Servicios de las plataformas móviles

Sesión 1: Servicios de persistencia y notificaciones

Puntos a tratar

- *Autenticación de usuarios con Google Accounts*
- *Persistencia y datos en la nube con Google Drive*
- *Notificaciones push*

Google Drive

API de Google Drive para Android

- Introducción
  - API nativa
  - Automatiza tareas complejas (acceso sin conexión, sincronización de archivos)
  - Reducción tamaño de las aplicaciones
  - Proporciona abstracciones para la gestión de archivos y metadatos
    - DriveFile
    - DriveFolder
    - DriveResource
    - DriveContents

Descripción API de Google Drive (I)

- Metadatos de los recursos
  - Clase Metadata
  - Obtener los metadatos (DriveResourceClient.getMetadata())
  - Establecer los metadatos (MetadaChangeSet y DriveResourceClient.updateMetadata())
- Ficheros
  - Recuperación de ficheros
    - Mediante programación con DriveResourceClient.openFile()
    - Mediante selección a través de DriveClient.newOpenFileActivityIntentSender()
  - Referencia al contenido (DriveContents)
    - Acceso al contenido (DriveContents.getInputStream())
  - Creación de ficheros
    - Directamente por programación con DriveResourceClient.createFile()
    - A través de un cuadro de diálogo con DriveClient.newCreateFileActivityIntentSender()

Descripción API de Google Drive (II)

- Carpetas
  - DriveResourceClient.createFile(), DriveResourceClient.createFolder()
  - DriveResourceClient.listChildren(), DriveResourceClient.queryChildren()
- Constructores de Actividad
  - DriveClient.newCreateFileActivityIntentSender()
  - DriveClient.newOpenFileActivityIntentSender()
- Consultas
  - Clase Query

Google Drive

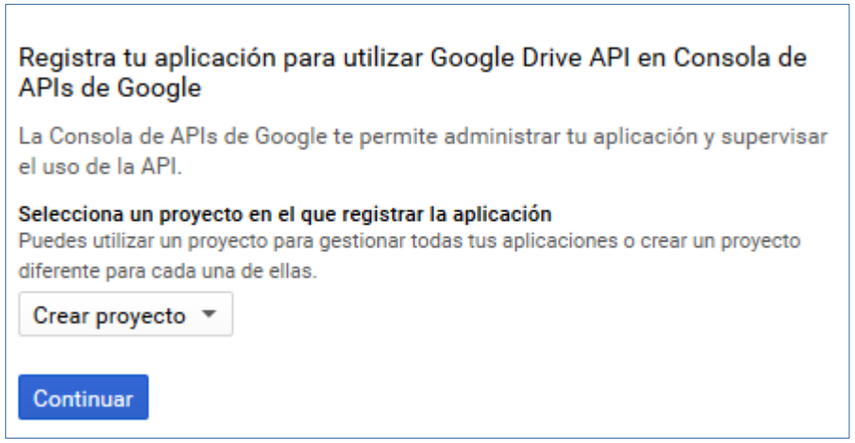
- Prerequisitos de la plataforma Google Drive para Android
- Habilitación de la API de Drive
- Programación de la aplicación
  - Configuración del proyecto
  - Inicialización del Cliente de la API de Google (GoogleApiClient)
  - Funcionalidades de la API
    - Creación de archivos
    - Trabajar con el contenido del archivo
    - Trabajar con los metadatos de archivos y carpetas
    - Fijar archivos
    - Trabajar con carpetas
    - Almacenamiento de datos del programa
    - Consulta de archivos

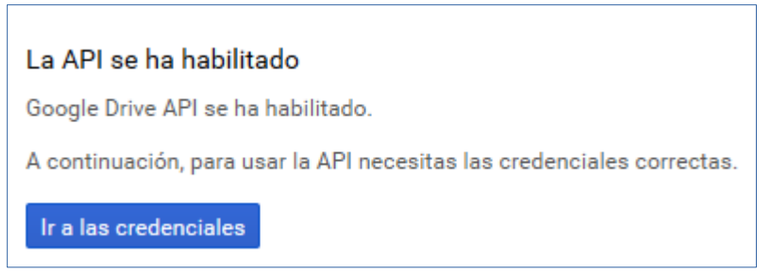
Prerequisitos de la plataforma Google Drive para Android

- Un dispositivo compatible con Android que ejecute Android 4.0 o superior y que incluya el Google Play Store, o un emulador de AVD (Dispositivo Virtual de Android) que tenga las Google APIs de Android 4.2.2 o superior y tenga la versión 11.8.0 o superior de Google Play Services.
- La última versión del Android SDK, incluyendo el componente SDK Tools.
- Un proyecto configurado para compilar contra Android 4.0 (Ice Cream Sandwich) o superior.
- El SDK de Google Play Services.

