

Compte Rendu

Mini Projet Développement Mobile

I- Introduction

Notre groupe, Kharrat Omar et Medini Mohamed Nour, ont créé une petite application mobile permettant de lister les contacts, les modifier, les supprimer et les appeler.

Chaque utilisateur possède sa propre liste de contacts.

Un Utilisateur A ne peut ni voir ni modifier les contacts de l'utilisateur B.

Cette application est connectée sur une base de données distante grâce au site web www.000webhost.com en utilisant des requêtes SQL intégré des fichiers PHP aussi téléversé sur le même site web.

L'application peut être téléchargée depuis [ici](#) (nhot lien hypertexte ala ici)

Nom De Paquet: omaretnour.tn.afinal

II- Gradle

-l'ajout de dépôt "jitpack" en ajoutant `jcenter()`

```
maven { url 'https://jitpack.io' }
```

dans le fichier `settings.gradle`

```
dependencyResolutionManagement { DependencyResolutionManagement it ->
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories { RepositoryHandler it ->
        google()
        jcenter()
        maven { url 'https://jitpack.io' }
        mavenCentral()
    }
}
```

-On a ajouté les Dependancies suivants au fichier `build.gradle` (de l'application):

```
implementation 'com.ebanx:swipe-button:0.4.0'  
implementation 'com.github.hajiyevelnur92:intentanimation:1.0'
```

```
dependencies {  
  
    implementation 'androidx.appcompat:appcompat:1.5.1'  
    implementation 'com.google.android.material:material:1.7.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'  
    testImplementation 'junit:junit:4.13.2'  
    implementation 'com.ebanx:swipe-button:0.4.0'  
    implementation 'com.github.hajiyevelnur92:intentanimation:1.0'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.4'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.0'  
  
}
```

La 1ere pour le fonctionnement de swipe button a l'ouverture de l'application.
La 2eme pour le fonctionnement de la 'SPLASHSCREEN' avec une petite animation fadein to fadeout

-L'ajout de la ligne `android.enableJetifier=true` au fichier `gradle.properties`
Jetifier convertira le support de bibliothèques de l'ensemble des dépendances d'Android X automatiquement.

```
android.useAndroidX=true  
# Enables namespaceing of each library's R class so that its R class includes only the  
# resources declared in the library itself and none from the library's dependencies,  
# thereby reducing the size of the R class for that library  
android.nonTransitiveRClass=true  
android.enableJetifier=true
```

III- Les XML(Layouts & Manifest)

1) AndroidManifest.xml

On a ajouté 3 permissions pour l'application.

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

La 1ere qui donne l'accès à l'internet pour envoyer/recevoir des informations depuis et vers la base de données.

Les 2 autres permissions pour pouvoir effectuer les appels.

On a ajouté aussi la ligne `android:usesCleartextTraffic="true"` pour permettre le traffic http.

-L'ajout des activités au manifest, avec "Swipe" l'activité principale

```
<activity
    android:name=".Swipe"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <meta-data
        android:name="android.app.lib_name"
        android:value="" />
</activity>
```

