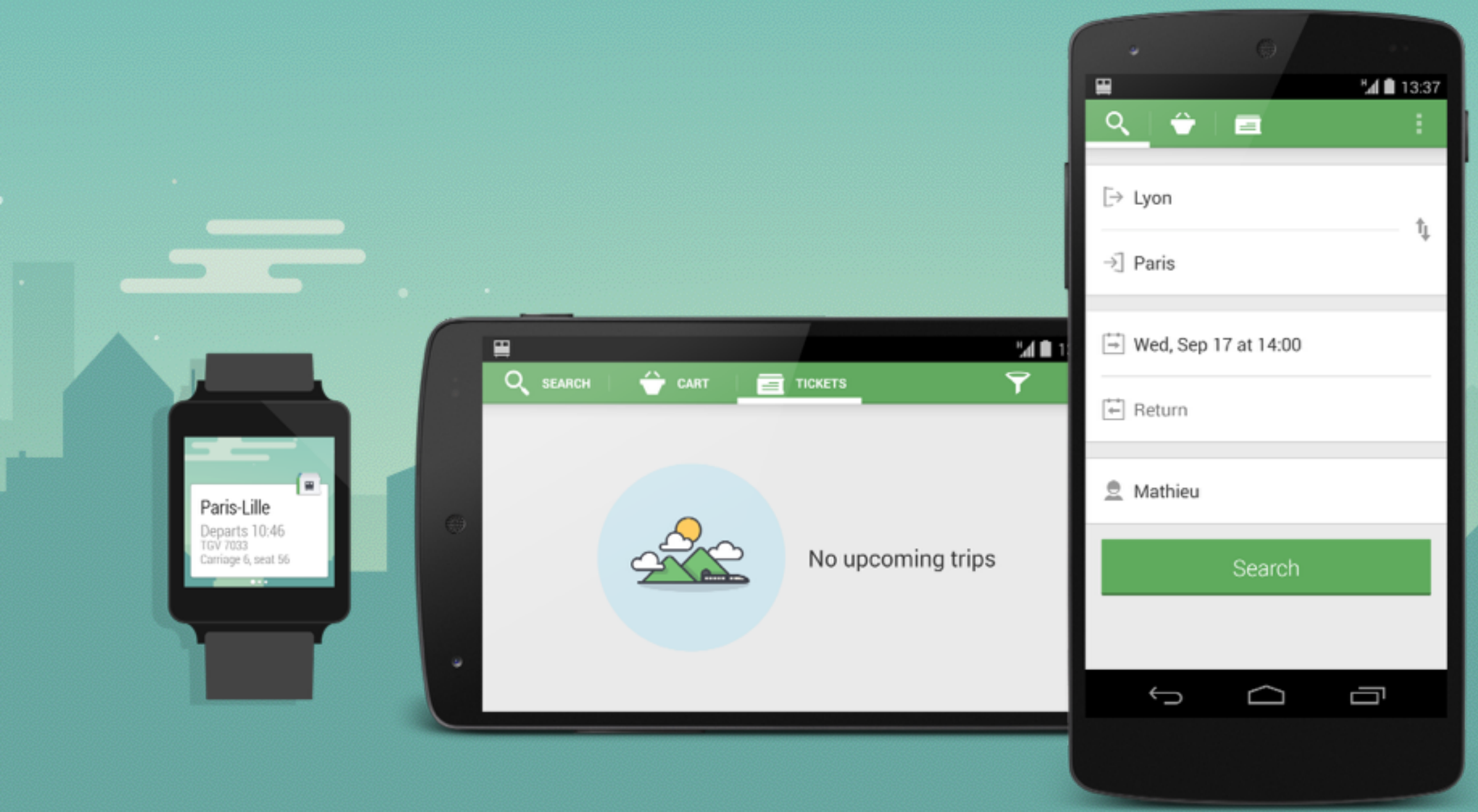
The Death of the Refresh Button

@Mathieu_Calba

+MathieuCalba

capitaine train

capitaine train



Once upon a time…

Julien was searching for an app to manage his bank account