Habilitación de la API de Google Drive

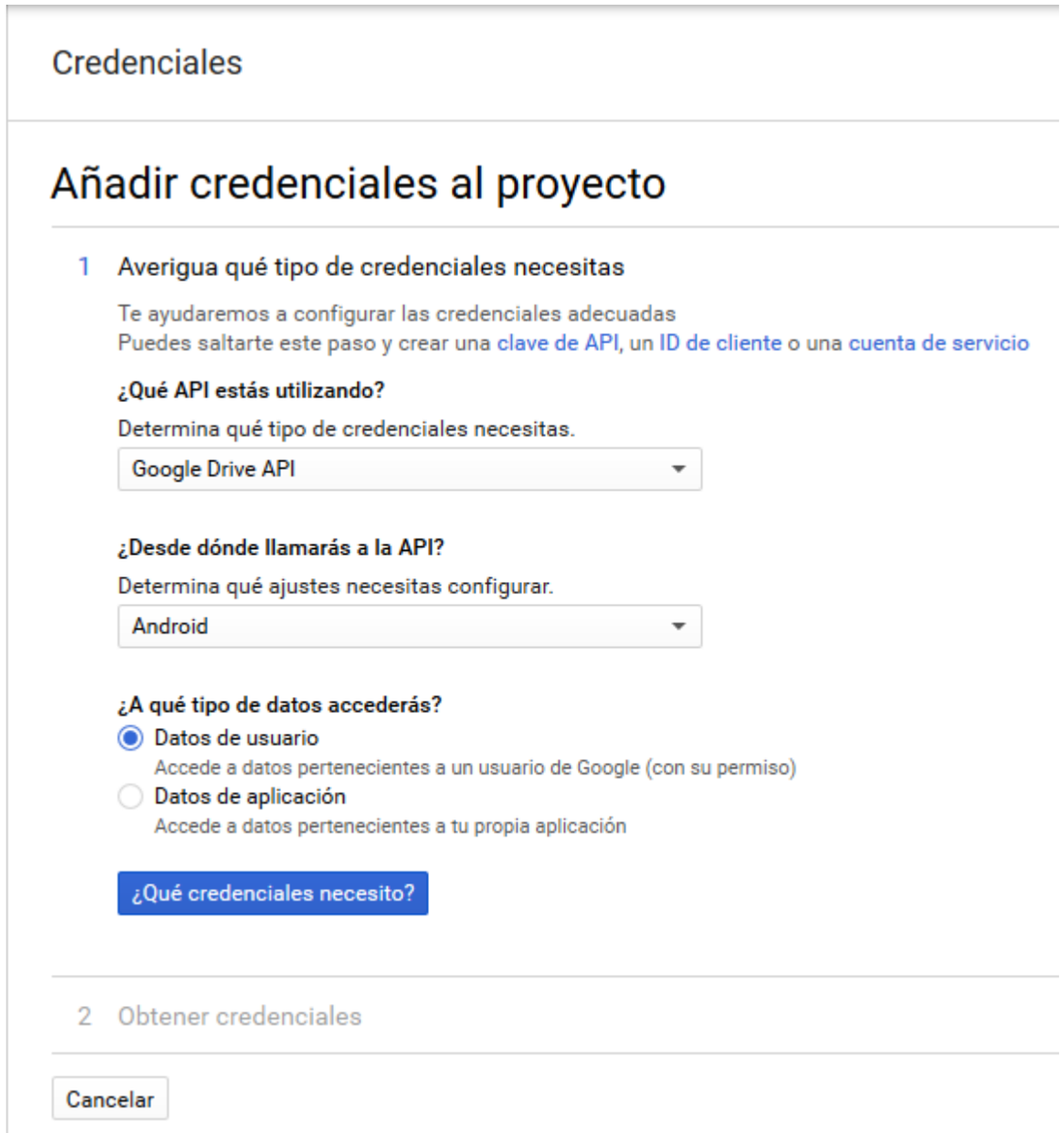
1. Ir a la dirección **https://console.developers.google.com/start/api?id=drive&credential=client_key**
2. Seleccionar un proyecto existente o crear uno nuevo.
3. Pulsar el botón para crear unas credenciales nuevas para nuestra aplicación.





Habilitación de la API de Google Drive

4. Rellenamos los datos solicitados para que el asistente nos lleve al tipo de credenciales que necesitamos.
5. Si tenemos unas credenciales ya creadas nos las ofrece por si nos valen. De lo contrario creamos unas nuevas.



Habilitación de la API de Google Drive

7. Elegir el tipo de aplicación. Seleccionamos “Android”. Introducir la huella digital del certificado de firma e introducimos el nombre del paquete de la aplicación.

Obtención huella del certificado (I)

