informació i sincronitzar la informació entre les diferents plataformes per a les quals està destinada. L'aspecte i usabilitat de l'aplicació és agradable, i la idea d'una interfície que ens permeti tenir una visió global de la setmana actual resulta atractiva. El servei complet de l'aplicació és de pagament, ja que la versió de prova té un límit de cursos.

L'aplicació no ens permet gestionar cap informació sobre els alumnes dels cursos, només la gestió de les assignatures.

Lesson Planner UK

Lloc web: http://www.lessonplanner.co.uk/

Aplicació per a PC *Windows, Mac* i *Ipad*. Totes les versions d'aquest software són de pagament. A la versió demostrativa podem observar com la configuració de l'aplicació està orientada a la informació detallada de cada classe, però perdem la perspectiva de curs que l'aplicació *Planbook* ens ofereix. La informació que podem emmagatzemar de cada classe és molt detallada, ampliant el ventall de paràmetres que l'usuari por introduir

respecte l'aplicació *Planbook*. No ens permet gestionar els alumnes de les diferents assignatures, i el funcionament s'apropa al clàssic bloc de notes on anotem les tasques que hem de portar a terme, en aquest cas les tasques són les classes.

Aplicacions Tablet

Teacher Notebook

Lloc Web:

https://play.google.com/store/apps/detailsid=com.amstapps.xnotebookdemo&hl=es

Aplicació per a plataformes *Android* que ens permet gestionar estudiants i classes d'un curs. Afegeix un sistema de gestió d'exàmens i notes per a les classes, a més d'un sistema de gestió d'assistència a les classes. No ofereix una visió global del curs, sinó una visió de les assignatures i dels alumnes que tenim en un curs. Permet exportar i importar cursos des de fitxer. La versió completa és de pagament. És l'aplicació més semblant a la nostra idea de projecte, però li manca la idea de calendari global i apunts per classes, a més de versió per a PC.

Teacher Planner Lite

Lloc web:

https://play.google.com/store/apps/details?id=net.cunniman.teacherplannerlite&hl=es

Permet gestionar les classes d'un curs. Afegeix la visió temporal del curs, permetent que l'usuari pugui accedir a les classes d'un dia en concret, o obtenir una visió en quadrícula dels dies de la setmana i les assignatures que ha de desenvolupar cada dia. No permet la gestió d'alumnes, però l'idea de la gestió temporal de les classes és molt semblant a la

que nosaltres volem desenvolupar. No permet la inclusió d'apunts. Permet exportar la informació a altres plataformes com *Gmail*.

Plan de clases para Maestros

Lloc web:

https://play.google.com/store/apps/detailsid=eu.schooltimetable.android.teacher.pro&hl=es

La visió setmanal i els horaris de les classes és el punt fort de l'aplicació. Observem una pantalla principal on es mostra per dies les classes que s'han de realitzar. Permet afegir anotacions a cada classe, però no ens permet tenir una visió global del curs, ja que no podem veure les classes segons data del calendari. No permet la gestió d'alumnes.

De l'observació i testeig d'aquestes aplicacions extraiem diverses conclusions:

- Gairebé cap aplicació reuneix la visió temporal del curs i la gestió de classes, alumnes i apunts.
- Algunes de les aplicacions actuen com a bloc de tasques a realitzar, però no ofereixen la funcionalitat d'agenda temporal del curs.
- Gairebé totes les aplicacions observades reserven la funcionalitat de sincronització entre diferents plataformes per a la versió de pagament.
- La gestió d'alumnes és poc freqüent a les diferents aplicacions testejades, i aquest es un punt al qual nosaltres volem donar importància a la nostra aplicació.

L'aplicació a desenvolupar intentarà basar-se en agrupar els punts forts de les diferents aplicacions observades (visió temporal del curs de l'aplicació multi-plataforma *Planbook* i gestió de l'alumnat de l'aplicació *Teacher Notebook* per a plataforma *Android*) i afegint un sistema de sincronització entre plataformes que no requereixi d'una versió *premium*.

1.2.2- Market research

With these premises, we will study what the market offer of this type of application and what we can add and improve:

Multi-platform applications

Hellmansoft Planbook

Website: **http://www.hellmansoft.com/**

Application for Windows PC, Mac and iPad. Allows you to create courses, subjects and attach notes and links to websites for each class. The user interface allows us to have an overview of the course and a detailed week. Lets make back-ups of information and synchronize the information between different platforms. The appearance and usability of the application is nice, and the idea of an interface that allows us to have an overview of the current week is attractive. The complete application must be paid. The trial version have a limited number of courses.

The application does not allow us to manage any information about students.

Lesson Planner UK

Website: http://www.lessonplanner.co.uk/

Application for Windows PC, Mac and iPad. All versions of this software requires payment. In the demo version you can see the configuration of the application is aimed at detailed information about each class, but we lose the course perspective that the application Planbook offers. The information that can we store in each class is very detailed, extending the range of parameters that the user can introduce regarding the application Planbook. We can't manage students of different subjects, and operation approaches the

classic notebook where we note which tasks we will carry out, in this case the tasks are classes.

Tablet apps

Teacher Notebook

Website:

https://play.google.com/store/apps/detailsid=com.amstapps.xnotebookdemo&hl=es

Application for Android platform that allows us to manage students and classes in a course. Add a management system of exams and qualifications. No offers an overview of the course, but offers an overview of the subjects and students that we have in a course. Allows us to export and import courses from file. The full version requires payment. Application is similar to our project idea but missing the idea of calendar and notes for classes. Doesn't have PC version.

Teacher Planner Lite

Website:

https://play.google.com/store/apps/details?

id=net.cunniman.teacherplannerlite&hl=es

Allows us to manage course classes . Add temporary vision of the course, allowing the user to access the classes on a particular day, or get an overview of the subjects. It does not allow students management, but the idea of temporary classes management is very similar to what we want to develop. Do not allow attach files. Allows to export information to other platforms like Gmail.

Plan de clases para maestros

Website:

https://play.google.com/store/apps/detailsid=eu.schooltimetable.android.teacher.pro&hl=es

The weekly view and schedule of classes is the strong point of the application. We observed a main screen where is showing classes that will we make. Allows you to add notes to each class, but it doesn't gives us an overview of the course. We can not see the classes according to the calendar date. Does not allow students management.

From observation and testing of these applications, we draw several conclusions:

- Almost any application meets the temporal course vision and management classes, students and notes.
- Some applications acts as a block of tasks, but do not provide the functionality of calendar.
- Almost all the applications reserved sync functionality between different platforms for the payment version.
- The management of students is rare into different applications tested, and this is an item which we want to give importance into our application.

The application will attempt to group the strong items of tested applications (temporal vision of course of Planbook application and students management of Teacher Notebook application) and adding a synk operation between platforms that not requires payment version.

2. Definició d'objectius i motivació del problema

2.1- Objectius del projecte

Fet l'anàlisi de l'oferta del mercat per a aplicacions destinades a ajudar a planificar el curs a professors i personal docent, podem definir els nostres objectius en el desenvolupament d'aquest projecte:

- **Desenvolupar un software orientat a la gestió de cursos per a les plataformes Android i PC**
- **Oferir un software de qualitat al mercat de les aplicacions PC**
- **Oferir un software de qualitat al mercat de les aplicacions *Android***
- **Analitzar les mancances del mercat per desenvolupar un software que afegeixi una marca identitària i diferencial de la resta.**
- **Implementar un sistema de sincronització entre les diferents plataformes.**

1.Desenvolupar un software orientat a la gestió de cursos per a les plataformes Android i PC

L'objectiu principal del projecte es desenvolupar un software per a *Android* i PC que ofereixi les següent funcionalitats:

- Gestionar classes i horaris
- Gestionar assignatures
- Gestió dels apunts i documents adjunts a les classes
- Gestió dels alumnes. En aquest cas volem donar la possibilitat d'introduir informació d'alumnes manualment o mitjançant un fitxer *Excel* que contingui la

informació dels alumnes d'una assignatura. Aquest fitxer *Excel* mantindrà el format dels fitxers *Excel* del campus virtual de la UB, facilitant les tasques al professorat.

Per tenir una visió global de les funcionalitats que oferirem, definim el següent diagrama de casos d'ús:

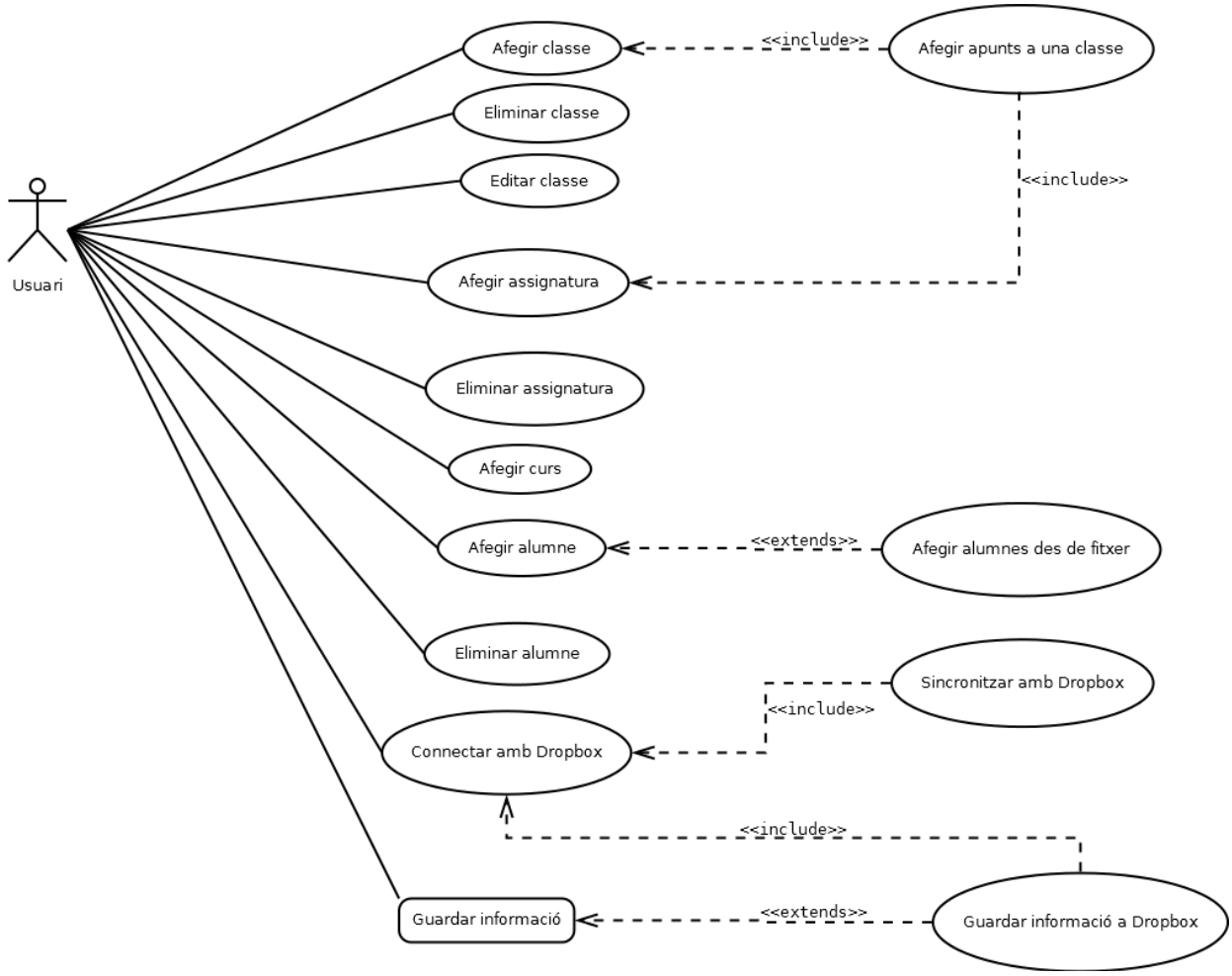

Figura 1. Diagrama de casos d'ús de les nostres aplicacions

2.Oferir un software de qualitat al mercat de les aplicacions PC

Volem aplicar els coneixements adquirits en àmbits del desenvolupament de software com l'anàlisi de requisits de l'aplicació, disseny de les classes i funcionalitats a implementar, programació orientada a objectes, programació orientada a events, etc... Per tal de desenvolupar un software amb garanties.