But he couldn’t find the one
that meets his needs

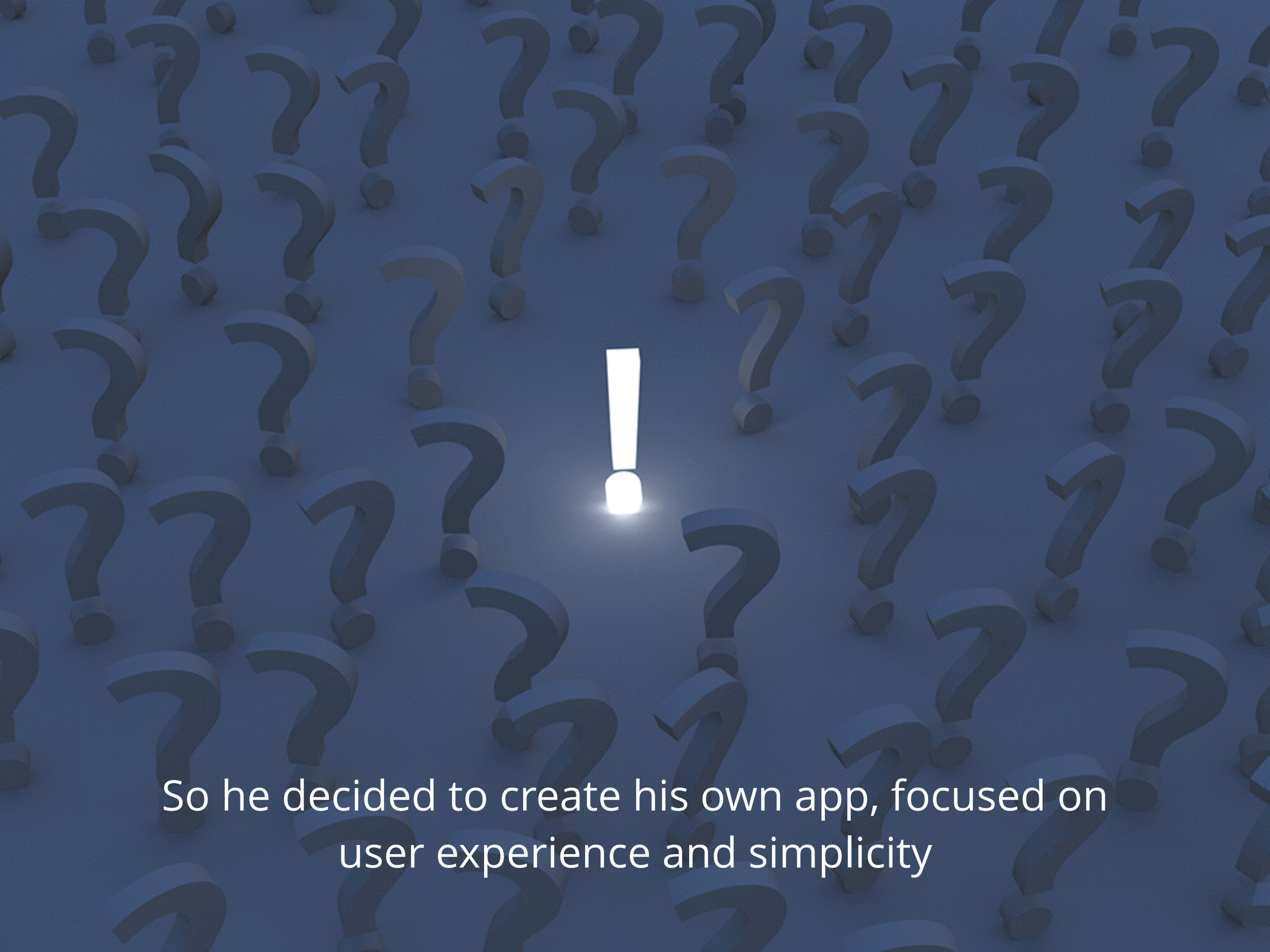

They all seemed
broken

They always needed an internet connection

Even just to check his leisure budget

Vincent was very **ANGRY**