Desde un terminal

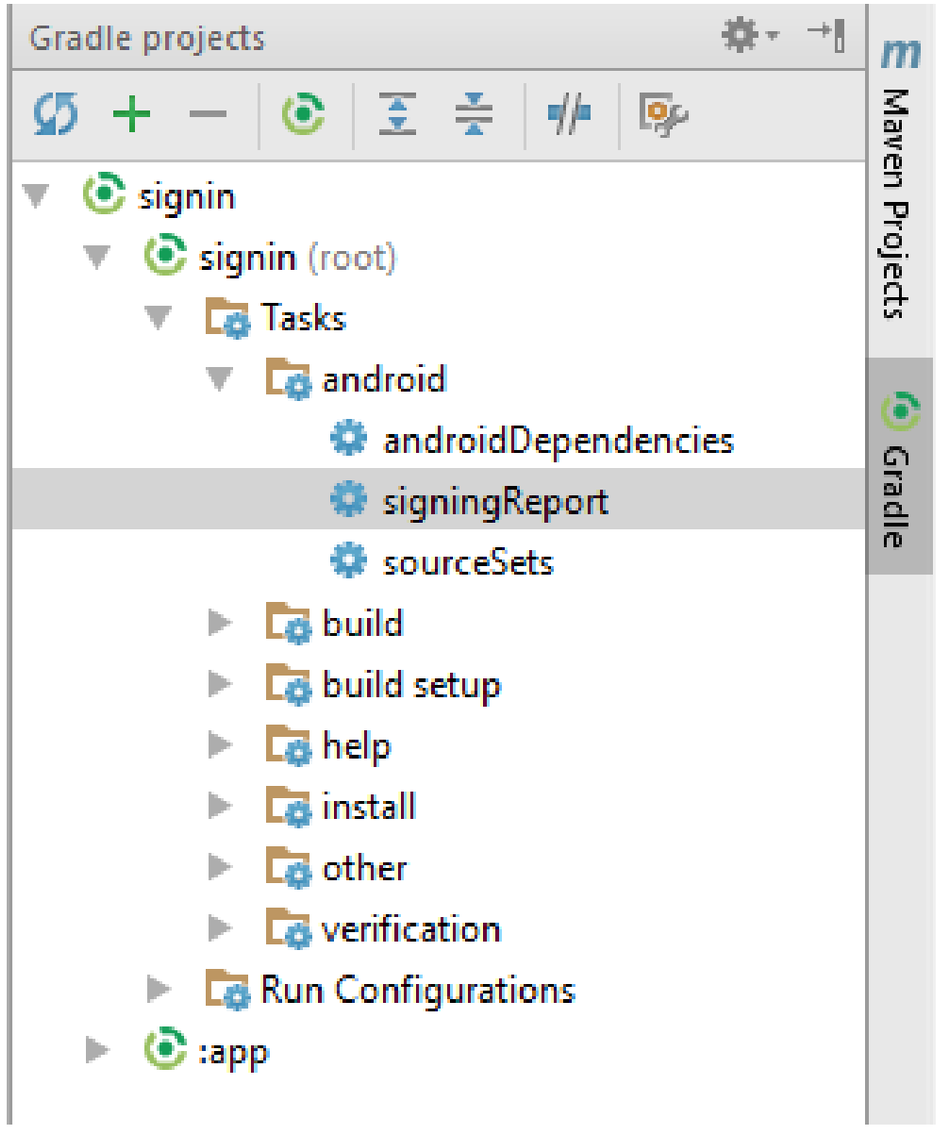
- Posicionarse en la carpeta en la que está la herramienta keytool (No necesario si la ruta está en la variable PATH).
- Ejecutar el comando:
  keytool -exportcert -alias androiddebugkey -keystore %userprofile%/.android/debug.keystore -list -v
  (ruta del almacén de certificados en windows)
- Introducir como contraseña **android**.
- La huella necesaria es el valor que sale en **SHA1**.

```
Certificate fingerprints:
   MD5:  1B:2B:2D:37:E1:CE:06:8B:A0:F0:73:05:3C:A3:63:DD
   SHA1: D8:AA:43:97:59:EE:C5:95:26:6A:07:EE:1C:37:8E:F4:F0:C8:05:C8
   SHA256: F3:6F:98:51:9A:DF:C3:15:4E:48:4B:0F:91:E3:3C:6A:A0:97:DC:0A:3F:B2:D2
   Signature algorithm name: SHA1withRSA
   Version: 3
```

Obtención huella del certificado (II)

Desde Android Studio

- Pulsar en la pestaña de Gradel
- Pulsar el botón de refrescar proyectos
- Desplegar el proyecto raiz
- Desplegar la carpeta Tasks
- Desplegar la carpeta android
- Hacer DobleClick sobre signingReport
- La huella necesaria es el valor que sale en **SHA1**.



Configuración del proyecto

- Añadir la dependencia de Google Play Services.

```
apply plugin: 'com.android.application'
    ...

    dependencies {
        implementation 'com.google.android.gms:play-services-auth:11.8.0'
        implementation 'com.google.android.gms:play-services-drive:11.8.0'
    }
```

Conectar y autorizar el API de Google Drive

- Comprobar si ya hay un usuario identificado, en caso contrario, crear el objeto GoogleSignInClient.
  - Indicar los ámbitos requeridos.

```
Set<Scope> requiredScopes = new HashSet<>( initialCapacity: 2);
requiredScopes.add(Drive.SCOPE_FILE);
requiredScopes.add(Drive.SCOPE_APPFOLDER);
GoogleSignInAccount signInAccount = GoogleSignIn.getLastSignedInAccount( context: this);
if (signInAccount != null && signInAccount.getGrantedScopes().containsAll(requiredScopes)) {
    initializeDriveClient(signInAccount);
} else {
    GoogleSignInOptions signInOptions =
            new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestScopes(Drive.SCOPE_FILE)
                .requestScopes(Drive.SCOPE_APPFOLDER)
                .build();
    GoogleSignInClient googleSignInClient = GoogleSignIn.getClient( activity: this, signInOptions);
    startActivityForResult(googleSignInClient.getSignInIntent(), REQUEST_CODE_SIGN_IN);
}
```

Conectar y autorizar el API de Google Drive

- Una vez iniciada la sesión obtenemos la cuenta del usuario.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode) {
        case REQUEST_CODE_SIGN_IN:
            if (resultCode != RESULT_OK) {
                // Sign-in may fail or be cancelled by the user. For this sample, sign-in is
                // required and is fatal. For apps where sign-in is optional, handle
                // appropriately
                Log.e(TAG,  msg: "Sign-in failed.");
                finish();
                return;
            }