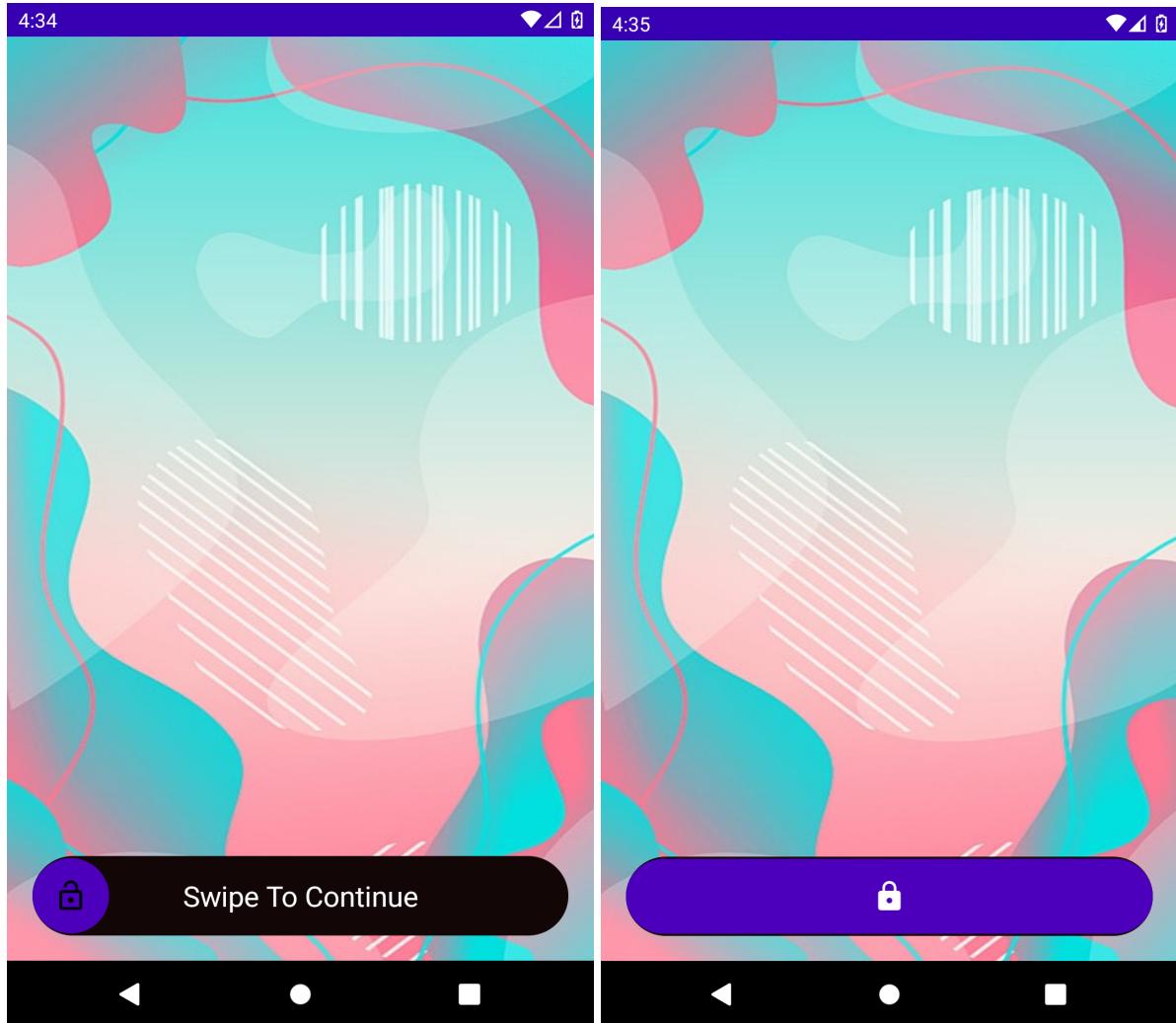
2) Layouts

Commençons par activity_swipe.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@drawable/bgg"
    tools:context=".Swipe">

    <com.ebanx.swipebtn.SwipeButton
        android:id="@+id/swipe_button"
        app:button_background="@drawable/button_shape"
        app:inner_text_background="@drawable/inner_text_shape"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        app:inner_text="Swipe To Continue"
        app:inner_text_size="20sp"
        app:inner_text_top_padding="15dp"
        app:inner_text_bottom_padding="15dp"
        app:button_image_disabled="@drawable/ic_baseline_lock_open_24"
        app:button_image_enabled="@drawable/ic_baseline_lock_24"
        app:button_bottom_padding="15dp"
        app:button_top_padding="15dp"
        app:button_right_padding="15dp"
        app:button_left_padding="15dp"
        android:layout_margin="18sp" />
</RelativeLayout>
```

Pour avoir une interface comme celle-ci:



1ère capture d'écran:
Lors de lancement de l'activité.

2ème capture d'écran:
Après le glissement du bouton

La photo de l'arrière-plan existe dans les “Drawables” bien sûr.
Bouton glissant existe aussi dans le même dossier.

Après le glissement du bouton on trouve un nouveau layout [activity_login_screen_app.xml](#)

Des EditText habituelles, pour insérer les coordonnées de login,
2 boutons, un pour se connecter, l'autre pour s'enregistrer.
Et finalement un ImageView pour le bouton afficher/cacher le mot de passe

```
<ImageView  
    android:id="@+id/show_pass_btn"  
    android:layout_width="40dp"  
    android:layout_height="40dp"  
    android:layout_alignParentRight="true"  
    android:layout_centerVertical="true"  
    android:layout_marginEnd="12dp"  
    android:layout_marginBottom="7dp"  
    android:alpha=".5"  
    android:onClick="ShowHidePass"  
    android:padding="5dp"  
    android:src="@drawable/ic_visibility_off"  
    app:layout_constraintBottom_toBottomOf="@+id/passwordEdit"  
    app:layout_constraintEnd_toEndOf="@+id/passwordEdit"  
    app:layout_constraintTop_toTopOf="@+id/passwordEdit"  
    app:layout_constraintVertical_bias="0.777" />
```

Si L'utilisateur ne possède pas de compte (il clique sur le bouton Sign Up)
pour trouver un layout [activity_sign_up.xml](#)

Contenant3 EditText (Username, Password, Confirming Password)
avec un petit bouton afficher/cacher le mot de passe.

Deux autres boutons: Retour. SignUp pour confirmer l'enregistrement.

Le clic sur ces deux boutons nous permettra d'accéder une autre fois vers la page login. [activity_login_screen_app.xml](#)

```
<ImageView  
    android:id="@+id/img"  
    android:layout_width="434dp"  
    android:layout_height="900dp"  
    android:layout_alignParentStart="true"  
    android:layout_alignParentTop="true"  
    android:layout_centerHorizontal="true"  
    android:layout_marginStart="1dp"  
    android:layout_marginTop="90dp"  
    android:src="@drawable/appssplash"  
    tools:ignore="ContentDescription" />
```

Après le remplissage des coordonnées de connexion et l'appui sur "Login"
On se trouvera dans le layout [activity_splashscreen.xml](#)
Où on trouvera une ImageView centrée horizontalement contenant le logo de notre Application mobile.

Pour finalement accéder la page principale `activity_show_contacts.xml`
On trouve une ListView et 4 boutons(Ajouter,Modifier,Supprimer,Appeler)