Volem aplicar conceptes d'usabilitat per tal de desenvolupar una interfície d'usuari que faciliti la interacció amb l'usuari i generi una experiència d'usuari satisfactòria.

Volem que la realització del treball serveixi com a mètode d'exploració i aprenentatge en el camp del desenvolupament del software, ampliant els coneixements adquirits fins aleshores.

3.Oferir un software de qualitat al mercat de les aplicacions *Android*

Volem que el software que desenvolupem sigui de la qualitat apropiada per a una de les plataformes més utilitzades i esteses arreu del món com és *Android*. Per fer-ho, aplicarem els coneixements en desenvolupament de software adquirits fins ara com son l'anàlisi de requisits de l'aplicació i el disseny de les funcionalitats i les classes a implementar.

A més, serà molt important l'anàlisi de la usabilitat per fer una aplicació atractiva, ja que *Android* aporta recursos molt interessants per a la interfície gràfica. Volem fer una aplicació on les diferents parts quedin ben diferenciades i estructurades, amb una fàcil navegació i accés a elles, per tant el disseny espacial dels elements serà clau.

Volem que a més, el procés de desenvolupament serveixi per adquirir coneixements sobre aplicacions *Android* i profunditzar els ja adquirits en aquesta matèria.

4.Analitzar les mancances del mercat per desenvolupar un software que afegeixi una marca identitària i diferencial de la resta.

L'oferta en matèria de software per a les plataformes a les quals enfocarem el nostre treball és bastant àmplia i de gran qualitat, per tant, hem de saber adaptar les nostres idees per crear un software que es desmarqui de l'oferta existent o que afegeixi algun ítem que la faci atractiva. Abans de fer el disseny de la nostra aplicació hem fet un treball

de camp observant que és el que ofereix el mercat i per tant, un dels nostres objectius és afegir alguna peculiaritat al nostre software, En un principi hem observat que cap aplicació gratuïta ofereix totes les característiques que un professor desitja per tenir un control de la seva feina, i que poques aplicacions permeten la sincronització i visualització de dades entre plataforma PC i *Android*, per tant els nostres esforços estaran orientats a satisfer aquesta necessitat.

5.Implementar un sistema de sincronització entre les diferents plataformes.

Dintre dels requisits plantejats en la definició del problema està un requisit bàsic, que suposa la sincronització de les dades de les dues aplicacions. Per fer-ho, hem optat per utilitzar una de les plataformes mes conegudes i utilitzades arreu del mon actualment, *Dropbox. Dropbox* és un servei d'allotjament d'arxius multi-plataforma, que ofereix a l'usuari emmagatzemar i sincronitzar arxius en línia. El seu ús suposa una innovació de cara a les altres aplicacions i alhora un repte i motivació per aprendre a utilitzar els seus recursos dins les nostres aplicacions. Utilitzarem *Dropbox* per a emmagatzemar el fitxer que contindrà la informació compartida per les dues aplicacions, a més de les fotografies dels alumnes i els fitxers associats a les classes del curs. De la decisió d'utilitzar *Dropbox* com a eina sincronitzadora de les nostres dades es crea un dels objectius bàsics d'aquest projecte, que és estudiar i integrar la recent API de *Dropbox* al nostre software.

3. Desenvolupament

3.1- Estudi previ del llenguatge utilitzat

Per al desenvolupament de la nostra aplicació el primer que farem serà estudiar que llenguatge de programació escollirem tenint en compte els objectius definits, quins recursos necessitarem i quin és el nostre coneixement sobre cadascun d'ells.
En els objectius hem definit que volem desenvolupar una aplicació per a PC i per *Android*, i hem de tenir en compte que habitualment les aplicacions *Android* es desenvolupen en llenguatge Java. Per simplificar volem desenvolupar les dues aplicacions amb el mateix llenguatge de programació, això és un primer motiu per triar Java com a llenguatge per tot el projecte. A més, si separem i estructurem bé el codi de les dues aplicacions, podem reutilitzar gairebé tot el codi, simplement reemplaçant el codi de les interfícies gràfiques ja que no utilitzaran les mateixes llibreries.

En el tema de les interfícies gràfiques, Java ens ofereix dues API's de desenvolupament de GUI com són *AWT* i *Swing* per a aplicacions d'escriptori que ens facilita les eines suficients per implementar allò que desitgem per a la versió PC. A més, el nostre domini en el desenvolupament de GUI amb llenguatge Java és més gran que en altres, facilitant així la planificació dels terminis de treball i el repartiment de càrregues de treball en un curt plaç de temps.

Per últim, hem esmentat en la definició dels nostres objectius que un pilar bàsic de la nostra aplicació i de l'aprenentatge d'aquest treball és la interacció de les nostres aplicacions amb l'API de *Dropbox. Dropbox* ofereix diferents API's en funció de les nostres

necessitats, i hem escollit Core API de *Dropbox* perquè ens ofereix les funcionalitats bàsiques que necessitem per al nostre projecte (escriptura i lectura de fitxers de *Dropbox*). *Dropbox* posa a disposició de l'usuari SDK's per a diferents llenguatges de programació per tal d'acoblar les peticions http a l'API de *Dropbox,* i d'entre les oficials estan les de *Java* i *Android*. Així, sabem que amb Java podrem utilitzar les funcionalitats de l'API de *Dropbox* per a la nostra aplicació.

L'entorn de desenvolupament que utilitzarem serà Eclipse, utilitzant el complement *Android Development Tools (ADT)* per a l'aplicació *Android.*

Per tots aquests motius esmentats anteriorment hem escollit Java com a llenguatge de programació del nostre projecte.

3.2- API de Dropbox

Dropbox ha posat a disposició dels desenvolupadors una sèrie d'API's per tal que la integració de *Dropbox* a les aplicacions de tercers sigui molt més assequible. Dintre de les diferents API's, nosaltres hem escollit Core API, perquè és la que ens permet operar amb *Dropbox* a més baix nivell amb operacions de pujada i descàrrega de fitxers.

Com ja hem esmentat anteriorment, *Dropbox* posa a la disposició dels desenvolupadors diferents SDK's per als diferents llenguatges de programació, i en aquest cas nosaltres utilitzarem els de Java i *Android*.

Explicarem pas a pas els punts més importants per a la integració i utilització d'aquesta API dins del nostre projecte:

Primers passos

Dropbox requereix que registrem la nostra aplicació al seu lloc web, indicant el tipus d'API que utilitzarem i els permisos que requerirà la nostra aplicació per a l'accés a les dades de *Dropbox*. En aquest cas tenim dues opcions, restringir a l'usuari l'accés a la carpeta de l'aplicació que es generarà en el seu espai *Dropbox* o permetre a l'aplicació accedir a totes les dades que es trobin a l'espai *Dropbox* de l'usuari. La configuració recomanada és la primera, doncs l'aplicació només podrà accedir a les dades emmagatzemades a la carpeta específica de l'aplicació, i és la que hem escollit per al nostre projecte.

Una vegada confirmem la creació de l'aplicació, *Dropbox* ens ofereix dues claus identificatives que haurem d'utilitzar dins les nostres aplicacions.

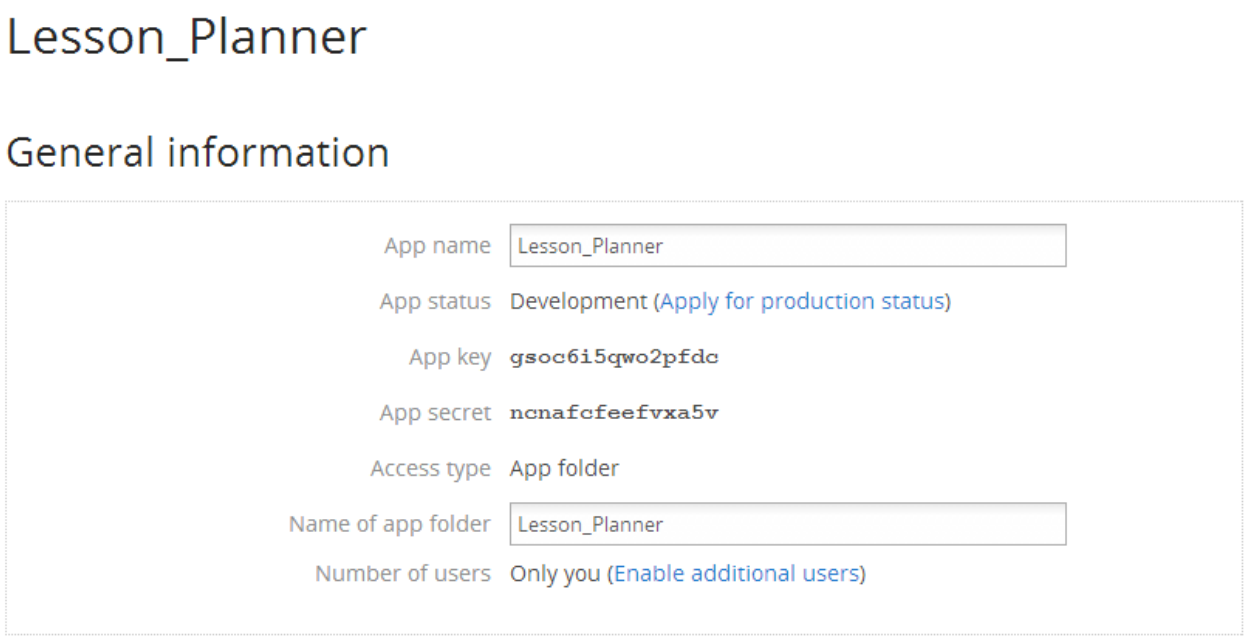

Figura 2. Panell de control al lloc Dropbox Developers

Podem observar al panell de la figura 2 que al crear l'aplicació l'estatus d'aquesta és *Development*, un estat on nosaltres com a desenvolupadors podem utilitzar totes les funcionalitats de Core Api per tal d'implementar la nostra aplicació però no es pot distribuir ni comercialitzar l'aplicació fins que *Dropbox* no validi que el desenvolupador ha seguit la guia d'estil que marca l'empresa en matèria de seguretat i ús d'imatges, tipografia i identificadors de *Dropbox*.