He needed a simple debit/credit managing app with a good backup

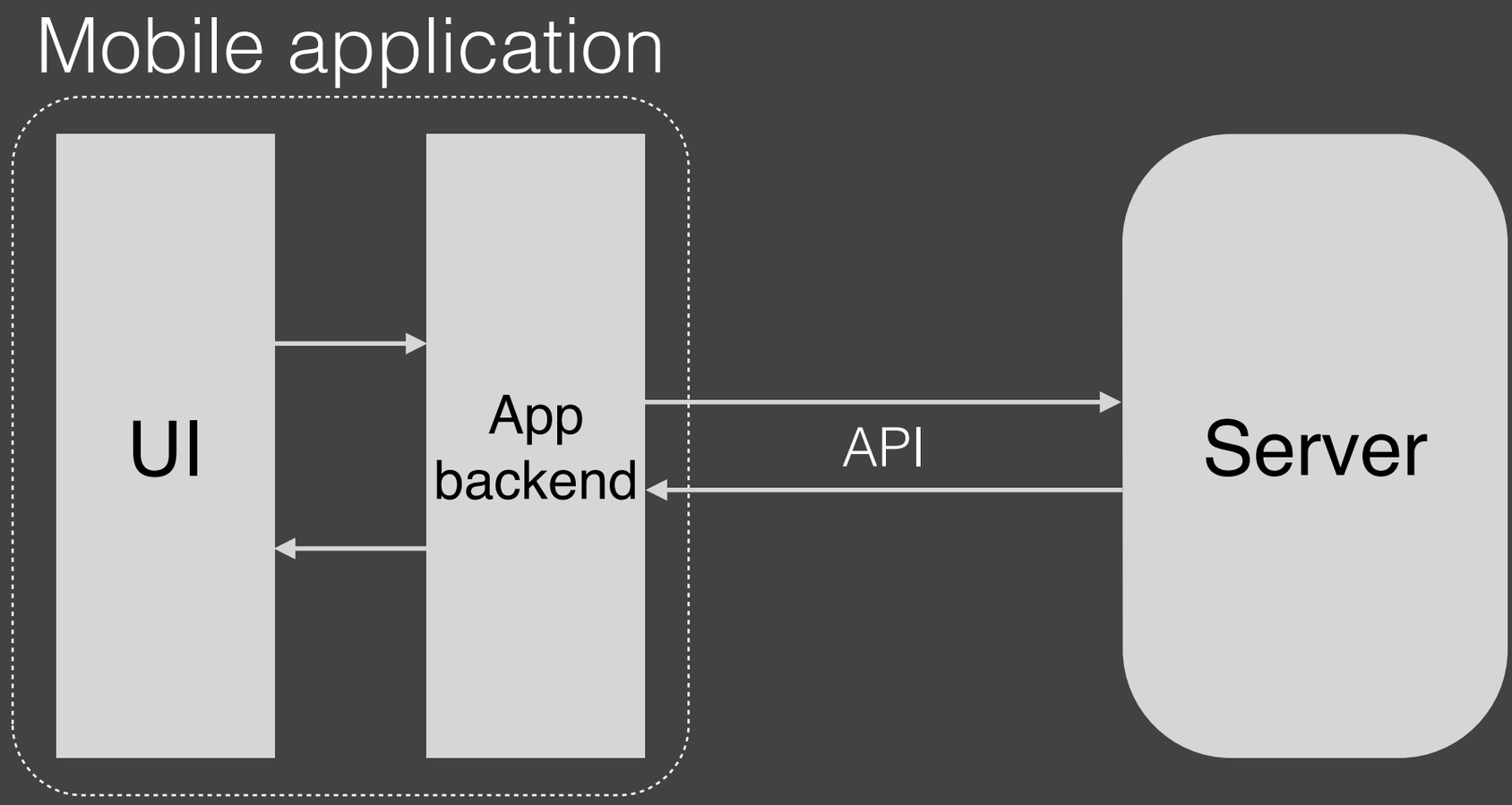


So he decided to create his own app, focused on user experience and simplicity

What does his app need?