```
<ListView
    android:id="@+id/lst"
    android:layout_width="wrap_content"
    android:layout_height="510dp"
    android:layout_marginLeft="12dp"
    android:layout_marginTop="10dp"
    android:layout_marginRight="12dp"
    android:divider="@color/purple_700"
    android:dividerHeight="2sp"
    android:listSelector="#40FFFFFF"
    android:scrollbarSize="10sp"
    android:scrollbarStyle="outsideOverlay"
    android:scrollbars="vertical"
    android:soundEffectsEnabled="true"
    tools:listitem="@layout/item" />
```

En Utilisant le layout `item.xml` qui contient 2 TextView

```
<TextView
    android:id="@+id/nom"
    android:layout_width="397dp"
    android:layout_height="45dp"
    android:textColor="@color/purple_700"
    android:textSize="25sp"
    android:paddingLeft="8sp"
    android:textStyle="italic"/>

<TextView
    android:id="@+id/num"
    android:layout_width="392dp"
    android:layout_height="54dp"
    android:paddingLeft="8sp"
    android:layout_marginLeft="10sp"
    android:textColor="@color/purple_700"
    android:textSize="25sp"
    android:textStyle="italic" />
```

Une pour le nom de la personne, l'autre pour son numéro de téléphone.

Parmi les 4 boutons.

-Le clic sur le bouton ajouter utilise la layout **activity_add.xml**

Ce fichier layout contient au plus important 2 EditText que l'utilisateur va les remplir par un nom du contact et l'autre par son numéro de téléphone.

Un bouton pour effectuer l'ajout de ce contact.

Un bouton pour le retour vers le layout **activity_show_contacts.xml**

-Le clic sur le bouton modifier utilise la layout **activity_modify.xml**

Ce fichier layout contient au plus important 2 EditText qui contiennent déjà les anciens informations du contact à modifier.

Un bouton pour effectuer la modification.

Un bouton pour le retour vers le layout **activity_show_contacts.xml**

Finalement

Un fichier layout **admin_show.xml** qui contient seulement la ListView.

Ce layout est créé juste pour que l'administrateur puisse voir tous les contacts dans l'application.

3) Strings.xml

Le fichier strings contient les données ci dessous

```

<resources>
    <string name="app_name">Contacts Manager</string>
    <string name="login">Login</string>
    <string name="username">Username</string>
    <string name="password">Password</string>
    <string name="logbtn">Login</string>
    <string name="add">Add</string>
    <string name="delete">Delete</string>
    <string name="deletetxt">Delete</string>
    <string name="addtxt">Add</string>
    <string name="modifytx">Modify</string>
    <string name="m">m</string>
    <string name="sign_in">Sign Up</string>
    <string name="confirm_password">Confirm password</string>
    <string name="signbtn">Sign Up</string>
    <string name="create_user_profile">Need an account?\nNo Problem</string>
    <string name="bt">Sign Up</string>
    <string name="addbtn2">Add</string>
    <string name="phone_number">Phone Number</string>
    <string name="name">Name</string>
    <string name="add_contact">Add Contact</string>
    <string name="modify_contact">Modify Contact</string>
    <string name="new_phone_number">New Phone Number</string>
    <string name="new_name">New Name</string>
    <string name="modify">Modify</string>
    <string name="pref_file">com.example.preference_file</string>
    <string name="pref_login_status">com.example.preference_login_status</string>
    <string name="pref_user_name">com.example.preference_user_name</string>
    <string name="splash_text">App Made with ❤\n By Nour And Omar</string>
    <string name="call">Call</string>
</resources>

```

IV- La Base de données et les fichiers PHP

Notre base de données nommée **id19877147_samir** est hébergée comme on a déjà dit sur le site www.000webhost.com

La Base de données contient 2 Tables:

- contacts
- user

La table contacts possède cette structure:

| # | Nom | Type | Interclassement | Attributs | Null | Valeur par défaut | Commentaires | Extra | Action |
|---|------|-------------|-----------------|-----------|------|-------------------|--------------|----------------|--------|
| 1 | id | int(11) | | | Non | Aucun(e) | | AUTO_INCREMENT | |
| 2 | nom | varchar(20) | utf8_unicode_ci | | Non | Aucun(e) | | | |
| 3 | num | varchar(20) | utf8_unicode_ci | | Non | Aucun(e) | | | |
| 4 | user | int(11) | | | Non | Aucun(e) | | | |

La colonne user est une clé étrangère depuis la table user, cela pour mémoriser l'utilisateur qui a créé le contact.

La table user avec cette structure:

| # | Nom | Type | Interclassement | Attributs | Null | Valeur par défaut | Commentaires | Extra | Action |
|-----|---|-------------|-----------------|-----------|------|-------------------|--------------|----------------|---|
| □ 1 | id  | int(11) | | | Non | Aucun(e) | | AUTO_INCREMENT |  Modifier  Supprimer ▾ Plus |
| □ 2 | name | varchar(30) | utf8_unicode_ci | | Non | Aucun(e) | | |  Modifier  Supprimer ▾ Plus |
| □ 3 | password | varchar(20) | utf8_unicode_ci | | Non | Aucun(e) | | |  Modifier  Supprimer ▾ Plus |

Les fichiers PHP sont aussi hébergés sur le même site.

N.B: les fichiers contiennent des variables appelés success. qui permettent de savoir si les requêtes sont bien exécutés ou pas.

N.B: On a utilisé la méthode GET sur tous les fichiers.

N.B: Tout fichier PHP contenant que des requêtes SELECT ou/et INSERT ne seront pas discutées ici.

N.B: Tous les fichier PHP contiennent la ligne

```
include ("db_connect.php");
```

db_connect.php, ce fichier permet la liaison/connexion avec la base de données.