Instal·lació de Core Api

Descarreguem l'SDK d'*Android* i de Java i afegim els JAR al *Build path* del nostre projecte. Una vegada hem afegit totes les llibreries necessàries, hem d'introduir el parell de claus esmentats anteriorment dins dels nostres projectes.

Pel que fa a *Android*, hem d'afegir al fitxer ***AndroidManifest.xml*** el següent codi per tal que *Dropbox* API pugui portar a terme l'autentificació:

```
<activity
      android:name="com.dropbox.client2.android.AuthActivity"
      android:launchMode="singleTask"
      android:configChanges="orientation|keyboard">
      <intent-filter>
      <!-- Change this to be db- followed by your app key -->
            <data android:scheme="db-INSERT_APP_KEY" />
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.BROWSABLE"/>
            <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
</activity>
```

El que hem fet és configurar l'activity de login que es mostra cada vegada que volem donar el nostre permís a *Dropbox* per accedir a les nostres dades. A més, haurem d'especificar que la nostra aplicació requerirà accés a Internet amb el següent codi:

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

Autentificació de l'usuari

L'API de *Dropbox* utilitza *OAuth* v1, un protocol obert per tal de garantir la seguretat en la interacció amb dades protegides com és en aquest cas el sistema d'autenticació.

El primer que hem de fer és definir les claus d'accés al nostra codi, tant al projecte *Android* com al projecte PC:

```
final static private String APP_KEY = "INSERT_APP_KEY";

final static private String APP_SECRET = "INSERT_APP_SECRET";

final static private AccessType ACCESS_TYPE = AccessType.INSERT_APP_ACCESS_TYPE;
```

ACCES_TYPE és la variable que defineix si tenim accés a tota la informació emmagatzemada a *Dropbox* o tan sols a la carpeta específica creada per a la nostra aplicació. En el nostra cas serà *AccessType.APP_FOLDER*.

El següent pas serà instanciar un objecte de tipus *DropboxApi* que serà el que utilitzarem per fer les peticions de pujada i descàrrega de fitxers i crear la sessió pertinent:

```
private DropboxAPI<AndroidAuthSession> mDBApi;

AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);

AndroidAuthSession session = new AndroidAuthSession(appKeys, ACCESS_TYPE);

mDBApi = new DropboxAPI<AndroidAuthSession>(session);
```

En el cas de l'aplicació PC no utilitzarem *AndroidAuthSession* sinó *WebAuthSession:*

```
WebAuthSession session = new WebAuthSession(appKeys, ACCESS_TYPE);
```

Finalment, quan l'usuari decideix que vol connectar-se a *Dropbox*, el que fem és mostrar la pàgina d'autentificació que ens facilita *Dropbox* fent la següent crida:

```
mDBApi.getSession().startAuthentication(MyActivity.this);
```

Si tenim l'aplicació de *Dropbox* instal·lada al nostre terminal ens redirigirà cap a ella, si no, ens obrirà el navegador predeterminat per tal que puguem confirmar o declinar el nostre accés. Si l'usuari accepta l'accés a *Dropbox*, s'adjunta a la sessió creada l'*acces token* de l'usuari en qüestió,que és l'identificador unívoc de l'usuari dins els sistema. Aquest identificador ens permet la persistència de la sessió encara que l'aplicació hagi estat tancada, i per fer-ho utilitzem diferents estratègies en cadascun dels projectes.

Pel que fa a *Android,* utilitzem *SharedPreferences* per emmagatzemar el parell de claus.

El fil d'execució d'un login complet a Android és el següent:

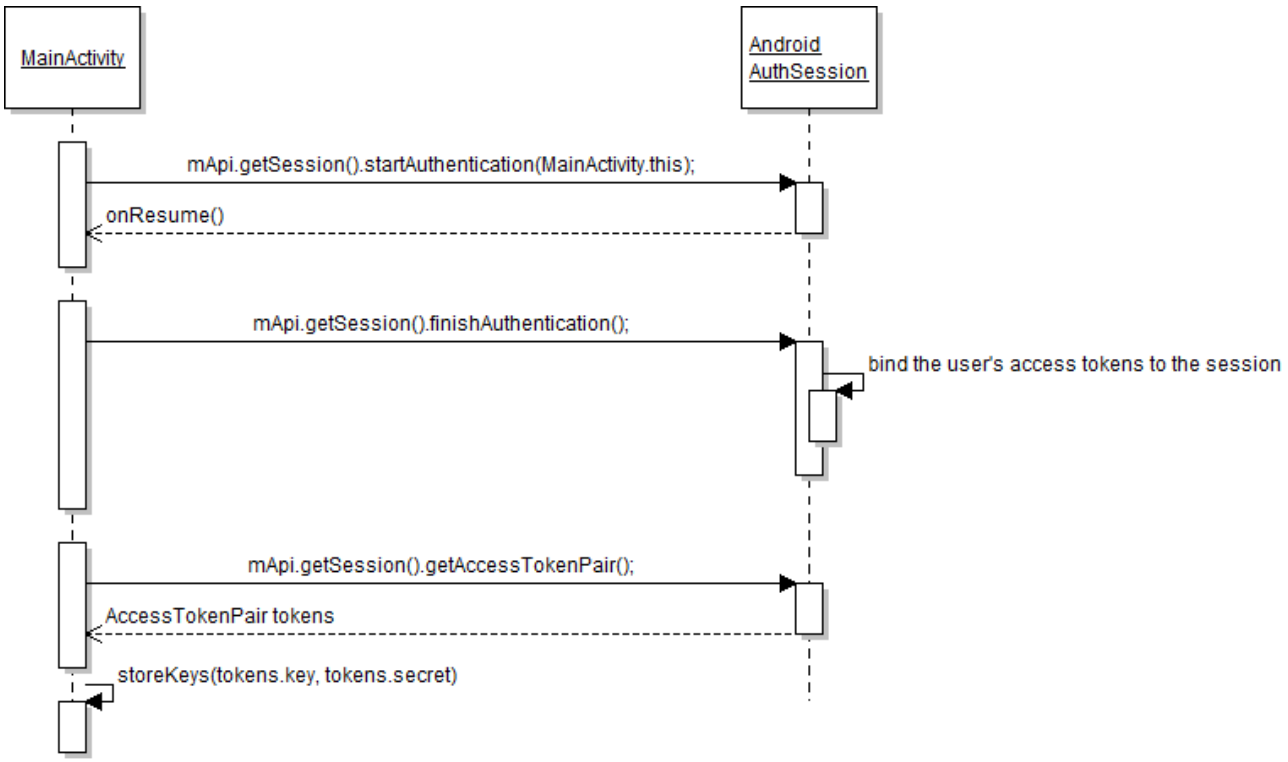

Figura 3. Diagrama d'execució del login de Dropbox

El mateix procés s'executa a l'aplicació PC, però de diferent manera. Amb l'objecte *DropboxApi* obtenim les dades de la sessió abans creada, i obtenim la direcció URL de la pàgina d'autentificació de *Dropbox.* Amb la classe Desktop obrim al navegador la pàgina, i esperem a que l'usuari accepti o declini l'accés, per tal d'emmagatzemar *l'AccesTokenPair.*

A l'aplicació PC no tenim la crida a *mApi.getSession().finishAuthentication()* disponible ni tenim el mètode *onResume()* que gestiona el retorn a l'aplicació des del navegador, per tant, mentre esperem que l'usuari decideixi si confirma o no l'accés, aturem l'execució de l'aplicació mostrant a l'usuari un diàleg de confirmació, per tal de saber si realment ha acabat acceptant l'accés o l'ha declinat. Si l'accepta, a diferencia de l'aplicació *Android* on

finishAuthentication() ja assigna a la sessió actual el parell de claus, hem d'assignar a la sessió el parell de claus generat.

Una de les diferències principals de l'aplicació PC respecte *Android* és l'emmagatzematge del parell de claus, ja que no tenim disponible *SharedPreferences*. El que fem és escriure dins d'un fitxer de text situat a la carpeta local de l'aplicació [3] les dues claus, per tal de recuperar-les quan tornem a entrar a l'aplicació. El fil d'execució d'un login complet a PC és el següent:

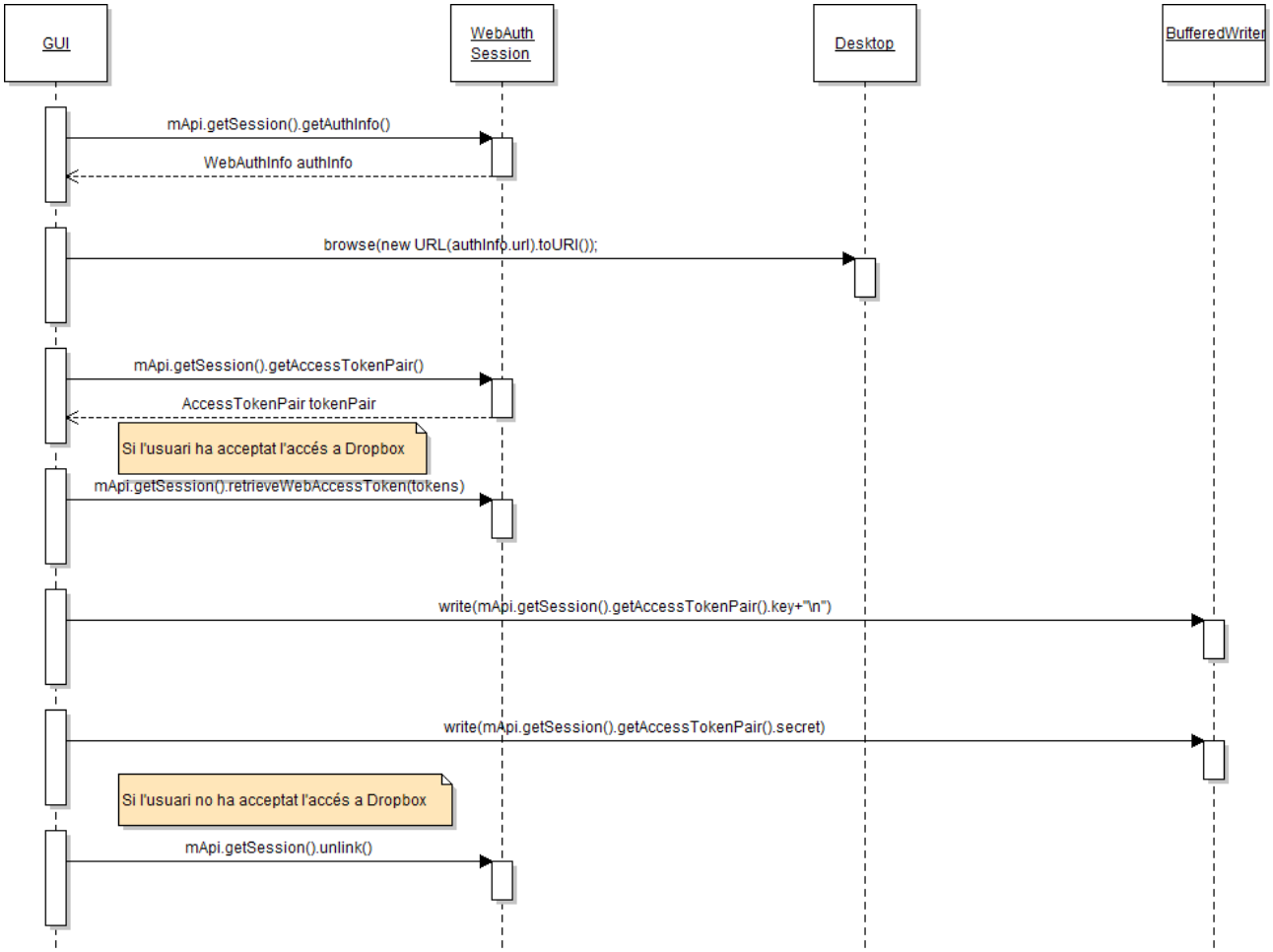

Figura 4. Diagrama d'execució del login de Dropbox a l'aplicació PC

Pujada de fitxers

Per al projecte d'*Android,* hem de definir una sèrie de classes *(Guardar_Apuntes, Guardar_Foto i Guardar_fichero)* que estenen d'*AsynkTask [9]* per tal de fer la pujada de fitxers a *Dropbox.* Com les peticions a *Dropbox* són tasques que poden trigar un temps i bloquejar l'aplicació mentre s'executen, necessitem fer-les des d'un *Thread* diferent al principal. La UI d'*Android* no permet fer crides des d'altres fils que no siguin el seu, i si volem implementar algun sistema com barres de progrés per tal de donar un feedback a l'usuari mentre s'executa la petició, el mètode més senzill és utilitzar el recurs esmentat.

Centrant-nos en la part estrictament de *Dropbox*, per fer una pujada d'un fitxer, hem d'obrir un *FileInputStream* indicant l'objecte File que volem pujar a *Dropbox.*

El següent pas és cridar al mètode *putFileOverwriteRequest* de l'objecte *DropboxApi* indicant-li com a paràmetres el nom que volem donar-li al fitxer que pujarem a *Dropbox*, *l'inputStream* abans indicat, la longitud del fitxer a pujar i opcionalment, podem indicar un *Listener* per tal d'actualitzar el progrés de la nostra tasca.

```
mRequest = (UploadRequest) mApi.putFileOverwriteRequest(path, inputStream,
                    file.length(), new ProgressListener() {
                        @Override
                        public long progressInterval() {
                            // Update the progress bar every half-second or
                            // so
                            return 100;
                        }
                        @Override
                        public void onProgress(long bytes, long total) {
                            if (isCancelled()) {

                            } else {
                                publishProgress(bytes);
                            }
                        }
                    });
```

Hem fet ús del mètode *overwrite* perquè ens permet sobreescriure el contingut del fitxer si ja ha estat creat anteriorment, evitant que cada vegada que pugem un fitxer es dupliqui.

Finalment cridem al mètode *upload* per fer efectiva la pujada:

```
mRequest.upload();
```

A l'aplicació de PC ens trobem gairebé amb el mateix problema. Tenim el fil *EDT(Event Dispatch Thread)* que és l'encarregat de tractar el events de teclat i ratolí, a més d'encarregar-se de l'actualització de la GUI. Ja hem esmentat que la petició de pujada a *Dropbox* pot resultar una tasca que trigui en executar-se, i si la fem dintre d'aquest fil no podrem actualitzar la GUI perquè romandrà bloquejada fins que acabi l'execució de la pujada de fitxers. Volem implementar un sistema de barra de progrés per millorar el feedback cap a l'usuari, per tant no podem fer-ho a l'*EDT.*

Podem utilitzar *Thread* per llençar un fil quan volem per una pujada, i utilitzar *invokeLater()* per tal d'anar refrescant la GUI al fil principal EDT, però des de Java 6 tenim la classe *SwingWorker [17]* que s'encarrega de llençar el fil i gestionar el canvi entre fils per fer les dues coses.

Pel que fa a la part de codi estrictament de *Dropbox*, utilitzem el mateix mètode que a l'aplicació *Android:*

```
Entry newEntry = mApi.putFileOverwrite(path, inputStream, file.length(), null);
```

SwingWorker implementa el mètode *process(List<Integer> chunks)* que ens permet actualitzar la barra de progrés de la GUI, per tant no fa falta passar-li cap *Listener* a *putFileOverWrite.*

Descàrrega de fitxers

És un procés similar a la pujada de fitxers, ja que necessitarem fer-ho en fils paral·lels per tal de no bloquejar el fil principal de la GUI. Pel que fa al projecte *Android*, hem d'utilitzar el mètode *getFile* de l'objecte de l'API de *Dropbox* per tal de descarregar informació d'un fitxer del servidor. L'estructura de la crida a aquest mètode és la següent:

```
DropboxFileInfo info = mApi.getFile(path, null, outputStream, null);
```

On *path* indica la localització (nom i extensió del fitxer inclosos) del fitxer que descarregarem i *outputStream* és un stream per escriure al fitxer local on volem descarregar la informació.

L'objecte *DropboxFileInfo* que retorna la descàrrega del fitxer ens serveix per obtenir informació del fitxer descarregar com les metadades.

En el cas de l'aplicació PC és la mateixa crida a la funció *getFile,* on també obtenim com a objecte retornar un objecte de tipus *DropboxFileInfo*.

Recordem que a l'aplicació *Android* la descàrrega es fa dins el marc d'una classe que estén *AsynkTask [9]* i que a l'aplicació PC es fa dins el marc d'una classe que estén *SwingWorker [17]*, ambdues classes faciliten l'execució paral·lela d'aquesta tasca.

Accés a metadades

Podem accedir a metadades mitjançant la mateixa crida en el dos projectes:

```
Entry existingEntry = mApi.metadata(path, 1, null, false, null);
```

Indiquem el *path* del fitxer del qual volem obtenir les metadades i la crida ens retorna un objecte *Entry.* Les metadades ens han estat útils sobretot per esbrinar quina és la data de modificació del fitxer i poder comparar antiguitats entre fitxers locals i fitxers al servidor :

```
date = df.parse(existingEntry.modified);
```

Per tal d'emmagatzemar les fotografies dels alumnes i no haver de descarregar-les cada cop que volem mostrar-les a la GUI utilitzem les metadades juntament amb la funció *share* per tal d'obtenir el link per compartir el fitxer [11]:

```
existingEntry = mApi.metadata("/"+apellidos+"."+extension, 1, null, false, null);
DropboxLink shareLink = mApi.share(existingEntry.path);
```

Developer branding Guide

Per últim, l'aplicació ha de seguir la guia d'estil fixada per *Dropbox* per tal de poder passar a fase de producció, que és on realment l'aplicació es pot distribuir. A continuació esmentarem les mes importants:

- El nom de l'aplicació no pot incloure 'Dropbox' ni començar per 'drop' ni ser similar fonèticament a *Dropbox.*
- És obligatori l'ús d'un logo personalitzat de l'aplicació que no s'assembli ni inclogui el logo de *Dropbox.*
- Es pot utilitzar el logo i el nom de *Dropbox* dins l'aplicació o en la publicitat de la nostra aplicació, utilitzant només els logos aprovats per l'equip de *Dropbox.*

La nostra aplicació ha passat la revisió obligatòria de l'equip de Dropbox per aquesta guia d'estil, amb la qual cosa l'aplicació està en fase de producció i es pot distribuir:

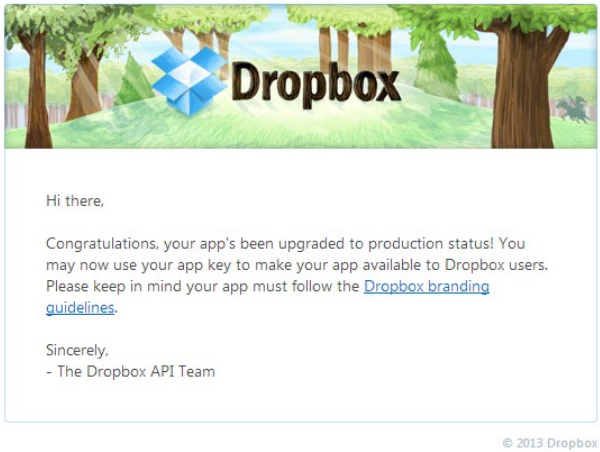

Figura 5. Notificació per a fase de producció per part de l'equip de Dropbox

3.3- Disseny i implementació de l'aplicació PC

A la descripció del projecte donem les especificacions pel desenvolupament d'una aplicació que permeti gestionar un curs acadèmic amb tot el que comporta (assignatures, classes, alumnes,etc..). Per tant, el primer que farem serà dissenyar el model Entitat-Relació del sistema per fer-nos una idea de quines classes haurem d'implementar i com interactuaran entre elles. El diagrama següent indica aquesta relació entre classes:

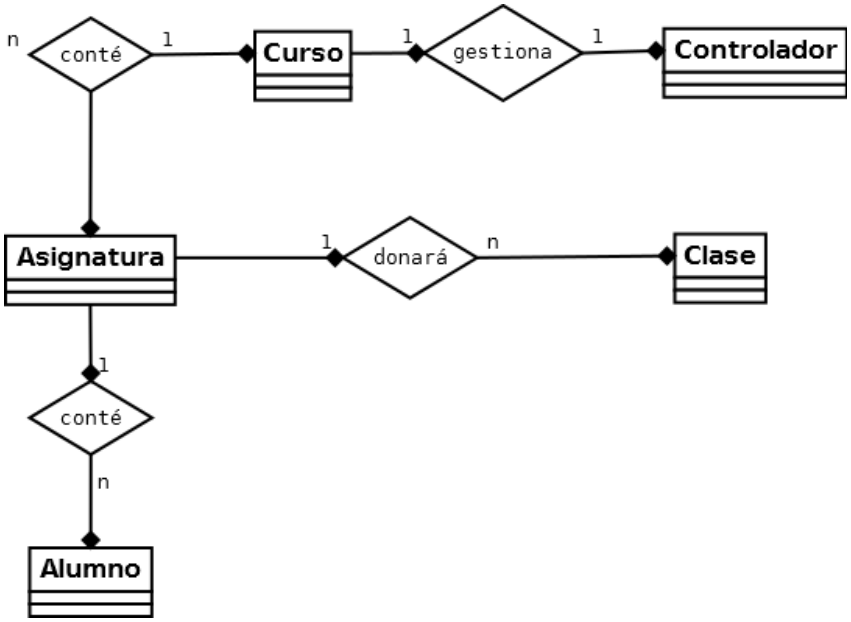

Figura 6. Model E-R

Definim els atributs de cadascuna de les classes definides al diagrama anterior:

-Controlador

- Curso curso

-Asignatura

- ArrayList<Alumno> alumnos
- ArrayList<Clase> clases
- String nombre

-Clase

- String descripcion
- long fecha
- String horaInicio
- int duracion
- boolean examen
- ArrayList<String> apuntes

-Curso

- ArrayList<Asignatura> asignaturas
- String nombre
- long fecha_inicio
- long fecha_final
- long fecha_actual

-Alumno

- String nombre
- String apellidos
- String enseñanza
- String NIUB
- String foto

La classe Controlador serà la que estarà instanciada a la classe principal de l'aplicació i s'encarregarà d'actuar entre la part d'informació de l'aplicació i la part visual o gràfica de l'aplicació. D'aquesta manera estructurem la nostra aplicació amb el patró d'arquitectura de software Model-Vista-Controlador, i així podrem re-utilitzar tota la part d'informació a les dues aplicacions i únicament variarà la part gràfica.