What does his app need?

A great app backend


Mobile application
UI
App backend
API
Server

With a great app backend,
you have more **flexibility**
at the UI level

What is syncing?

What is syncing?

Executing a

BACKGROUND JOB

to update local data

WHEN APPROPRIATE

BACKGROUND JOB

- Non-user facing operation
- Upload and/or download
- Requires network connectivity
- Always leave the data in good state

What is syncing?

Executing a

BACKGROUND JOB

to update local data

WHEN APPROPRIATE

WHEN APPROPRIATE

aka scheduling & triggering

- Only when needed, and when relevant (network availability, battery level, etc)
- Restore after reboot
- Exponential back-off
- Be careful with battery life
- etc.

WHEN APPROPRIATE

aka scheduling & triggering

- Best solution: push message
- Not always available, fallback: periodic polling

What is syncing?

Executing a

BACKGROUND JOB

to update local data

WHEN APPROPRIATE

How?

How?

The Android way

How?

The Android ways

AsyncTask & AlarmManager

Create your `AsyncTask`:

```
public final class PeriodicTaskRunnable extends AsyncTask<Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... voids) {
        // TODO the task !
        return null;
    }
}
```

Start the periodic run with the `AlarmManager`:

```
private void triggerPeriodicAsyncTask() {
    final Intent intent = new Intent(this, PeriodicTaskReceiver.class);

    final PendingIntent receiverPendingIntent = PendingIntent.
                getBroadcast(this, 140916, intent, PendingIntent.FLAG_UPDATE_CURRENT);

    final AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
    alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME, //
            SystemClock.elapsedRealtime() + AlarmManager.INTERVAL_HOUR, //
            AlarmManager.INTERVAL_HOUR, //
            receiverPendingIntent);
}
```

AsyncTask & AlarmManager

Create your `BroadcastReceiver`:

```
public class PeriodicTaskReceiver extends BroadcastReceiver {

    public static final String ACTION_SYNC = “com.mathieucalba.tasks.ACTION_SYNC”;

    @Override
    public void onReceive(Context context, Intent intent) {
        if (ACTION_SYNC.equals(intent.getAction())) {
            new PeriodicTaskRunnable().execute();
        }
    }
}
```

And declare it in the `Manifest`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mathieucalba.testjobscheduler">
    <application>

        <receiver
            android:name=".tasks.PeriodicTaskReceiver"
            android:exported="false">
            <intent-filter>
                <action android:name="com.mathieucalba.tasks.ACTION_SYNC" />
            </intent-filter>
        </receiver>

    </application>
</manifest>
```

AsyncTask & AlarmManager

- `AsyncTask` & `Receiver` can be replaced by an `IntentService` for simplicity
- Restoring periodic sync after a reboot?
- Handling connectivity availability?
- Indicate sync state to others
- etc.

SyncAdapter

Definition

One method to do the

UPLOAD & DOWNLOAD SYNC

(`SyncAdapter`)

for one DATA PROVIDER (`ContentProvider`)

associated with a USER ACCOUNT

(`AccountAuthenticator`)

which can be TRIGGERED MANUALLY
or by the SYSTEM.

1- Account

```
public final class AccountAuthenticator extends AbstractAccountAuthenticator {

    public static final String ACCOUNT_TYPE = "com.mathieucalba.testjobscheduler";
    public static final String ACCOUNT_NAME_SYNC = "com.mathieucalba.testjobscheduler";

    public AccountAuthenticator(Context context) {
        super(context);
    }

    @Override
    public Bundle addAccount(AccountAuthenticatorResponse response, String accountType,
                             String authTokenType, String[] requiredFeatures, Bundle options)
                             throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle confirmCredentials(AccountAuthenticatorResponse response, Account account,
                                     Bundle options) throws NetworkErrorException {
        return null;
    }