```
1 <?php
2
3 $cnx=mysqli_connect("localhost","id19877147_omarandnour","*****123");
4 if(!$cnx)
5 {
6     echo "erreur de connexion au serveur";
7 }
8
9
10 $db=mysqli_select_db($cnx,"id19877147_samir");
11 if(!$db)
12 {
13     echo "erreur de connexion à la base";
14 }
15
16
17 ?>
```

Server: localhost

User: id19877147_omarandnour

Password: *****

Database Name: id19877147_samir

delete.php contient comme important ce bout de code:

```
$contact_id=$_GET["contact_id"];
$req=mysqli_query($cnx, " delete from contacts where  id = '$contact_id' ");
if($req)
{
    $response["success"]=1;
    $response["message"]="successful delete";
    echo json_encode($response);
}
```

update.php contient comme important ce bout de code:

```
{  
    $contact=$_GET["contact"];  
    $nom=$_GET["nom"];  
    $num=$_GET["num"];  
    $req=mysqli_query($cnx, " update contacts set nom='$nom' , num='$num' where id='$contact' ");  
    if($req)  
    {  
        $response["success"]=1;  
        $response["message"]="updated successfully";  
        echo json_encode($response);  
    }  
    else  
    {  
        $response["success"]=0;  
        $response["message"]="request error";  
  
        echo json_encode($response);  
    }  
}
```

V- Code Java

Dans cette partie on va parler du code java de « Contacts Manager » . Cette partie présente le Backend de l'application où on manipule les données venant de la base de données distantes ou entrées par l'utilisateur de « Contacts Manager » et aussi la logique derrière les fonctionnalités de l'application.

La manipulation des données consiste à :

→Les recevoir à partir du serveur lié à la base de données sous forme d'un JSON OBJECT

→Extraire les données à utiliser à travers la classe java « JsonParser »

→Acheminer les données entre les différentes activités de l'application

→Afficher les données dans l'interface graphique

→Modifier ces données

→Supprimer les données

→Créer des objets à partir de ces données

→Envoyer les données encapsulées dans des objets à une base de données distantes

⇒ Cette application permet aussi de passer des appels vers les contacts stockés dans la base de données

1) L'échange de données avec la base de données :

Pour gérer la communication entre le serveur web contenant les fichiers PHP qui manipule la base de données, on a créé une classe JAVA appelée « JsonParser » .

JsonParser contient une seule méthode « makeHttpRequest() »

Cette méthode prend comme paramètres l'URL du serveur en question, la méthode utilisée par le « request » et une « HashMap »

Qui contient les données à échanger avec la base de données à travers les fichiers PHP

Cette méthode envoie le « request » vers le serveur et reçoit la réponse sous la forme d'un JSON OBJECT (clé : valeur)

Exemple d'un JSON OBJECT :

```
Setting binder bot got note:  
: result: {"id":"23","success":1}  
: result: {"success":1,"message":"inserted !"}  
:
```

Les « import statements » de la classe JsonParser :

```
package omaretnour.tn.afinal;
import android.util.Log;
import org.json.JSONException;
import org.json.JSONObject;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.util.HashMap;
```

La signature de la méthode « makeHttpRequest() » :

```
public JSONObject makeHttpRequest(String url, String method,
                                  HashMap<String, String> params) {
```

La formulation du « request » :

```
sbParams = new StringBuilder();
int i = 0;
for (String key : params.keySet()) {
    try {
        if (i != 0){
            sbParams.append("&");
        }
        sbParams.append(key).append("=");
        .append(URLEncoder.encode(params.get(key), charset));
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    i++;
}
```

Si la méthode utilisé dans le fichier PHP est « POST » :

```
if (method.equals("POST")) {
    // request method is POST
    try {
        urlObj = new URL(url);
        conn = (HttpURLConnection) urlObj.openConnection();
        conn.setDoOutput(true);
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Accept-Charset", charset);
        conn.setReadTimeout(10000);
        conn.setConnectTimeout(15000);
        conn.connect();
        paramsString = sbParams.toString();
        wr = new DataOutputStream(conn.getOutputStream());
        wr.writeBytes(paramsString);
        wr.flush();
        wr.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Si la méthode utilisée par le fichier PHP est « GET » :

```
else if(method.equals("GET")){
    // request method is GET
    if (sbParams.length() != 0) {
        url += "?" + sbParams.toString();
    }
    try {
        urlObj = new URL(url);
        conn = (HttpURLConnection) urlObj.openConnection();
        conn.setDoOutput(false);
        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept-Charset", charset);
        conn.setConnectTimeout(15000);
        conn.connect();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

La réception de la réponse de serveur :

```
try {
    //Receive the response from the server
    InputStream in = new BufferedInputStream(conn.getInputStream());
    BufferedReader reader = new BufferedReader(new InputStreamReader(in));
    result = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        result.append(line);
    }
    Log.d( tag: "JSON Parser", msg: "result: " + result.toString());
} catch (IOException e) {
    e.printStackTrace();
}
conn.disconnect();
```

La conversion du l'InputStream en un objet JSON :

```
// try parse the string to a JSON object
try {
    jObj = new JSONObject(result.toString());
} catch (JSONException e) {
    Log.e( tag: "JSON Parser", msg: "Error parsing data " + e.toString());
}
// return JSON Object
return jObj;
```

2) L'encapsulation des données d'un contact :

Pour stocker les données de chaque contact enregistré on doit instancier un objet de la classe Contact.

La classe Contact :