            Task<GoogleSignInAccount> getAccountTask =
                GoogleSignIn.getSignedInAccountFromIntent(data);
            if (getAccountTask.isSuccessful()) {
                initializeDriveClient(getAccountTask.getResult());
            } else {
                Log.e(TAG,  msg: "Sign-in failed.");
                finish();
            }
            break;
```

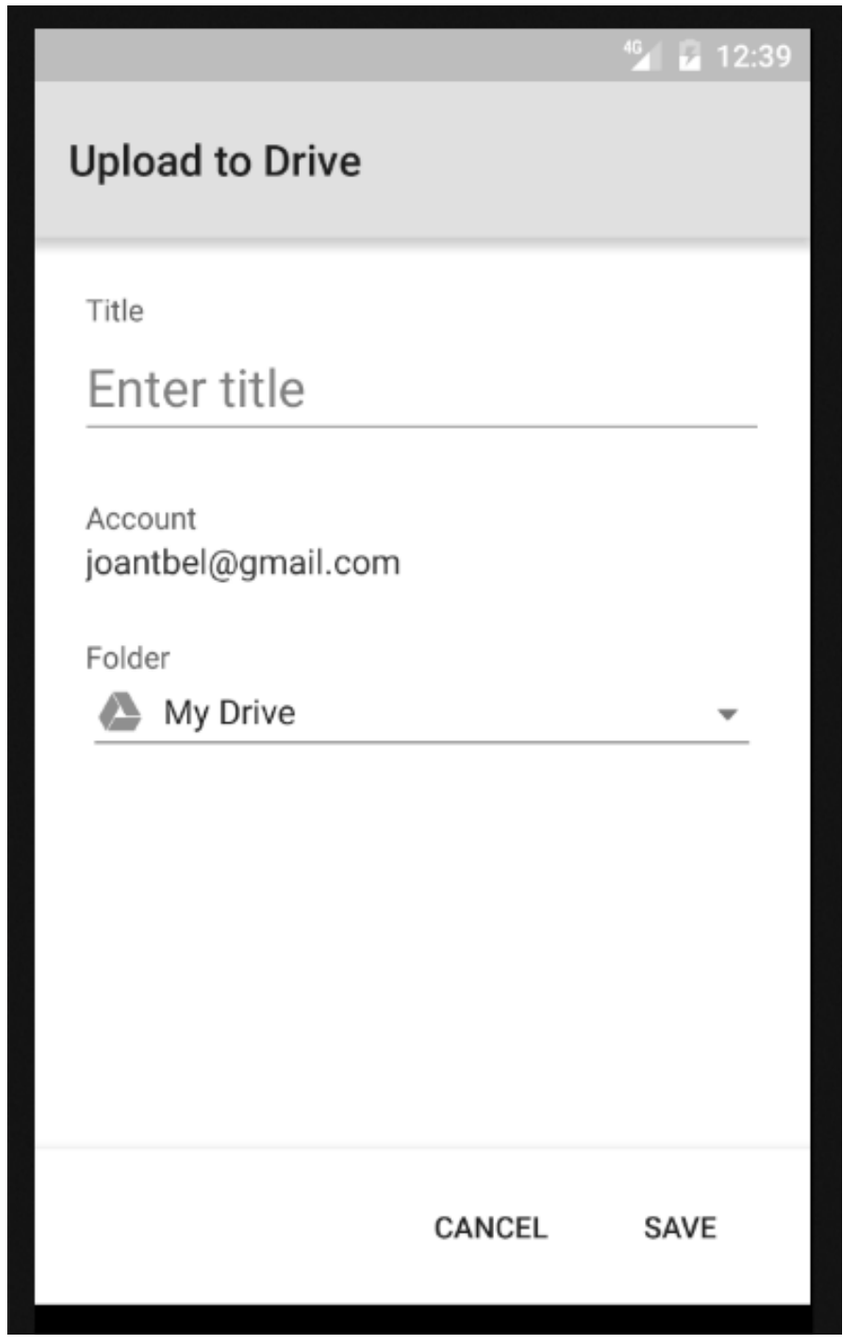
Conectar y autorizar el API de Google Drive

- Una vez que tenemos un objeto GoogleSignInAccount, creamos una instancia de DriveClient y de DriveResourceClient.

```
/**
 * Continues the sign-in process, initializing the Drive clients with the current
 * user's account.
 */
private void initializeDriveClient(GoogleSignInAccount signInAccount) {
    mDriveClient = Drive.getDriveClient(getApplicationContext(), signInAccount);
    mDriveResourceClient = Drive.getDriveResourceClient(getApplicationContext(), signInAccount);
    onDriveClientReady();
}
```

Creación de archivos

- Utilizando el constructor de actividad.
- Mediante programación
- Crear archivos vacíos



Creación de archivos

- Utilizando el constructor de actividad

```
Task<DriveContents> createContentsTask = getDriveResourceClient().createContents();
createContentsTask
        .continueWithTask(new Continuation<DriveContents, Task<IntentSender>>() {
            @Override
            public Task<IntentSender> then(@NonNull Task<DriveContents> task)
                    throws Exception {
                DriveContents contents = task.getResult();
                OutputStream outputStream = contents.getOutputStream();
                try (Writer writer = new OutputStreamWriter(outputStream)) {
                    writer.write("Hello World!");
                }

                MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                                    .setTitle("New file")
                                                    .setMimeType("text/plain")
                                                    .setStarred(true)
                                                    .build();

                CreateFileActivityOptions createOptions =
                        new CreateFileActivityOptions.Builder()
                                .setInitialDriveContents(contents)
                                .setInitialMetadata(changeSet)
                                .build();
                return getDriveClient().newCreateFileActivityIntentSender(createOptions);
            }
        })
```

Creación de archivos

- Utilizando el constructor de actividad (continuación)

```
.addOnSuccessListener(this,
        new OnSuccessListener<IntentSender>() {
            @Override
            public void onSuccess(IntentSender intentSender) {
                try {
                    startIntentSenderForResult(
                            intentSender, REQUEST_CODE_CREATE_FILE, null, 0, 0, 0);
                } catch (IntentSender.SendIntentException e) {
                    Log.e(TAG, "Unable to create file", e);
                    showMessage(getString(R.string.file_create_error));
                    finish();
                }
            }
        })
.addOnFailureListener(this, new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Log.e(TAG, "Unable to create file", e);
        showMessage(getString(R.string.file_create_error));
        finish();
    }
});
```

Creación de archivos

- Mediante programación