    @Override
    public Bundle editProperties(AccountAuthenticatorResponse response, String accountType) {
        throw new UnsupportedOperationException();
    }
    //…
}
```

1- Account

```
public final class AccountAuthenticator extends AbstractAccountAuthenticator {
    //…
    @Override
    public Bundle getAuthToken(AccountAuthenticatorResponse response, Account account,
                               String authTokenType, Bundle options) throws NetworkErrorException {
        throw new UnsupportedOperationException();
    }

    @Override
    public String getAuthTokenLabel(String authTokenType) {
        throw new UnsupportedOperationException();
    }

    @Override
    public Bundle hasFeatures(AccountAuthenticatorResponse response, Account account,
                              String[] features) throws NetworkErrorException {
        throw new UnsupportedOperationException();
    }

    @Override
    public Bundle updateCredentials(AccountAuthenticatorResponse response, Account account,
                                    String authTokenType, Bundle options)
                                    throws NetworkErrorException {
        throw new UnsupportedOperationException();
    }
}
```

1- Account

Bind the `AccountAuthenticator` to the framework:

```
public final class AccountAuthenticatorService extends Service {

    private static final Object LOCK = new Object();

    private static AccountAuthenticator sAuthenticator;

    @Override
    public void onCreate() {
        super.onCreate();
        synchronized (LOCK) {
            if (sAuthenticator == null) {
                sAuthenticator = new AccountAuthenticator(getApplicationContext());
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        synchronized (LOCK) {
            return sAuthenticator.getIBinder();
        }
    }
}
```

1- Account

Declare your `AccountAuthenticatorService` into the `Manifest`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mathieucalba.testsyncadapter">
   <application>

        <service
            android:exported="true"
            android:name=".accounts.AccountAuthenticatorService">
            <intent-filter>
                <action android:name="android.accounts.AccountAuthenticator" />
            </intent-filter>

            <meta-data
                android:name="android.accounts.AccountAuthenticator"
                android:resource="@xml/account_authenticator" />
        </service>

    </application>
</manifest>
```

Configure your `AccountAuthenticator`:

```
<?xml version="1.0" encoding="utf-8"?>
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="@string/config_accountType"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:smallIcon="@drawable/ic_launcher" />
```

1- Account

Add the `Account` required by the framework:

```
private void initAccountAuthenticator() {
    final AccountManager accountManager = AccountManager.get(this);
    final Account[] accounts = accountManager.getAccountsByType(AccountAuthenticator.ACCOUNT_TYPE);
    for (Account account : accounts) {
        if (AccountAuthenticator.ACCOUNT_NAME_SYNC.equals(account.name)) {
            return;
        }
    }

    accountManager.addAccountExplicitly(new Account(AccountAuthenticator.ACCOUNT_NAME_SYNC,
            AccountAuthenticator.ACCOUNT_TYPE), null, null);
}
```

For more info on implementing an AccountAuthenticator: http://udinic.wordpress.com/2013/04/24/write-your-own-android-authenticator/

2- ContentProvider

Create a stub `ContentProvider`:

```
public class StubProvider extends ContentProvider {

    @Override
    public boolean onCreate() { return true; }

    @Override
    public String getType(Uri uri) { return new String(); }

    @Override
    public Cursor query(Uri uri, String[] strings, String s, String[] strings2, String s2) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues contentValues) { return null; }

    @Override
    public int update(Uri uri, ContentValues contentValues, String s, String[] strings) { return 0; }

    @Override
    public int delete(Uri uri, String s, String[] strings) { return 0; }

}
```

2- ContentProvider

Declare your `ContentProvider` in the `Manifest`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mathieucalba.testsyncadapter">

    <application>

        <provider
            android:authorities="com.mathieucalba.testsyncadapter.provider.stub"
            android:exported="false"
            android:name=".provider.StubProvider"
            android:syncable="true" />