```
package omaretnour.tn.afinal;

public class Contact {
    private String user ;
    private String nom ;
    private String num ;

    public Contact(String user, String nom, String num) {
        this.user = user;
        this.nom = nom;
        this.num = num;
    }

    public String getUser() { return user; }

    public void setUser(String user) { this.user = user; }

    public String getNom() { return nom; }

    public void setNom(String nom) { this.nom = nom; }

    public String getNum() { return num; }

    public void setNum(String num) { this.num = num; }
}
```

3) Afficher les données dans l'interface graphique :

On a créé une classe ShowContacts qui va gérer l'affichage des contacts de l'utilisateur dans un ListView .

Les contacts sont stockés dans un ArrayList qui prend seulement des objets de type Contact et on utilise un ArrayAdapter qui prend en paramètres l'ArrayList pour l'affichage.

L'ArrayAdapter :

```
package omaretnour.tn.afinal;

import ...

public class Adapter extends ArrayAdapter<Contact> {
    private Context context ;
    private int resource ;

    public Adapter(@NonNull Context context, int resource, @NonNull ArrayList<Contact> objects) {
        super(context, resource, objects);
        this.context = context ;
        this.resource = resource;
    }

    @NonNull
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        LayoutInflator layoutInflater = LayoutInflator.from(context);
        convertView = layoutInflater.inflate(resource,parent, attachToRoot: false);

        TextView nom = (TextView) convertView.findViewById(R.id.nom);
        TextView num = (TextView) convertView.findViewById(R.id.num);
        nom.setText("Name : "+getItem(position).getNom());
        num.setText("Phone : "+getItem(position).getNum());

        return convertView ;
    }
}
```

Pour le Backend du l'affichage on a créé une classe ShowUserContacts qui hérite de la classe AsyncTask et effectue un redéfinition des méthodes « onPostExecute() » , « onPreExecute() » et « doInBackground() »

La méthode « onPreExecute() » :

```
@Override
protected void onPreExecute() {
    super.onPreExecute();
    dialog=new ProgressDialog( context: ShowContacts.this);
    dialog.setMessage("Please wait");
    dialog.show();
}
```

La méthode « doInBackground() » :

```
@Override
protected String doInBackground(String... strings) {
    HashMap<String, String> mp = new HashMap<>();
    mp.put("user", user);
    user_id_show = getIdHttpRequest(mp);
    HashMap<String, String> map = new HashMap<~>();
    map.put("user", String.valueOf(user_id_show));
    try {
        list_contacts = getContactsHttpRequest(map);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
```

On peut voir que doInBackground() fait appel à 2 méthodes «getIdHttpRequest() » et «getContactsHttpRequest() »
Ces deux méthodes prennent comme paramètres un HashMap qui contient les données à transférer vers la base de données dans la requête.

La méthode «getIdHttpRequest() » (retourne l'id de l'utilisateur actuel de l'application) :

```
public int getIdHttpRequest(HashMap<String, String> map) {
    int success=0,id =0;
    JSONObject object=parser.makeHttpRequest(
        url: "https://omarandnour.000webhostapp.com/and/user_id_get.php", method: "GET", map);
    try {
        success = object.getInt( name: "success");
        id = object.getInt( name: "id");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    if(success == 1)
        return id ;
    else
        return 0 ;
}
```

La méthode «getContactsHttpRequest() » (retourne un ArrayList d'objets de type Contact :

```
public ArrayList<Contact> getContactsHttpRequest(HashMap<String, String> map) throws JSONException {
    int success = 0;
    String message = "";
    ArrayList<Contact> list_contacts = new ArrayList<>();
    JSONObject object = parser.makeHttpRequest(
        url: "https://omarandnour.000webhostapp.com/and/select_one.php", method: "GET", map);
    try {
        success = object.getInt("success");
        message = object.getString("message");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    if (success == 1 || message.equals("no data found")) {

        JSONArray contacts = object.getJSONArray("contacts");
        for (int i = 0; i < contacts.length(); i++) {
            JSONObject contact = contacts.getJSONObject(i);
            list_contacts.add(new Contact(user, contact.getString("nom"), contact.getString("num")));
        }
    }
    return list_contacts;
}
```

La partie relié à l'affichage dans la méthode « onPostExecute() » :

```
@Override
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    dialog.cancel();

    Adapter adapter = new Adapter(getApplicationContext(), R.layout.item, list_contacts);
    ls.setAdapter(adapter);
```

Exemple d'affichage dans « Contacts Manager » :



4) La modification des données :

Pour modifier un des contacts présentés dans le ListView, l'utilisateur doit cliquer sur un « item » dans la liste.

Donc on doit utiliser « onItemClickListener() » pour qu'on capte le clic et surtout la position du l'item choisi .

Dans le « onItemClickListener() » on récupère les composantes graphiques (Pour qu'ils ne fonctionnent que lorsque on clique sur un des contacts)

Puis, dans le onClickListener du bouton de modification on crée un intent qui va être chargée avec le nom du contact à modifier et son numéro de téléphone , cette intent redirige l'utilisateur vers l'interface graphique de l'activité ModifyActivity .