```
final Task<DriveFolder> rootFolderTask = getDriveResourceClient().getRootFolder();
final Task<DriveContents> createContentsTask = getDriveResourceClient().createContents();
Tasks.whenAll(rootFolderTask, createContentsTask)
        .continueWithTask(new Continuation<Void, Task<DriveFile>>() {
            @Override
            public Task<DriveFile> then(@NonNull Task<Void> task) throws Exception {
                DriveFolder parent = rootFolderTask.getResult();
                DriveContents contents = createContentsTask.getResult();
                OutputStream outputStream = contents.getOutputStream();
                try (Writer writer = new OutputStreamWriter(outputStream)) {
                    writer.write("Hello World!");
                }

                MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                                    .setTitle("HelloWorld.txt")
                                                    .setMimeType("text/plain")
                                                    .setStarred(true)
                                                    .build();

                return getDriveResourceClient().createFile(parent, changeSet, contents);
            }
        })
```

Creación de archivos

- Mediante programación (continuación).

```
.addOnSuccessListener(this,
        new OnSuccessListener<DriveFile>() {
            @Override
            public void onSuccess(DriveFile driveFile) {
                showMessage(getString(R.string.file_created,
                        driveFile.getDriveId().encodeToString()));
                finish();
            }
        })
.addOnFailureListener(this, new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception e) {
        Log.e(TAG, "Unable to create file", e);
        showMessage(getString(R.string.file_create_error));
        finish();
    }
});
```

Creación de archivos

- Crear archivos vacíos → Pasar null en lugar de DriveContents.

```
getDriveResourceClient()
        .getRootFolder()
        .continueWithTask(new Continuation<DriveFolder, Task<DriveFile>>() {
            @Override
            public Task<DriveFile> then(@NonNull Task<DriveFolder> task) throws Exception {
                DriveFolder parentFolder = task.getResult();
                MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                                    .setTitle("New file")
                                                    .setMimeType("text/plain")
                                                    .setStarred(true)
                                                    .build();
                return getDriveResourceClient().createFile(parentFolder, changeSet, null);
            }
        })
        .addOnSuccessListener(this,
                new OnSuccessListener<DriveFile>() {
                    @Override
                    public void onSuccess(DriveFile driveFile) {
                        showMessage(getString(R.string.file_created,
                                driveFile.getDriveId().encodeToString()));
                        finish();
                    }
                })
        .addOnFailureListener(this, new OnFailureListener() {
```

Trabajar con el contenido del archivo

- Acceso al contenido
  - Usando InputStream y OutputStream
  - Mediante ParcelFileDescriptor
- Ciclo de vida de un archivo de Drive
- Lectura de archivos
  - Recuperar el objeto DriveFile
  - Abrir el contenido del archivo
  - Lectura desde el flujo de entrada
  - Cerrando el contenido del archivo
- Escribiendo en un archivo
  - Recuperar el objeto DriveFile
  - Abrir el contenido del archivo
  - Hacer modificaciones
  - Cerrando el contenido del archivo

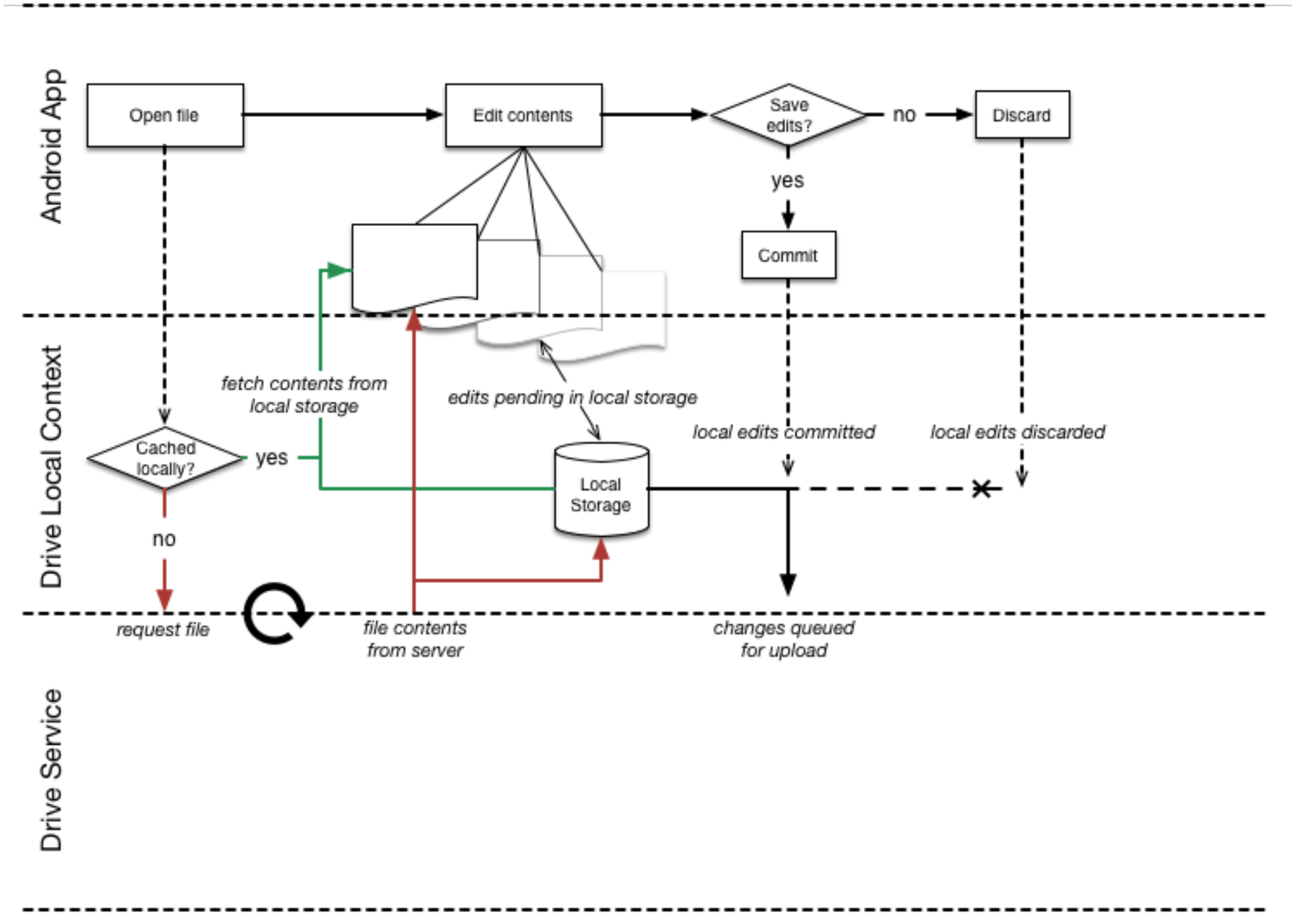
Acceso al contenido

- Clase DriveContents.

- Modo, métodos y clases de acceso.

Modo	Método	Clase
MODE_READ_ONLY	getInputStream	InputStream
MODE_WRITE_ONLY	getOutputStream	OutputStream
MODE_READ_WRITE	getParcelFileDescriptor	ParcelFileDescriptor

Ciclo de vida de un archivo de Drive



Lectura de archivos

- Recuperar el objeto DriveFile.