    </application>

</manifest>
```

For more details about `ContentProvider`s,
see my slides at http://bit.ly/ContentProvider

3- SyncAdapter

Implement the synchronization mechanism:

```
public class SyncAdapter extends AbstractThreadedSyncAdapter {

    public SyncAdapter(Context context) {
        super(context, false, false); // Context, auto initialize, parallel sync
    }

    @Override
    public void onPerformSync(Account account, Bundle extras, String authority,
                              ContentProviderClient provider, SyncResult syncResult) {
        // TODO the sync
    }

}
```

3- SyncAdapter

Bind the `SyncAdapter` to the framework:

```
public class SyncAdapterService extends Service {
    private static final Object LOCK = new Object();
    private static SyncAdapter sSyncAdapter = null;
    @Override
    public void onCreate() {
        super.onCreate();
        synchronized (LOCK) {
            if (sSyncAdapter == null) {
                sSyncAdapter = new SyncAdapter(getApplicationContext());
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        synchronized (LOCK) {
            return sSyncAdapter.getSyncAdapterBinder();
        }
    }
}
```

3- SyncAdapter

Declare your `SyncAdapterService` into the `Manifest`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mathieucalba.testsyncadapter">

    <application>

        <service
            android:exported="true"
            android:name=".sync.SyncAdapterService">
            <intent-filter>
                <action android:name="android.content.SyncAdapter" />
            </intent-filter>

            <meta-data
                android:name="android.content.SyncAdapter"
                android:resource="@xml/sync_adapter" />
        </service>

    </application>

</manifest>
```

3- SyncAdapter

Configure your `SyncAdapter`:

```
<?xml version="1.0" encoding="utf-8"?>
<sync-adapter xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="@string/config_accountType"
    android:allowParallelSyncs="false"
    android:contentAuthority="com.mathieucalba.testsyncadapter.provider.stub"
    android:isAlwaysSyncable="true"
    android:supportsUploading="true"
    android:userVisible="false" />
```

4- Triggering

How to trigger a one time sync:

```
private static void triggerSyncAdapter(Account account, boolean now) {
    final Bundle extras = new Bundle();

    if (now) {
        // ignore backoff && settings
        extras.putBoolean(ContentResolver.SYNC_EXTRAS_MANUAL, true);
        // put the request at the front of the queue
        extras.putBoolean(ContentResolver.SYNC_EXTRAS_EXPEDITED, true);
    }

    ContentResolver.requestSync(account, StubProvider.AUTHORITY, extras);
}
```

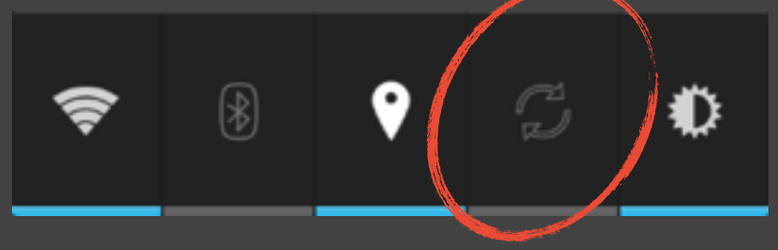
4- Triggering

How to configure an account for periodic sync, auto sync when network is up:

```
public static void setupSync(Account account) {
    if (account == null) {
        return;
    }

    // Inform the system that this account supports sync
    ContentResolver.setIsSyncable(account, StubProvider.AUTHORITY, 1);
    // Inform the system that this account is eligible for auto sync when the network is up
    ContentResolver.setSyncAutomatically(account, StubProvider.AUTHORITY, true);
    // Recommend a schedule for automatic synchronisation. The system may modify this based
    // on other scheduled syncs and network utilisation.
    ContentResolver.addPeriodicSync(account, //
                                    StubProvider.AUTHORITY, //
                                    new Bundle(), //
                                    TimeUnit.HOURS.toSeconds(1));
}
```