Definida la part del model i del controlador, passem a dissenyar i definir la part de la vista de l'aplicació:

Disseny

Dels requisits i de l'estudi previ hem extret que volem implementar una interfície d'usuari que mostri una visió temporal del curs, amb possibilitat de navegació pel calendari i afegir una visió mes detallada setmanal. Així l'usuari pot escollir entre veure un resum del seu dia, de la seva setmana, i del seu curs.

També hem de tenir en compte que la gestió dels alumnes serà una part important dins la GUI, per tant reservarem un espai per poder accedir ràpidament a aquesta funcionalitat.

Finalment, hem de reservar un espai per a la gestió dels cursos, de les assignatures i de les classes, a més de les funcionalitats que ens ofereix l'API de *Dropbox.*

El disseny de la nostra GUI tenint en compte aquests requisits és el següent:

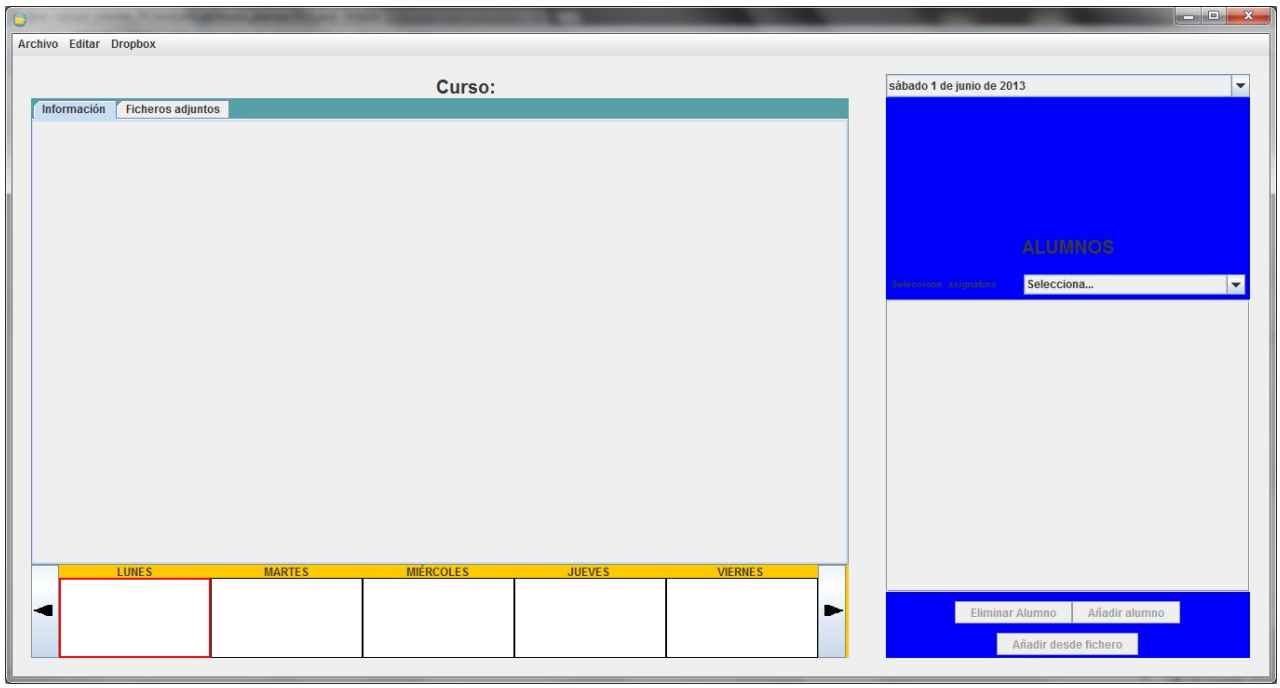

Figura 7. Aspecte de la nostra GUI

Com veiem a la figura 7, dividim la pantalla principal en dos zones. A la part esquerra tenim la part principal de l'aplicació, on tenim la visió setmanal del nostre curs, i un sistema de navegació entre setmanes. A la part central tenim on es mostra la informació del dia escollit, dividida entre informació general de les classes d'aquest dia i fitxers adjunts.

A la part dreta de l'aplicació hem situat un selector de calendari per dotar a l'usuari d'un accés mes ràpid a una data concreta del curs, a més del panell de la part de gestió d'alumnes, on podem seleccionar els alumnes de les diferents assignatures i les opcions d'afegir o eliminar-ne.

Per últim tenim el menú a dalt, amb una organització clàssica per funcionalitats com són nou curs, guardar els canvis, afegir/eliminar assignatures i el menú *Dropbox* que ens permet autenticar-nos i sincronitzar el contingut.

L'organització de tots els elements es fa mitjançant layouts de tipus *BoxLayout* que ens permeten definir l'orientació en la qual s'afegeixen els elements (X_AXYS,Y_AXYS).

Aquesta forma de treballar ens permet adaptar la grandària de la interfície gràfica a diferents resolucions de pantalla.

Implementació

Nou curs

Implementem *Curso_GUI* per tal d'afegir la informació del nou curs. A la interfície es mostra un camp de text per introduir el nom del curs i dos selectors de data per escollir data d'inici i data de final de curs. Una vegada l'usuari guarda les dades del nou curs, instanciem el nou objecte curs i l'afegim al controlador de la classe principal.

Omplim la vista setmanal amb les dates generades:

Figura 8. Vista setmanal del curs

Per fer-ho, comprovem a quin dia de la setmana comença el curs i en base a això omplim els *textPane* definits. La funció *getFechaFormateada(int dia)* ens retorna la data resultant de sumar el sencer *dia* a la data inicial, i aquest sencer dependrà de la setmana del curs en la que ens trobem. La fórmula per generar aquest sencer *dia* és la següent:

```
offset+(7*semana)
```

El valor *offset* dependrà del dia de la setmana on comenci el curs. Per exemple, si el curs comença en Dimecres, els dos primers *textPane* han d'estar buits, i en el tercer *textPane* generarem la data amb `0+(7*semana)` perquè al trobar-nos en la primera setmana el càlcul donarà `0+(7*0) = 0`. Així, sumarem 0 dies al dia inicial, i ens retornarà simplement el dia

inicial. D'aquesta manera és com generarem a totes les funcions les dates mostrades als *textPane*.

Afegir assignatura

Implementem el *JDialog Asignatura_GUI* per tal que l'usuari introdueixi els dies on es donaran classes, els horaris i la duració de les classes. Una vegada es confirmen les dades introduïdes, instanciem la nova assignatura i procedim a generar les classes de l'assignatura per a tot el curs. Per fer-ho, comprovem quantes setmanes té el curs i les recorrem generant les dates amb la mateixa fórmula utilitzada al crear un nou curs:

```
offset+(7*contadorSemanas)
```

Comprovem quins dies son els que ha escollit l'usuari per a aquesta assignatura, i tan sols generem classes per a aquests dies de la setmana. També tenim en compte que potser l'última setmana no es fa sencera, i comprovem si la data generada amb la fórmula anterior és anterior a la data final del curs. Finalment, també tenim en compte que la primera setmana també potser no es fa sencera, per tant, si l'usuari ha marcat un dia que la primera setmana no és lectiu, no crearem aquesta classe.

Pel que fa a l'offset aplicat, es té en compte el dia d'inici del curs. Per exemple, si el curs comença un Dimarts, per generar la data del següent Dilluns haurem de sumar una setmana i restar-li un dia,aplicant un *offset* `-1+(7*contadorSemanas)`.

Eliminar assignatura

Implementem el *JDialog Eliminar_Asignatura_GUI* per tal que l'usuari seleccioni les assignatures que vol eliminar. Quan l'usuari confirma els canvis, actualitzem la llista d'assignatures del curs i obtenim la llista d'assignatures eliminades. Aquestes

assignatures ens interessen perquè quan eliminem assignatures, hem d'eliminar els apunts associats a aquestes, a més de les fotografies del seus alumnes. Per això hem definit el vector *listaEliminados* que conté el nom dels fitxers que hem d'eliminar al confirmar el canvis. Preferim no eliminar els recursos esmentats quan l'usuari elimina l'assignatura perquè si després no desitja guardar els canvis, aquesta eliminació és més difícil de recuperar. Així, si no guarda els canvis i tanca l'aplicació tot queda com abans.

Afegir classe

Quan seleccionem un *textPane* que conté una data, al panell central es dóna l'opció d'afegir una classe per al dia seleccionat. Implementem el *JDialog Anadir_clase_GUI* per tal que l'usuari introdueixi l'assignatura de la qual serà la classe, l'horari, la duració i una descripció.

Eliminar classe

Donem a l'usuari l'opció d'eliminar una de les classes plantejades per a un dia determinat. Per fer-ho, l'usuari clica en un dels panells que representen les classes del dia concret i després escull l'opció d'eliminar. Quan es confirma l'eliminació, recorrem la llista de fitxers associats a aquella classe i guardem els seus noms al vector *listaEliminados* per eliminar-los quan l'usuari guardi els canvis.

Eliminar fitxers

Si cliquem un cop sobre un dels fitxers i escollim l'opció d'eliminar aquell fitxer, afegim el nom del fitxer al vector *listaEliminados* i després eliminem del vector d'apunts de la classe corresponents l'entrada escollida.

Selecció de data al CalendarCombo

Quan escollim una data al calendari desplegable [4] de la GUI, el primer que fem és comprovar que la data escollida està dins els marges establerts per la data d'inici i la data de final de curs. Si està dintre dels marges, haurem d'obtenir en quina setmana del curs es troba el dia seleccionat, perquè necessitem la setmana per generar les dates mostrades al panell setmanal. Per obtenir aquesta setmana, apliquem la fórmula:

```
c.setTimeInMillis(calendarCombo.getDate().getTime() - inicio.getTime().getTime());
semana = (c.get(Calendar.DAY_OF_YEAR)+offset)/7;
```

On l'*offset* dependrà com abans del dia d'inici del curs. Finalment marquem com a seleccionat el *textPane* corresponent al dia de la setmana de la data seleccionada. Les dates es carregaran cridant a la funció *actualizarGui* que genera els 5 dies de la setmana. Com prèviament hem modificat la variable *setmana* la creació de les dates serà correcta. A més, cridem a la funció *publicarContenidoDia(Calendar calendar)* per tal de carregar la informació de les classes al panell central.