```
ls.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        Button delete,modify,call;
        TextView adtx,deltx,modtx,calltx;
        itemToDeletePosition = i;
        call = (Button) findViewById(R.id.call) ;
        calltx = (TextView) findViewById(R.id.calltxt);
        deltx = (TextView) findViewById(R.id.txtdelete);
        modtx = (TextView) findViewById(R.id.txtmodify);
        delete = (Button) findViewById(R.id.delete);
        modify = (Button) findViewById(R.id.modify);
        modify.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent o = new Intent(getApplicationContext(),ModifyActivity.class);
                o.putExtra("nom",list_contacts.get(i).getNom());
                o.putExtra("num",list_contacts.get(i).getNum());
                o.putExtra("user",list_contacts.get(i).getUser());
                startActivity(o);
            }
        });
    }
});
```

Dans la classe ModifyActivity, On va créer une classe Modify qui hérite de AsyncTask et effectue une redéfinition de doInBackground() et onPostExecute()

La méthode « doInBackground() » :

```
@Override  
protected String doInBackground(String... strings)  
{  
    HashMap<String, String> mp = new HashMap<>();  
    mp.put("user", user);  
    id = getIdHttpRequest(mp);  
    HashMap<String, String> mpi = new HashMap<>();  
    mpi.put("user_id", String.valueOf(id));  
    mpi.put("nom", nom);  
    mpi.put("num", num);  
    contact_id = getContactIdHttpRequest(mpi);  
    HashMap<String, String> map = new HashMap<>();  
    map.put("num", phone.getText().toString());  
    map.put("nom", name.getText().toString());  
    map.put("contact", String.valueOf(contact_id));  
    success_modify = updateHttpRequest(map);  
    return null;  
}
```

La méthode getIdHttpRequest() retourne l'id de l'utilisateur de l'application (capture d'écran au-dessus)

La méthode getContactIdHttpRequest() retourne l'id du contact à modifier dans la base de données .

```
public int getContactIdHttpRequest(HashMap<String, String> map) {  
    int success=0, contact_id = 0 ;  
    JSONObject object = parser.makeHttpRequest(  
        url: "https://omarandnour.000webhostapp.com/and/contact_id.php", method: "GET", map);  
    try {  
        success = object.getInt( name: "success");  
        contact_id = object.getInt( name: "contact_id");  
    } catch (JSONException e) {  
        e.printStackTrace();  
    }  
    if(success == 1)  
        return contact_id ;  
    else  
        return 0 ;  
}
```

La méthode « updateHttpRequest() »:

```
public int updateHttpRequest(HashMap<String, String> map) {
    int success=0;
    JSONObject object = parser.makeHttpRequest(
        url: "https://omarandnour.000webhostapp.com/and/update.php", method: "GET", map);
    try {
        success = object.getInt( name: "success");

    } catch (JSONException e) {
        e.printStackTrace();
    }
    return success ;
}
```

La méthode « onPostExecute() »

```
@Override
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    updateStatusToaster(success_modify);
}
```

Cette méthode fait appel à « updateStatusToaster() » qui ,selon la réponse du serveur (success_modify) , affiche un Toast sur l'écran .

```
public void updateStatusToaster(int success) {
    if(success==0)
    {
        Toast.makeText( context: ModifyActivity.this, text: "Failed to modify",Toast.LENGTH_LONG).show();
    }
    else {
        Toast.makeText( context: ModifyActivity.this, text: "Contact modified",Toast.LENGTH_LONG).show();
    }
}
```

5) Suppression des données

Lorsque l'utilisateur clique sur un contact les fonctionnalités du bouton « delete » sont activées.

Lorsque l'on clique sur le bouton « delete » le contact sélectionné va être supprimé et le contenu de la ListView va être actualisé.

On réalise ça à travers la classe Supprimer qui hérite de la classe AsyncTask et implémente les méthodes « doInBackground() », « preExecute() » et « postExecute() » .

Dans doInBackground() on utilise la méthode getUserHttpRequest()

Pour récupérer l'id de l'utilisateur dans la base de données

Puis on utilise l'« user_id » , le nom du contact et le num pour identifier le contact_id du contact qu'on veut supprimer avec la méthode getContactIdHttpRequest().

Et enfin on passe le contact_id comme paramètre à la méthode deleteHttpRequest() pour la suppression du contact

La méthode « doInBackground() » :

```
class Delete extends AsyncTask<String, String, String>
{
    @Override
    protected String doInBackground(String... strings)
    {
        HashMap<String, String> mappy = new HashMap<>();
        mappy.put("user", user);
        user_id = getIdHttpRequest(mappy);
        Contact contact = list_contacts.get(itemToDeletePosition);
        HashMap<String, String> map = new HashMap<>();
        map.put("user_id", String.valueOf(user_id));
        map.put("nom", contact.getNom());
        map.put("num", contact.getNum());
        contact_id = getContactIdHttpRequest(map);
        HashMap<String, String> mapped = new HashMap<>();
        mapped.put("contact_id", String.valueOf(contact_id));
        delete_success = deleteHttpRequest(mapped);
        return null;
    }
}
```

La méthode « deleteHttpRequest() » :

```
public int deleteHttpRequest(HashMap<String, String> map) {  
    int success = 0 ;  
    JSONObject ob = parser.makeHttpRequest(  
        url: "https://omarandnour.000webhostapp.com/and/delete.php", method: "GET", map);  
    try {  
        success= ob.getInt( name: "success");  
    } catch (JSONException e) {  
        e.printStackTrace();  
    }  
    return success ;  
}
```

La méthode « postExecute() » :