Periodic & automatic sync doesn't works if user has deactivated it



5- ContentProvider Triggering

One neat functionality: run the `SyncAdapter` when `ContentProvider`'s data changes

5- ContentProvider Triggering

Mark the item to be deleted instead of deleting it:

```
@Override
public int delete(Uri uri, String s, String[] strings) {
    final int count;
    if (StubContract.hasNeedSyncToNetworkParameter(uri)) {
        final int match = URI_MATCHER.match(uri);
        switch (match) {
            case OPTION_ID:
                final ContentValues values = new ContentValues();
                values.put(SyncColumns.SYNC_DELETED, SyncColumns.SYNC_DELETED_MARKED);
                count = buildSimpleSelection(uri).
                        where(selection, selectionArgs).
                        update(getDatabaseHelper().getWritableDatabase(), values);
                break;

            default:
                throw new SyncToNetworkUnknownUriException(uri);
        }

    } else {
        count = buildSimpleSelection(uri).
                where(selection, selectionArgs).
                delete(getDatabaseHelper().getWritableDatabase());
    }

    if (count > 0) {
        notifyChange(uri);
    }
    return count;
}
```

5- ContentProvider Triggering

Notify the change to the SyncAdapter:

```
private void notifyChange(Uri uri) {
    getContext().getContentResolver().
                    notifyChange(uri, null, isCallerUriUploadReady(uri) && !isCallerSyncAdapter(uri));
}

public boolean isCallerSyncAdapter(Uri uri) {
    return StubContract.hasCallerIsSyncAdapterParameter(uri);
}

private boolean isCallerUriUploadReady(Uri uri) {
    if (StubContract.hasNeedSyncToNetworkParameter(uri)) {
        final int match = URI_MATCHER.match(uri);
        switch (match) {
            case OPTION_ID:
                return true;

            default:
                throw new SyncToNetworkUnknownUriException(uri);
        }
    }
    return false;
}
```

5- ContentProvider Triggering

How we detect an `URI` if from the `SyncAdapter`, and when a `SyncAdapter` trigger is needed:

```
interface SyncExtras {
    String IS_CALLER_SYNC_ADAPTER = "is_caller_sync_adapter";

    String NEED_SYNC_TO_NETWORK = "need_sync_to_network";
}

public static Uri addCallerIsSyncAdapterParameter(Uri uri) {
    return uri.buildUpon().appendQueryParameter(SyncExtras.IS_CALLER_SYNC_ADAPTER,
Boolean.toString(true)).build();
}

public static boolean hasCallerIsSyncAdapterParameter(Uri uri) {
    final String parameter = uri.getQueryParameter(SyncExtras.IS_CALLER_SYNC_ADAPTER);
    return parameter != null && Boolean.parseBoolean(parameter);
}

public static Uri addNeedSyncToNetworkParameter(Uri uri) {
    return uri.buildUpon().appendQueryParameter(SyncExtras.NEED_SYNC_TO_NETWORK,
Boolean.toString(true)).build();
}

public static boolean hasNeedSyncToNetworkParameter(Uri uri) {
    final String parameter = uri.getQueryParameter(SyncExtras.NEED_SYNC_TO_NETWORK);
    return parameter != null && Boolean.parseBoolean(parameter);
}
```

5- ContentProvider Triggering

Be careful, your corresponding query should not return the deleted data if it’s not the `SyncAdapter` querying.

SyncAdapter

- Google way
- Easily triggered by the system when appropriate (network availability, change in associated ContentProvider)

- Needs an AccountAuthenticator (at least a stub)
- Needs a ContentProvider (at least a stub)

JobScheduler

Definition

SCHEDULE the execution of a JOB (via a `Service`)

with VARIOUS PARAMETERS

about WHEN

the execution SHOULD HAPPEN

(during a window of time, periodically, with network needed, etc.)