Eliminar Alumne

Quan l'usuari escull un alumne dels mostrats i escull eliminar-lo, guardem a *listaEliminados* el nom de la fotografia associada a l'alumne eliminat. Sempre guardem les fotografies a *Dropbox* amb el format *cognoms.extensio*. Hem d'explicar que al camp foto dels alumnes no tenim guardat aquest format, ja que això implicaria que cada cop que l'usuari vol carregar els alumnes d'una assignatura, hauríem de descarregar les fotografies de cadascun mitjançant l'API de *Dropbox*. El que fem és guardar les URL de les fotografies [11] i accedir a elles mitjançant la classe *java.net.URL.*

Afegir alumne

Implementem el *JDialog Alumno_GUI* per tal que l'usuari introdueixi les dades del nou usuari com són el nom, els cognoms, l'ensenyament al que pertany, el NIUB i la fotografia. Una vegada confirma les dades introduïdes, pugem la fotografia mitjançant la classe que estén *SwingWorker* i guardem el nou alumne al controlador.

Afegir alumnes des de fitxer

Mostrem a l'usuari un selector de fitxers per tal que esculli el fitxer adient. Quan l'usuari tria una fitxer, encarreguem a la classe *Guardar_Alumno_Excel* que sigui qui s'encarregui de cridar al lector de fitxers. La classe *Guardar_Alumno_Excel* estén *SwingWorker* perquè a mesura que llegim les fotografies del fitxer *Excel* les anem pujant a *Dropbox*, i com a hem esmentat anteriorment, aquestes operacions s'han de fer a fils apart del principal.

Pel que fa al lector del fitxer *Excel*, utilitzem la llibreria *jxl* que ens permet, donat un fitxer amb extensió *xls,* accedir a les fulles i a les cel·les d'aquestes, podent llegir imatges o dades. Utilitzem el format de fitxer del campus virtual de la UB , on primer hi ha la fotografia i després està la informació.

Carregar Alumnes

Cada cop que l'usuari escull una de les assignatures del desplegable situat al panell d'alumnes, cridem a la funció *actualizarAlumnos(String nombre)* on fem ús de la classe *RenderLista* [13], que ens permet implementar un *JList* amb icones, que en aquest cas seran les fotografies dels alumnes. *RenderLista* implementa una taula *hash* que associa una imatge a un objecte, que en aquest cas serà el nom+cognoms de l'alumne al qual

pertany la fotografia. Per accedir a les fotografies, utilitza la classe *java.net.URL,* ja que el camp *foto* dels alumnes conté l'URL a la fotografia.

La *JList* té associat un *Listener* per tal de saber si hem clicat en algun dels alumnes. Si l'usuari clica dues vegades sobre un alumne, pot editar la informació de l'alumne en qüestió. Per fer-ho, hem implementat el *JDialog Editar_Alumno_GUI* que al obrir-se, ja conté les dades antigues de l'usuari. Quan l'usuari confirma les noves dades, modifiquem aquestes al controlador i comprovem si la fotografia associada a l'alumne ha variat o no. Si ho ha fet, hem de pujar-la a *Dropbox*.

Funció *publicarContenidoDia*

Aquesta funció és la que s'encarrega de la càrrega de la informació associada a la data escollida per l'usuari. S'encarrega de generar els panells de cada classe, tant al panell informació com al panell de fitxers associats. Per saber quina grandària han de tenir els panells per tal d'aprofitar tot l'espai que ens ofereix el panell central, primer fem una cerca per obtenir el nombre de classes que s'han de realitzar la data escollida. Així, dividim l'espai disponible entre el número de classes trobades.

El panell d'informació de cada classe porta associat un *listener* per tal d'escoltar els events de click produïts. Si cliquem un cop es ressalta la seva selecció, però si cliquem dues vegades, l'usuari pot editar les dades de la classe en concret. Per fer-ho, hem implementat el *JDialog Editar_Clase_GUI* que al obrir-se, conté les dades antigues de la classe i permet l'edició d'aquestes.

Pel que fa al panell de fitxer associats, també té associat un *Listener*. Si cliquem dos cops al panell, l'usuari pot pujar un fitxer per associar-lo a la classe corresponent. Per fer-ho

hem implement el *JDialog Editar_fichero_GUI* que té la mateixa estructura que el *JDialog* per editar una classe però aquest conté a més, l'opció de desplegar un selector de fitxers.

Alhora, la *JList* que conté el nom dels fitxers també té associada un *Listener*. Si cliquem un cop sobre un fitxer es ressalta la seva selecció, i si cliquem dos cops, ens permet obrir el fitxer des de l'aplicació. Per fer-ho, fem ús del recurs següent:

```
Process p = Runtime.getRuntime().exec("rundll32url.dll,FileProtocolHandler"+directory+"/"+item.toString());
p.waitFor();
```

Aquesta manera d'obrir fitxers des d'una aplicació Java funciona exclusivament en S.O Windows [16].

Creació de fitxer .exe

Per a la creació del fitxer *.exe* de l'aplicació hem utilitzat el programa *JSmooth [1][2]*. Aquest programa ens permet crear un fitxer executable de Windows a partir d'un fitxer Java JAR. En la creació d'aquest fitxer executable podem especificar quin és el missatge mostrar a l'usuari si no té instal·lada la màquina virtual de Java (necessària per l'execució de l'aplicació) i quina icona volem que tingui el fitxer executable. A més, hem d'especificar el fitxer JAR des del qual es crearà l'executable i quina és la classe principal de l'aplicació des d'on s'executarà. Per últim, podem especificar la versió mínima de Java que es necessita per executar la nostra aplicació.

3.4- Disseny i implementació de l'aplicació Android

La separació en capes de Model-Vista-Controlador ens permet re-utilitzar el 100% de la implementació de les parts Model i Controlador, perquè els requisits de les dues aplicacions són els mateixos i les dades a gestionar també. A més, per comunicar les dues aplicacions fem servir l'objecte controlador, per tant el contingut de les classes definides a les dues aplicacions ha de ser el mateix per tal de poder llegir els objectes correctament. La informació del controlador és emmagatzemada mitjançant *ObjectOutputStream* a un fitxer de text allotjat a *Dropbox*. Per poder carregar aquesta informació a l'altre aplicació, hem de llegir l'objecte mitjançant *ObjectInputStream*, i si no tenen el format exactament igual, no funcionarà. Així doncs, el disseny d'aquesta part serà el mateix que a l'aplicació PC i ens centrarem en explicar la part específica, que és la gràfica.

Disseny

Dels requisits previs i de l'estudi de mercat fet amb anterioritat, hem extret que busquem mostrar a l'usuari una visió temporal del curs, no simplement una llista ordenada de classes que ha de realitzar. Per tant, farem una primera pantalla on l'usuari pugui interactuar amb algun element per navegar entre dates i que alhora, ens mostri, per a la data escollida, les classes que s'han de realitzar.

També es vol gestionar assignatures, i de la mateixa manera que a l'aplicació PC, situarem totes les gestions relacionades amb les assignatures a un menú apart.

Per últim, la gestió dels alumnes és la tercera part important de la nostra aplicació, i també reservarem un apartat diferenciat de la resta tal com hem fet a l'aplicació PC. Hem de

tenir en compte que l'espai és més reduït, per tant haurem d'estudiar que recursos ens ofereix *Android* per distribuir de la millor manera totes aquestes funcionalitats.

Una de les maneres que més s'utilitzen actualment, i de fet el mateix S.O *Android* utilitza és el *ViewPager [7]*. Aquest element ens permet crear un efecte visual de continuïtat entre els diferents menús, permetent la navegació entre ells utilitzant l'efecte *sliding.* Així, definirem 3 apartats que seran Curs *(Figura 9),* Assignatures *(Figura 10)* i Alumnes *(Figura 11).* Curs serà l'apartat principal de la GUI, ja que permetrà a l'usuari la navegació pel calendari i mostrarà el planning de classes del dia. Assignatures mostrarà el llistat d'assignatures del curs i les diferents opcions relacionades, i per últim Alumnes mostrarà el llistat d'alumnes d'una assignatura en concret i les diferents opcions relacionades.

Les opcions de connexió a *Dropbox,* sincronització amb aquest i guardar els canvis produïts volem que en tot moment estiguin visibles, per tant, les definirem fora del *ViewPager*, i restaran visibles encara que l'usuari es desplaci entre els diferents apartats.

L'estructura és la següent:

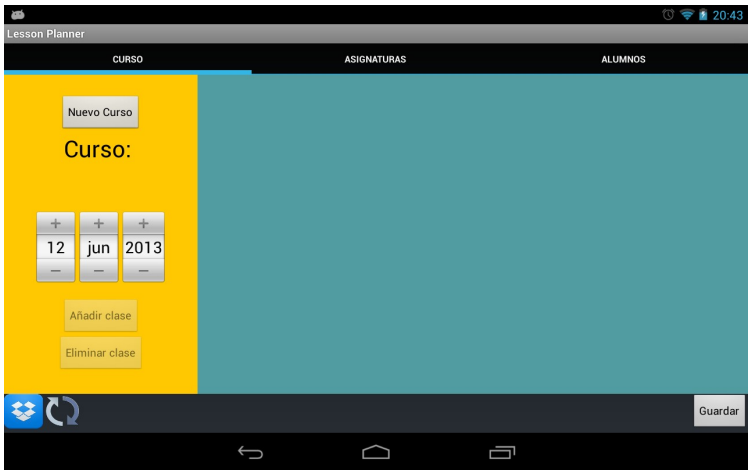

Figura 9. Panell 'Curso' de l'aplicació

Observem a la figura 9 com distribuïm l'espai disponible en dos subespais ben diferenciats. A l'espai groc hi situem els elements de gestió de les funcionalitats de

l'aplicació, mentre que a la part blava mostrarem la informació emmagatzemada a l'aplicació. Com ens trobem en el panell del curs, mostrarem les classes i els apunts del dia que l'usuari esculli al calendari.