```
/**
 * Prompts the user to select a text file using OpenFileActivity.
 *
 * @return Task that resolves with the selected item's ID.
 */
protected Task<DriveId> pickTextFile() {
    OpenFileActivityOptions openOptions =
            new OpenFileActivityOptions.Builder()
                    .setSelectionFilter(Filters.eq(SearchableField.MIME_TYPE,  t: "text/plain"))
                    .setActivityTitle("Select file")
                    .build();
    return pickItem(openOptions);
}
```

```
@Override
protected void onDriveClientReady() {
    pickTextFile()
            .addOnSuccessListener( activity: this,
                    (OnSuccessListener) (driveId) → {
                            retrieveContents(driveId.asDriveFile());
                    })
            .addOnFailureListener( activity: this, (e) → {
                    Log.e(TAG,  msg: "No file selected", e);
                    showMessage("No file selected, can not continue.");
                    finish();
            });
}
```

Abrir el contenido del archivo (I)

- Modo de apertura
  - DriveFile.MODE_READ_ONLY(InputStream/ParcelFileDescriptor)
  - DriveFile.MODE_READ_WRITE (ParcelFileDescriptor)
- Método para abrir el archivo

```
Task<DriveContents> openFileTask =
        getDriveResourceClient().openFile(file, DriveFile.MODE_READ_ONLY);
openFileTask
        .continueWithTask(new Continuation<DriveContents, Task<Void>>() {
            @Override
            public Task<Void> then(@NonNull Task<DriveContents> task) throws Exception {
                DriveContents contents = task.getResult();
                // Process contents...

                Task<Void> discardTask = getDriveResourceClient().discardContents(contents);
                return discardTask;
            }
        })
        .addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                // Handle failure
            }
        });
```

Abrir el contenido del archivo (II)

- Abrir el archivo viendo la descarga

```
OpenFileCallback openCallback = new OpenFileCallback() {
    @Override
    public void onProgress(long bytesDownloaded, long bytesExpected) {
        // Update progress dialog with the latest progress.
        int progress = (int) (bytesDownloaded * 100 / bytesExpected);
        Log.d(TAG, String.format("Loading progress: %d percent", progress));
        mProgressBar.setProgress(progress);
    }

    @Override
    public void onContents(@NonNull DriveContents driveContents) {
        // onProgress may not be called for files that are already
        // available on the device. Mark the progress as complete
        // when contents available to ensure status is updated.
        mProgressBar.setProgress(100);
        // Read contents
    }

    @Override
    public void onError(@NonNull Exception e) {
        // Handle error
    }
};

getDriveResourceClient().openFile(file, DriveFile.MODE_READ_ONLY, openCallback);
```

Lectura desde el flujo de entrada

- *DriveContents* ofrece un *java.io.InputStream* a través del método *getInputStream*, para leer el contenido del archivo.

```
try (BufferedReader reader = new BufferedReader(
                new InputStreamReader(contents.getInputStream()))) {
    StringBuilder builder = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        builder.append(line).append("\n");
    }
    showMessage(getString(R.string.content_loaded));
    mFileContents.setText(builder.toString());
}
```

Escribiendo en un archivo

- Recuperar el objeto DriveFile.
  - Se realiza igual que en la lectura de archivo.
- Abrir el contenido del archivo, en este caso para escritura.
- Recuperar y procesar el contenido.
  - OutputStream
  - ParcelFileDescriptor
- Cerrar el contenido del archivo.

Abrir el contenido del archivo

- Modo de apertura
  - DriveFile.MODE_WRITE_ONLY (OutputStream/ParcelFileDescriptor)
  - DriveFile.MODE_READ_WRITE (ParcelFileDescriptor)
- Método para abrir el archivo