```
@Override  
protected void onPostExecute(String s) {  
    super.onPostExecute(s);  
    deleteStatusToaster(delete_success);  
}
```

La méthode « deleteStatusToaster() » affiche un Toast en fonction du succès de suppression du contact :

```
public void deleteStatusToaster(int success) {  
    if(success==0)  
    {  
        Toast.makeText( context: ShowContacts.this, text: "Failed to delete contact",Toast.LENGTH_LONG).show();  
    }  
    else if(success==1)  
    {  
        Toast.makeText( context: ShowContacts.this, text: "Contact deleted",Toast.LENGTH_LONG).show();  
        finish();  
        startActivity(getIntent());  
    }  
}
```

6) Ajout des données (Contacts)

Pour ajouter des contacts à la base de données l'utilisateur appuie sur « Add » qui le renvoie vers l'interface de l'ajout de contact.

Cette interface est connectée à AddActivity ou le backend de la fonctionnalité de l'ajout prend place .

Le bouton « Add » :

```
add.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent i = new Intent(getApplicationContext(), Add.class);
        i.putExtra("name", "user");
        startActivity(i);
    }
});
```

Le onCreate() de l'activité Add :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add);
    title = findViewById(R.id.addTitle);
    phone = findViewById(R.id.phoneEdit);
    name = findViewById(R.id.nameEdit);
    add = findViewById(R.id.addButton);
    back = (Button) findViewById(R.id.back);
    user = (String) getIntent().getStringExtra("name");
    add.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            new AddContact().execute();
        }
    });
    back.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent intent = new Intent(getApplicationContext(), ShowContacts.class);
            intent.putExtra("name", "user");
            intent.putExtra("nom", name.getText().toString());
            intent.putExtra("num", phone.getText().toString());
            startActivity(intent);
            finish();
        }
    });
}
```

Le numéro de téléphone de contact a ajouté doit être composé de 8 chiffres. Ces contraintes sur les données entrées par l'utilisateur sans exigées . On réalise ce contrôle à travers la méthode checkPhoneNumber() :

```
    public boolean checkPhoneNumber(String num) {  
        return !num.trim().equals("") && num.trim().length() == 8;  
    }  
}
```

La requête http qui assure l'ajout du contact dans la base de données se trouve dans la classe « AddContact » qui hérite aussi de la classe AsyncTask et implémente ses méthodes :

La méthode doInBackground() :

```
@Override  
protected String doInBackground(String... strings) {  
    HashMap<String, String> map_user = new HashMap<~>();  
    map_user.put("user", user);  
    id = getIdHttpRequest(map_user);  
    HashMap<String, String> map = new HashMap<~>();  
    map.put("user", String.valueOf(id));  
    map.put("nom", name.getText().toString());  
    map.put("num", phone.getText().toString());  
    success = addHttpRequest(map);  
    return null;  
}
```

On utilise getIdHttpRequest() pour récupérer l'id de l'utilisateur Puis on utilise addHttpRequest() pour l'ajout , cette méthode prend comme paramètre une HashMap contenant le « user_id » le nom du contact à ajouter et son numéro de téléphone et retourne un entier success qui représente le succès de l'insertion du contact dans la base de données.

La méthode addHttpRequest() :

```
public int addHttpRequest(HashMap<String, String> map) {
    int success = 0 ;
    JSONObject object_add = parser.makeHttpRequest(
        url: "https://omarandnour.000webhostapp.com/and/add.php", method: "GET", map);
    try {
        success = object_add.getInt( name: "success");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return success;
}
```

Dans la méthode onPostExecute() on fait appel à addStatusToaster()
Qui selon le succès de l'insertion affiche un toast pour l'utilisateur

```
public void addStatusToaster(int success) {
    if(success==1)
    {
        Toast.makeText( context: Add.this,
                        text: "Contact added",Toast.LENGTH_LONG).show();
    }
    else
    {
        Toast.makeText( context: Add.this,
                        text: "Adding contact failed",Toast.LENGTH_LONG).show();
    }
}
```

Si l'utilisateur utilise le « native back button » du téléphone pour revenir la ListView va s'actualiser automatiquement pour afficher le nouveau contact même si l'utilisateur n'a pas utilisé le « back button » fourni par l'application ; cela est assuré par une redéfinition de méthode onBackPressed() :

```
@Override
public void onBackPressed() {
    Intent i = new Intent( packageContext: this, ShowContacts.class);
    i.putExtra( name: "refresh", value: true);
    i.putExtra( name: "user", user);
    i.putExtra( name: "nom", name.getText().toString());
    i.putExtra( name: "num", phone.getText().toString());
    startActivity(i);
    finish();
}
```

7) Ajout des données (Comptes Utilisateurs)

Pour pouvoir utiliser « Contacts Manager », l'utilisateur doit créer un compte à travers l'interface « Sign Up » puis utiliser le nom et le mot de passe du compte créé dans l'interface « Login » pour accéder a la liste des contacts . L'interface de signup est liée a la classe SignUp ou l'utilisateur doit choisir un nom d'utilisateur et un mot de passe et créer un compte. Le nom d'utilisateur choisi doit être unique. Donc l'application n'accepte pas un nom qui existe déjà .