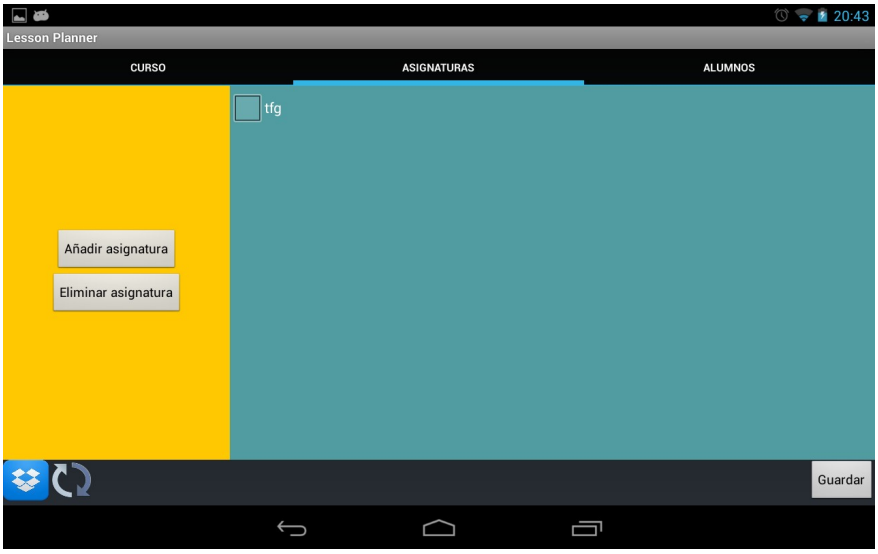

Figura 10. Panell 'Asignaturas' de l'aplicació

Seguint amb l'esquema de distribució de l'espai del panell Curs *(figura 9),* situem al panell *Asignaturas* (*figura 10*) els botons de les funcionalitats a la part groga, i a la part blava mostrem la informació de l'aplicació referent al panell on ens trobem, en aquest cas el nom de les assignatures.

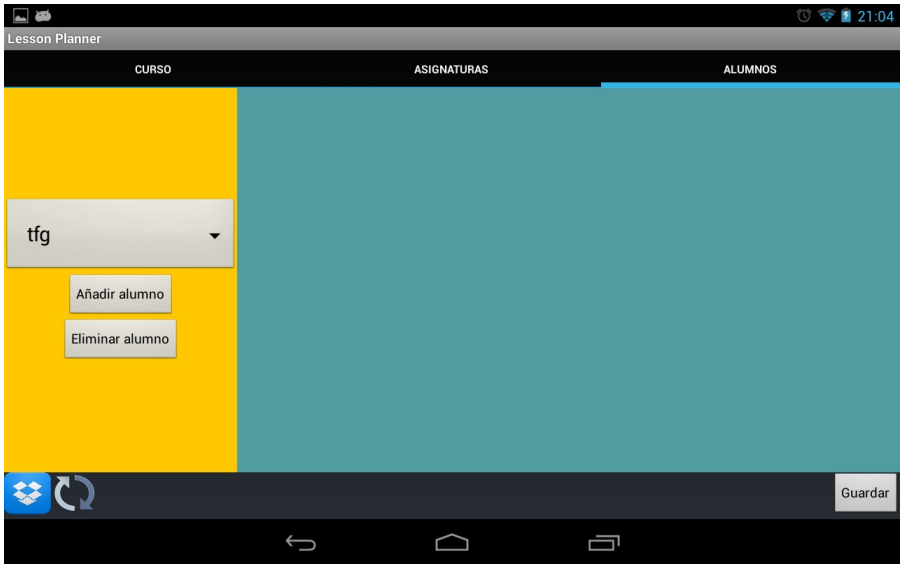

Figura 11. Panell 'Alumnos' de l'aplicació

Seguim l'esquema mostrat als anterior panells *(figures 9 i* 10), i al panell *Alumnos (figura 11)* definim una part groga on tenim els elements per executar les diferents funcionalitats i una part blava on mostrem la informació del panell en qüestió, en aquest cas els noms dels alumnes i les seves fotografies.

La distribució dels elements es fa utilitzant *LinearLayout* que ens permet agrupar els elements en vertical o en horitzontal.

Implementació

Per a la implementació de la GUI, implementem una classe principal *MainActivity* que estén *FragmentActivity*. Aquesta és la classe principal de la GUI, que contindrà l'objecte de l'API de *Dropbox* per a fer les peticions adients, el controlador per accedir a les dades de l'aplicació, i els elements de la GUI com són els botons de *Dropbox* i el *ViewPager.*

Dintre d'aquesta classe definim la classe que controlarà els diferents Fragment que es mostraran al *ViewPager.* Aquesta classe s'encarrega d'emmagatzemar l'estat dels Fragments i de carregar-los quan l'usuari navega entre les diferents pàgines.

Cada cop que l'usuari canvia de pantalla dintre del *ViewPager,* la classe *FragmentStatePagerAdapter* fa la següent crida:

```
TestFragment.newInstance(CONTENT[position % CONTENT.length],controlador.getCurso())
```

La classe *TestFragment* és la que s'encarrega de crear els diferents Fragment amb el seu contingut. *FragmentStatePagerAdapter* indica per paràmetre quin Fragment ha estat escollit, i *TestFragment* dins la funció *newInstance* crea el nou Fragment. Quan creem el nou Fragment, el mètode *oncreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)* s'encarrega de crear la vista del Fragment.

Per crear la vista, utilitzem *LayoutInflater* indicant quin layout de la carpeta *resources* volem carregar.

El següent esquema ens dóna una idea esquematitzada de tots els elements que conformen aquesta part de la GUI:

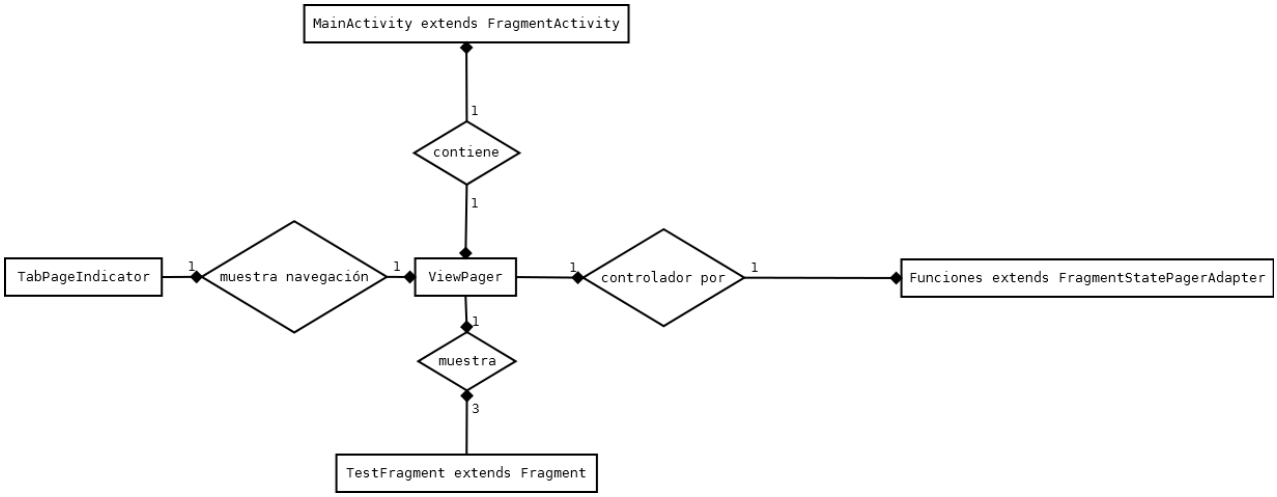

Figura 12. Esquema de relacions entre les classes que implementen la GUI

Observem a l'esquema de la figura 12 com apareix la classe *TabPageIndicator.* Aquesta classe forma part de la llibreria implementada per Jack Wharton *ViewPagerIndicator [8].* Aquesta llibreria ens permet afegir al *ViewPager* un indicador de navegació, i tot i que la llibreria sencera conté molts tipus d'indicadors de pàgina, nosaltres solament hem utilitzat els fitxers que implementen la manera d'indicar les pàgines anomenada *Tabs.* Aquests fitxers Java corresponen a *TabPageIndicator, IconPagerAdapter i PageIndicator.*

Pel que fa a les funcionalitats implementades, re-utilitzem la major part del codi implementat a l'aplicació PC, ja que la manera d'afegir i eliminar objectes és la mateixa.

Com a l'aplicació PC, sempre que hem de fer peticions a *Dropbox,* les hem de fer des de un fil apart, i en aquest cas implementem les classes *Cargar_ficheros, Sincronizacion,*

Descargar_fichero, Guardar_foto i Guardar_Apuntes que estenen de la classe *AsynkTask* [9] per tal de cobrir totes les funcionalitats requerides.

Aquestes classes són classes internes de la classe principal *MainActivity,* que és la que s'encarregarà d'instanciar-les i executar-les per procedir a realitzar les peticions *http* a *Dropbox*. Aquestes peticions seran moltes vegades requerides des d'algun dels Fragment definits, obligant a la comunicació entre Fragment i *MainActivity.* Per fer-ho, dins dels Fragment podem accedir a les funcions de l'activitat que els conté (en aquest cas *MainActivity)* fent la següent crida:

```
((MainActivity)getActivity())
```

D'aquesta manera ja podem cridar a qualsevol dels mètodes definits a *MainActivity:*

```
((MainActivity)getActivity()).guardarApuntes(fichero.getText().toString(),a,c);
```

Hem definit a *MainActivity* les funcions *guardarFoto, guardarApuntes, descargarArchivo, actualizarListaEliminados i actualizaGUI* per tal que des dels Fragment es pugui notificar a l'activitat principal que es vol fer una pujada o baixada de fitxers, que s'ha eliminat algun recurs de l'aplicació o simplement que l'usuari ha actualitzat la informació de l'aplicació i es vol actualitzar el contingut de la GUI.

Per veure gràficament aquesta comunicació *Fragment-MainActivity-AsynkTask* esquematitzem l'exemple d'afegir un alumne:

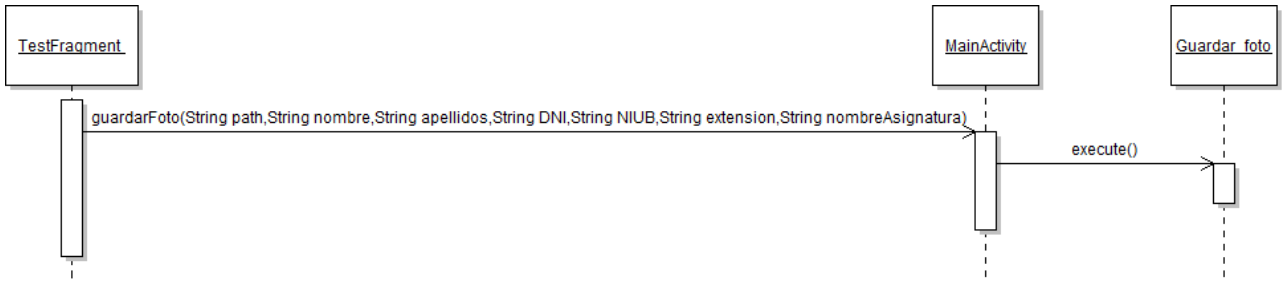

Figura 13. Exemple d'una comunicació Fragment-MainActivity-AsynkTask.

TestFragment s'encarrega de rebre les dades del formulari de l'usuari i passar-les a *MainActivity*. *MainActivity* s'encarrega d'instanciar i executar la classe que instanciarà el nou alumne i farà la petició a *Dropbox* per pujar la seva fotografia.