1- JobService

The `JobService` is where we are awaken:

```
public class MyJobService extends JobService {

    private ExecutorService mExecutor;
    private final Handler mHandler = new Handler(Looper.getMainLooper());

    @Override
    public void onCreate() {
        super.onCreate();
        mExecutor = Executors.newSingleThreadExecutor();
    }

    @Override
    public void onDestroy() {
        mExecutor.shutdown();
        super.onDestroy();
    }

    @Override
    public boolean onStartJob(JobParameters jobParameters) {
        // We are on the Main thread, so post the Task to a background thread
        mExecutor.execute(new Task(jobParameters));
        return true;
    }

    @Override
    public boolean onStopJob(JobParameters jobParameters) {
        // TODO interrupt Task
        return true;
    }
}
```

1- JobService

```
private final class Task implements Runnable {

    private final JobParameters mJobParameters;

    private Task(JobParameters jobParameters) { mJobParameters = jobParameters; }

    @Override
    public void run() {
        // TODO the network call
        mHandler.post(new FinishedTask(mJobParameters, true));
    }
}

private final class FinishedTask implements Runnable {

    private final JobParameters mJobParameters;
    private final boolean mIsSuccess;

    private FinishedTask(JobParameters jobParameters, boolean isSuccess) {
        mJobParameters = jobParameters;
        mIsSuccess = isSuccess;
    }

    @Override
    public void run() {
        // Notify that the job has ended
        jobFinished(mJobParameters, mIsSuccess);
    }
}
```

1- JobService

Declare your `MyJobService` into the `Manifest`:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mathieucalba.testjobscheduler">

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

    <application>

        <service
            android:name=".tasks.MyJobService"
            android:permission="android.permission.BIND_JOB_SERVICE"
            android:exported="true"/>

    </application>

</manifest>
```

2- Triggering

Exemple for triggering a one time Job:

```
JobInfo.Builder builder = new JobInfo.Builder(JOB_ID_1, new ComponentName(this, MyJobService.class)).
                        setBackoffCriteria(TimeUnit.MINUTES.toMillis(1),
                                           JobInfo.BackoffPolicy.EXPONENTIAL).
                        setMinimumLatency(TimeUnit.SECONDS.toMillis(5)).
                        setOverrideDeadline(TimeUnit.HOURS.toMillis(1)).
                        setRequiredNetworkCapabilities(JobInfo.NetworkType.ANY);

((JobScheduler) getSystemService(Context.JOB_SCHEDULER_SERVICE)).schedule(builder.build());
```

Exemple for triggering a periodic Job:

```
builder = new JobInfo.Builder(JOB_ID_PERIODIC, new ComponentName(this, MyJobService.class)).
                        setPeriodic(TimeUnit.HOURS.toMillis(1)).
                        setRequiredNetworkCapabilities(JobInfo.NetworkType.UNMETERED).
                        setRequiresDeviceIdle(true).
                        setRequiresCharging(true);

((JobScheduler) getSystemService(Context.JOB_SCHEDULER_SERVICE)).schedule(builder.build());
```

JobScheduler

- Configurable scheduling (idle-mode, network availability, etc) across all the system
- Simple, based upon a Service
- Persisted state

- Android-L+ only

Conclusion

Conclusion

- `SyncAdapter` perfect if account and `ContentProvider`, great otherwise but can be tricky
- `JobScheduler` very promising, but Android L+ only, a limited compat library would help spread its use (based upon `Service` & `AlarmManager`).

Conclusion

- We have the tools to create an app the works seamlessly without an internet connection so the user never have to worry about his internet connection

Questions?

Credits

- *Up & Down icons* by Guillaume Berry
- *The Search Underneath the Bed* by Arielle Nadel
- *Yorkshire Moors Milky Way 3* by Matthew Savage
- *Injured Piggy Bank With Crutches* by Ken Teegardin
- *No Internet* by Marcelo Graciolli
- *Balancing The Account By Hand* by Ken Teegardin
- *Rage* by Amy McTigue
- *Daniel Foster* by Online Shopping
- *3D Bright Idea* by Chris Potter
- *Landscape* by Ben Simo
- *Christmas Eve Sunrise* by Jay Parker