```
Task<DriveContents> openTask =
        getDriveResourceClient().openFile(file, DriveFile.MODE_READ_WRITE);
```

Hacer modificaciones

```
openTask.continueWithTask(new Continuation<DriveContents, Task<Void>>() {
            @Override
            public Task<Void> then(@NonNull Task<DriveContents> task) throws Exception {
                DriveContents driveContents = task.getResult();
                ParcelFileDescriptor pfd = driveContents.getParcelFileDescriptor();
                long bytesToSkip = pfd.getStatSize();
                try (InputStream in = new FileInputStream(pfd.getFileDescriptor())) {
                    // Skip to end of file
                    while (bytesToSkip > 0) {
                        long skipped = in.skip(bytesToSkip);
                        bytesToSkip -= skipped;
                    }
                }
                try (OutputStream out = new FileOutputStream(pfd.getFileDescriptor())) {
                    out.write("Hello world".getBytes());
                }

                MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                                    .setStarred(true)
                                                    .setLastViewedByMeDate(new Date())
                                                    .build();
                Task<Void> commitTask =
                        getDriveResourceClient().commitContents(driveContents, changeSet);

                return commitTask;
            }
        })
        .addOnSuccessListener(this,
```

Cerrando el contenido del archivo

- Validando cambios

```
Task<Void> commitTask =
        getDriveResourceClient().commitContents(driveContents, null);
```

- Actualizando metadatos

```
MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                        .setStarred(true)
                                        .setLastViewedByMeDate(new Date())
                                        .build();
Task<Void> commitTask =
        getDriveResourceClient().commitContents(driveContents, changeSet);
```

- Descartando cambios

```
Task<Void> discardTask = getDriveResourceClient().discardContents(contents);
```

Trabajar con los metadatos de archivos y carpetas

- Tipos de metadatos
  - Estándar
    - Nombre del archivo o carpeta
    - Tipo MIME del archivo o carpeta
    - Si el archivo o carpeta es destacado
    - Si el archivo es editable
    - Si el archivo o carpeta está eliminado
  - Personalizados
- Recuperación de los metadatos
  - DriveResourceClient.getMetadata
- Establecimiento de los metadatos
  - MetadataChangeSet
  - DriveResourceClient.updateMetadata

Recuperación de los metadatos

```
Task<Metadata> getMetadataTask = getDriveResourceClient().getMetadata(file);
getMetadataTask
        .addOnSuccessListener(this,
                new OnSuccessListener<Metadata>() {
                    @Override
                    public void onSuccess(Metadata metadata) {
                        showMessage(getString(
                                R.string.metadata_retrieved, metadata.getTitle()));
                        finish();
                    }
                })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.e(TAG, "Unable to retrieve metadata", e);
                showMessage(getString(R.string.read_failed));
                finish();
            }
        });
```

Establecimiento de los metadatos

```
MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                    .setStarred(true)
                                    .setIndexableText("Description about the file")
                                    .setTitle("A new title")
                                    .build();
Task<Metadata> updateMetadataTask =
        getDriveResourceClient().updateMetadata(file, changeSet);
updateMetadataTask
        .addOnSuccessListener(this,
                new OnSuccessListener<Metadata>() {
                    @Override
                    public void onSuccess(Metadata metadata) {
                        showMessage(getString(R.string.metadata_updated));
                        finish();
                    }
                })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.e(TAG, "Unable to update metadata", e);
                showMessage(getString(R.string.update_failed));
                finish();
            }
        });
```

Metadatos personalizados

- Tienen la forma clave-valor
- Visibilidad:
  - Pública
  - Privada
- Se identifican por su nombre + visibilidad

Límites por recurso	**Máximo**
Propiedades personalizadas Públicas	30
Propiedades personalizadas privadas por aplicación	30
Total de propiedades personalizadas	100
Longitud de propiedades personalizadas (clave + valor)	124 bytes (codificación UTF-8)

Insertar propiedad personalizada

```
CustomPropertyKey approvalPropertyKey =
        new CustomPropertyKey("approved", CustomPropertyKey.PUBLIC);
CustomPropertyKey submitPropertyKey =
        new CustomPropertyKey("submitted", CustomPropertyKey.PUBLIC);
MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                    .setCustomProperty(approvalPropertyKey, "yes")
                                    .setCustomProperty(submitPropertyKey, "no")
                                    .build();
Task<Metadata> updateMetadataTask =
        getDriveResourceClient().updateMetadata(file, changeSet);
updateMetadataTask
        .addOnSuccessListener(this,
                new OnSuccessListener<Metadata>() {
                    @Override
                    public void onSuccess(Metadata metadata) {
                        showMessage(getString(R.string.custom_property_updated));
                        finish();
                    }
                })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.e(TAG, "Unable to update metadata", e);
                showMessage(getString(R.string.update_failed));
                finish();
            }
        });
```

Eliminar propiedad personalizada

```
CustomPropertyKey approvalPropertyKey =
        new CustomPropertyKey("approved", CustomPropertyKey.PUBLIC);
CustomPropertyKey submitPropertyKey =
        new CustomPropertyKey("submitted", CustomPropertyKey.PUBLIC);
MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                    .deleteCustomProperty(approvalPropertyKey)
                                    .deleteCustomProperty(submitPropertyKey)
                                    .build();
Task<Metadata> updateMetadataTask =
        getDriveResourceClient().updateMetadata(file, changeSet);
updateMetadataTask
        .addOnSuccessListener(this,
                new OnSuccessListener<Metadata>() {
                    @Override
                    public void onSuccess(Metadata metadata) {
                        showMessage(getString(R.string.custom_property_deleted));
                        finish();
                    }
                })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.e(TAG, "Unable to update metadata", e);
                showMessage(getString(R.string.update_failed));
                finish();
            }
        });
```

Fijar archivos

- Atributos de la fijación
  - IsPinnable()
  - IsPinned()
- Flujo de trabajo típico para fijar y desfijar
  1. *Solicitar los metadatos del archivo.*
  2. *Cuando la tarea se complete, recuperar el objeto Metadata.*
  3. *Comprobar si isPinnable devuelve true.*
  4. *Comprobar el valor de isPinned.*
  5. *Establecer el parámetro setPinned en consecuencia.*
  6. *Llamar al método DriveResourceClient.updateMetadata, pasándole el objeto Metadata.*

Ejemplo de fijación