Llibreries i funcionalitats externes

A més de les llibreries de *Dropbox*, hem utilitzat la llibreria *Universal Image Loader* [14] per a *Android* per a gestionar la càrrega de les imatges dels alumnes a la interfície gràfica. Com sabem, les imatges dels alumnes estan situades a *Dropbox,* i les carreguem amb la seva URL. Aquesta llibreria ens permet la càrrega asíncrona d'imatges d'Internet, permetent l'ús de memòria *caché* per tal de fer més ràpida la càrrega de les imatges.

Per a implementar un selector de fitxers a l'estil de *JFileChooser* de Java, hem utilitzat la classe *DirectoryBrowser* [15] implementada per *REM Android.* Aquesta classe estén de ListActivity i ens permet navegar entre els directoris de la *sdCard.*

L'altre recurs que necessitem és el layout que defineix la *ListView filechooser.xml.* Per finalitzar, necessitem especificar a *AndroidManifest* l'activitat del *DirectoryBrowser*:

```
<activity
  android:name="com.ub.lesson_planner.DirectoryBrowser"
 android:screenOrientation="landscape">
 <intent-filter>
     <action android:name="android.intent.action.VIEW" />
     <category android:name="android.intent.category.DEFAULT" />
 </intent-filter>
</activity>
```

Per obrir el selector de fitxers, utilitzem la classe *Intent* que ens serveix per llençar una Activity:

```
Intent intent = new Intent(getActivity(), DirectoryBrowser.class);
startActivityForResult(intent,1);
```

3.5- Problemes trobats

- A l'aplicació *Android* hem definit un objecte *DatePicker* per seleccionar data. Hem detectat com quan l'usuari clica, l'event *onDateChange* s'executa dues vegades per click. La solució que hem trobat és comprovar dins l'event quantes vegades hem entrat i solament executar el codi interior un cop.

- Les primeres proves van anar encaminades a la comunicació mitjançant fitxers de text entre les dues aplicacions. Després de completar les GUI de les dues aplicacions, al tornar a a comprovar el funcionament de la comunicació, detectem un error en la lectura dels fitxers. El problema radica en que les variables de tipus Calendar, al escriure els objectes al fitxer de text no les podem llegir correctament. La solució ha estat definir les variables com a tipus *long*, definint les dates en *ms.*

- En les primeres proves de pujades de fitxers a *Dropbox* i sobre-escriptura, he observat que si el contingut del fitxer a pujar no s'ha modificat des de l'última pujada, el fitxer de *Dropbox* no es sobreescriu ni s'actualitza la data de modificació.

- En un principi la versió mínima d'SDK definida al fitxer *AndroidManifest* era una versió antiga, i en aquest cas es podien fer les peticions *http* a *Dropbox* des del fil principal de l'aplicació. Quan comencem a desenvolupar la interfície gràfica, per tal d'utilitzar alguns recursos definits a versions SDK superiors, augmentem la versió definida i observem com les peticions a *Dropbox* ja no funcionen. La solució

aplicada ha estat utilitzar *AsynkTask* [9] per tal de gestionar les peticions a *Dropbox* en fils apart del principal.

- Per obrir els fitxers d'apunts a l'aplicació PC utilitzem *Desktop.open* i *Desktop.browser,* i detectem que a vegades l'aplicació detecta un error d'execució de Java i es tanca. La solució aplicada ha estat utilitzar:

```
Process p = Runtime.getRuntime().exec("rundll32

url.dll,FileProtocolHandler"+directory+"/"+item.toString());
p.waitFor();
```

Aquesta solució solament funciona per a Windows, però funciona en tot moment. Aquest error ha estat detectat per altres desenvolupadors:

http://stijndewitt.wordpress.com/2010/09/22/java-awt-desktop-open-fails-silently-without-exception/

https://www.java.net//node/684742

https://forums.oracle.com/forums/thread.jspa?threadID=1299449

4. Metodologia i resultats

4.1- Terminis de desenvolupament i costos

L'estudi sobre el contingut del projecte va començar a finals de Gener de 2013, quan començo a revisar i testejar aplicacions relacionades amb el tema. Les primeres setmanes ens serveixen per determinar objectius, estudiar els elements principals de la nostra aplicació i acotar quins recursos necessitarem per desenvolupar-la correctament. En aquestes primers setmanes determinem quin serà el sistema de sincronització, en aquest cas *Dropbox,* i comencem a estudiar el seu funcionament.

- Gener/Febrer

  - Estudi del mercat

  - Definició dels recursos que necessitarem (entorn de desenvolupament, mòbil o tablet per a *Android*,etc...)

  - Estudi de l'API de *Dropbox*

  - Implementació bàsica de les dues aplicacions per a primeres proves.

Després de comprovar que la comunicació es pot realitzar, decidim començar el disseny i desenvolupament de l'aplicació PC, obtenint una primera versió bàsica a principis del mes d'Abril. Per tant, durant tot el mes de Març treballem bàsicament en el codi de l'aplicació PC, que servirà també per a l'aplicació *Android*. En aquest moments, l'aplicació *Android* solament conté els elements bàsics per a poder-se comunicar amb *Dropbox*, però no té una interfície gràfica on puguem mostrar les dades.

- Març

  - Disseny de les classes que necessitarem implementar

  - Disseny de la GUI per a PC

  - Adaptació total de l'API de *Dropbox* dins el nostre codi.

- Primeres proves de disseny de la GUI de l'aplicació *Android*

En aquest moment ens trobem amb un prototip amb les funcionalitats bàsiques ja funcionant, i decidim deixar de banda l'aplicació PC per desenvolupar el projecte d'*Android* i tenir les dues aplicacions al mateix nivell de desenvolupament, a falta de la implementació de les funcionalitats més complicades.

- Abril/Maig

  - Centrem els nostres esforços en l'aplicació *Android*, adaptant el codi de les funcionalitats ja implementat a l'aplicació PC.

  - Dissenyem i implementem la GUI d'*Android* per poder mostrar les dades de sincronització de *Dropbox* i alhora, poder introduir dades, completant al 100% la comunicació entre les dues aplicacions.

  - Una vegada completada l'aplicació *Android,* implementem funcionalitats com la lectura de fitxers *Excel* o la càrrega de fotografies, a més de millorar el feedback cap a l'usuari amb la implementació de barres de progrés.

  - Modifiquem l'aspecte dels elements relacionat amb *Dropbox* per adaptar-nos a la seva guia d'estil.

  - Enviem les aplicacions a *Dropbox* per tal de ser testejades i revisades.

Una vegada *Dropbox* ens confirma l'estat de producció, decidim pujar l'aplicació *Android* a *Google Play* i crear un fitxer *.exe* de l'aplicació PC per tal de poder-la distribuir.

En total un treball de 5 mesos, i una estimació de treball setmanal de 13 o 14 hores. La dedicació mitjana diària es de 2 hores, i tenint en compte que la durada del projecte ha estat de 5 mesos, el total estimat d'hores de dedicació és de 300 hores.

4.2- Funcionalitats no implementades i possibles millores

En quant a les funcionalitats bàsiques definides al principi del projecte, l'aplicació PC és la que conté la implementació de totes. L'aplicació *Android* no té implementada la funcionalitat d'afegir alumnes des d'un fitxer *Excel*, perquè mentre llegim del fitxer *Excel* necessitem anar pujant les fotografies llegides a *Dropbox,* i la classe *AsynkTask* no ens permet mentre estem pujant parar la lectura del fitxer.

Pel que fa a les possibles millores, el ventall de funcionalitats que es podrien afegir és bastant ampli, però les que teníem pensat a curt plaç eren:

- Possibilitat de desplaçar les classes dins el calendari per si algun dia no es pot realitzar una classe, el contingut i els fitxers d'aquesta es moguin al següent dia i la resta de classes es desplacin amb ella.
- Exportació de cursos, per tal de facilitar la creació de plantilles per a cursos posteriors.
- Gestió d'exàmens i notes.

5. Conclusions

5.1- Anàlisi dels resultats

Revisarem els objectius inicials fixats i analitzarem quin és el resultat obtingut:

- **Desenvolupar un software de qualitat orientat a la gestió de cursos per a les plataformes Android i PC**

  Hem desenvolupat un software que compleix els requisits funcionals establerts als objectius, oferint a l'usuari un resum de les funcionalitats que ja oferien altres aplicacions al mercat però unides en una sola. Hem intentat que l'aspecte visual i l'organització dels elements a les dues aplicacions sigui semblant per tal de fer més fàcil l'aprenentatge i l'ús a l'usuari. Tot i que el ventall de funcionalitats oferides és limitat, sobretot per la limitació temporal del projecte, considerem que cobreix les necessitats bàsiques d'un usuari que busqui aquest tipus d'aplicacions. Finalment considerem que, pel que fa l'aplicació *Android*, l'aspecte visual i la navegació és el seu punt fort respecte altres aplicacions semblants de la mateixa plataforma.

- **Implementar un sistema de sincronització entre les diferents plataformes.**

  Hem utilitzat la plataforma *Dropbox* com a eina sincronitzadora i emmagatzemadora de la informació de les dues aplicacions. Hem utilitzat la seva recent API per poder utilitzar les funcionalitats de pujada i baixada de fitxers dins les nostres aplicacions, utilitzant els fitxers de text com a sistema de comunicació entre elles. A més, es la marca identitària de la nostra aplicació, i considerem que pot ser un reclam per als usuaris que diàriament utilitzen *Dropbox.*

6. Bibliografia

6.1- Bibliografia

[1]http://jsmooth.sourceforge.net/

[2]http://molda.es/crear-jar-ejecutable-que-contenga-todo-lo-necesario-en-eclipse/

[3]http://www.devtroce.com/2010/03/26/verificar-y-crear-directorio-con-java/

[4]http://saforas.wordpress.com/2011/01/29/codigo-java-agregar-un-componente-jcalendar-al-proyecto/

[5]http://androideity.com/2012/02/13/pestanas-en-android-usando-eclipse/

[6]https://github.com/vancexu/AimTo/blob/master/src/com/hackingtrace/vancexu/AimToActivity.java

[7]http://thepseudocoder.wordpress.com/2011/10/13/android-tabs-viewpager-swipe-able-tabs-ftw/

[8]http://viewpagerindicator.com/

[9]http://developer.android.com/reference/android/os/AsyncTask.html

[10]http://www.verious.com/qa/android-asynctask-upload-multiple-files-to-dropbox/

[11]http://stackoverflow.com/questions/5765533/dropbox-sharing-file-url

[12]http://stackoverflow.com/questions/7263291/viewpager-pageradapter-not-updating-the-view

[13]http://www.myjavazone.com/2011/01/jlist-con-iconos.html

[14]https://github.com/nostra13/Android-Universal-Image-Loader/blob/master/README.md

[15]http://www.remwebdevelopment.com/dev/a34/Directory-Browser-Application.html

[16]http://www.mkyong.com/java/how-to-open-a-pdf-file-in-java/

[17]http://chuwiki.chuidiang.org/index.php?title=Ejemplo_sencillo_con_SwingWorker