```
Task<Metadata> pinFileTask = getDriveResourceClient().getMetadata(file).continueWithTask(
        new Continuation<Metadata, Task<Metadata>>() {
            @Override
            public Task<Metadata> then(@NonNull Task<Metadata> task) throws Exception {
                Metadata metadata = task.getResult();
                if (!metadata.isPinnable()) {
                    showMessage(getString(R.string.file_not_pinnable));
                    return Tasks.forResult(metadata);
                }
                if (metadata.isPinned()) {
                    showMessage(getString(R.string.file_already_pinned));
                    return Tasks.forResult(metadata);
                }
                MetadataChangeSet changeSet =
                        new MetadataChangeSet.Builder().setPinned(true).build();
                return getDriveResourceClient().updateMetadata(file, changeSet);
            }
        });
```

```
pinFileTask
        .addOnSuccessListener(this,
                new OnSuccessListener<Metadata>() {
                    @Override
                    public void onSuccess(Metadata metadata) {
                        showMessage(getString(R.string.metadata_updated));
                        finish();
                    }
                })
        .addOnFailureListener(this, new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.e(TAG, "Unable to update metadata", e);
                showMessage(getString(R.string.update_failed));
                finish();
            }
        });
```

Trabajar con carpetas

- Obtener la carpeta raíz.
  - DriveResourceClient.getRootFolder()
- Crear una carpeta.
  - DriveResourceClient.createFolder()
- Crear un archivo dentro de una carpeta.
  - DriveResourceClient.createFile()
- Listar el contenido de una carpeta.

```
Task<MetadataBuffer> queryTask = getDriveResourceClient().listChildren(folder);
```

- Buscar contenido en una carpeta.

```
Task<MetadataBuffer> queryTask = getDriveResourceClient()
    .queryChildren(folder, query);
```

Crear una carpeta en la carpeta raiz

```
private void createFolder() {
    getDriveResourceClient()
            .getRootFolder()
            .continueWithTask(new Continuation<DriveFolder, Task<DriveFolder>>() {
                @Override
                public Task<DriveFolder> then(@NonNull Task<DriveFolder> task)
                        throws Exception {
                    DriveFolder parentFolder = task.getResult();
                    MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                                        .setTitle("New folder")
                                                        .setMimeType(DriveFolder.MIME_TYPE)
                                                        .setStarred(true)
                                                        .build();
                    return getDriveResourceClient().createFolder(parentFolder, changeSet);
                }
            })
            .addOnSuccessListener(this,
                    new OnSuccessListener<DriveFolder>() {
                        @Override
                        public void onSuccess(DriveFolder driveFolder) {
                            showMessage(getString(R.string.file_created,
                                    driveFolder.getDriveId().encodeToString()));
                            finish();
                        }
                    })
            .addOnFailureListener(this, new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.e(TAG, "Unable to create file", e);
                    showMessage(getString(R.string.file_create_error));
                    finish();
                }
            });
}
```

Carpeta de la aplicación

- ¿Qué es la carpeta de la aplicación?
- Obtener autorización para utilizarla
  - SCOPE_APPFOLDER
- Obtener la carpeta de la Aplicación
  - DriveResourceClient.getAppFolder
- Comprobar si un archivo o carpeta está en esta carpeta
  - Metadata.isInAppFolder
- Crear un archivo en la carpeta de la aplicación

Crear un archivo en la carpeta de la aplicación

```
private void createFileInAppFolder() {
    final Task<DriveFolder> appFolderTask = getDriveResourceClient().getAppFolder();
    final Task<DriveContents> createContentsTask = getDriveResourceClient().createContents();
    Tasks.whenAll(appFolderTask, createContentsTask)
            .continueWithTask(new Continuation<Void, Task<DriveFile>>() {
                @Override
                public Task<DriveFile> then(@NonNull Task<Void> task) throws Exception {
                    DriveFolder parent = appFolderTask.getResult();
                    DriveContents contents = createContentsTask.getResult();
                    OutputStream outputStream = contents.getOutputStream();
                    try (Writer writer = new OutputStreamWriter(outputStream)) {
                        writer.write("Hello World!");
                    }

                    MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
                                                        .setTitle("New file")
                                                        .setMimeType("text/plain")
                                                        .setStarred(true)
                                                        .build();

                    return getDriveResourceClient().createFile(parent, changeSet, contents);
                }
            })
            .addOnSuccessListener(this,
                    new OnSuccessListener<DriveFile>() {
                        @Override
                        public void onSuccess(DriveFile driveFile) {
                            showMessage(getString(R.string.file_created,
                                    driveFile.getDriveId().encodeToString()));
                            finish();
                        }
                    })
            .addOnFailureListener(this, new OnFailureListener() {
                @Override
                public void onFailure(@NonNull Exception e) {
                    Log.e(TAG, "Unable to create file", e);
                    showMessage(getString(R.string.file_create_error));
                    finish();
                }
            });
}
```

¿Preguntas...?