La méthode onCreate() :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sign_in);
    back = (Button) findViewById(R.id.back);
    user = (EditText) findViewById(R.id.usernameedit);
    pwd = (EditText) findViewById(R.id.passwordedit);
    conf = (EditText) findViewById(R.id.confirmedit);
    title = (TextView) findViewById(R.id.signTitle);
    sign = (Button) findViewById(R.id.signButton);
    sign.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if (check(user.getText().toString(), pwd.getText().toString(), conf.getText().toString())) {
                new Signup().execute();
            }
        }
    });
    back.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent i = new Intent(getApplicationContext(), LoginScreenApp.class);
            startActivity(i);
            finish();
        }
    });
}
```

Le contrôle de saisie est réalisé par la méthode check () :

```
public boolean check(String name , String pass, String connf) {  
    if (name.trim().equals("")) {  
        Toast.makeText( context: this, text: "Type a username", Toast.LENGTH_SHORT).show();  
    }  
    else if (pass.trim().equals("")) {  
        Toast.makeText( context: this, text: "Please choose a password", Toast.LENGTH_SHORT).show();  
    }  
    else if (!pass.equals(connf)) {  
        Toast.makeText( context: this, text: "Please confirm your password", Toast.LENGTH_SHORT).show();  
    }  
    else {  
        return true ;  
    }  
    | return false ;  
}
```

Pour assurer la confidentialité de mot de passe on couvre le mot de passe que l'utilisateur tape mais il peut le voir en appuyant sur l'œil
La méthode showHidePass() :

```
public void ShowHidePass(View view) {  
  
    if (view.getId() == R.id.show_pass_bttn) {  
        if (pwd.getTransformationMethod().equals(PasswordTransformationMethod.getInstance())) {  
            ((ImageView) (view)).setImageResource(R.drawable.ic_visibility);  
            //Show Password  
            pwd.setTransformationMethod(HideReturnsTransformationMethod.getInstance());  
        } else {  
            ((ImageView) (view)).setImageResource(R.drawable.ic_visibility_off);  
            //Hide Password  
            pwd.setTransformationMethod(PasswordTransformationMethod.getInstance());  
        }  
    }  
}
```

Pour l'envoie de requête http pour l'insertion du nouveau compte dans la table « user » on crée une classe « Signup » qui hérite de AsyncTask et implémente les méthodes :

La méthode onPreExecute() :

```
@Override  
protected void onPreExecute() {  
    super.onPreExecute();  
    dialog=new ProgressDialog( context: SignUp.this);  
    dialog.setMessage("Please Wait");  
    dialog.show();  
}
```

La méthode doInBackground() :

```
@Override  
protected String doInBackground(String... strings) {  
    HashMap<String, String> mp=new HashMap<~>();  
    mp.put("nom", user.getText().toString());  
  
    addPossible = canAddUserHttpRequest(mp);  
    if(addPossible) {  
        HashMap<String, String> map = new HashMap<~>();  
        map.put("nom", user.getText().toString());  
        map.put("password", pwd.getText().toString());  
        success = signUpHttpRequest(map);  
    }  
    return null;  
}
```

La méthode onPostExecute() :

```
@Override  
protected void onPostExecute(String s) {  
    super.onPostExecute(s);  
    dialog.cancel();  
    signupToaster(success, addPossible);  
}
```

Pour déterminer si le nom choisi par l'utilisateur est déjà utilisé par un autre utilisateur ou non on a créé une méthode canAddUserHttpRequest() qui retourne un booléen (true pour la possibilité de l'ajout et false sinon) :

```
public boolean canAddUserHttpRequest(HashMap<String, String> map) {  
    int success=0,found=0;  
    JSONObject object=parser.makeHttpRequest(  
        url: "https://omarandnour.000webhostapp.com/and/select_all_user.php", method: "GET",map);  
    try {  
        success = object.getInt( name: "success");  
        found = object.getInt( name: "found");  
    } catch (JSONException e) {  
        e.printStackTrace();  
    }  
    if(found == 0)  
        return true ;  
    else  
        return false ;  
}
```

Si l'ajout de compte est possible on fait appel à signUpHttpRequest()

```
public int signUpHttpRequest(HashMap<String, String> map) {
    int success = 0;
    JSONObject object = parser.makeHttpRequest(
        url: ["https://omarandnour.000webhostapp.com/and/signin.php"], method: "GET", map);

    try {
        success = object.getInt("name: \"success\"");
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return success;
}
```

Cette méthode retourne un entier success , selon cet entier la méthode signUpToaster() qu'on appelle dans onPostExecute() affiche un toast sur l'écran :

```
public void signupToaster(int success, boolean addPossible) {
    if(!addPossible) {
        Toast.makeText(getApplicationContext(), text: "Username Already Exists", Toast.LENGTH_SHORT).show();
    }
    if(success==1)
    {
        Toast.makeText(context: SignUp.this, text: "Sign Up Success", Toast.LENGTH_LONG).show();
        Intent i = new Intent(packageContext: SignUp.this, LoginScreenApp.class);
        startActivity(i);
    }
    else
    {
        Toast.makeText(context: SignUp.this, text: "Sign Up Failure", Toast.LENGTH_LONG).show();
    }
}
```

Après le Sign Up l'utilisateur est redirigé vers l'interface de login ou il doit se connecter au compte créé pour accéder à ces données
Cette interface est liée à la classe Loginscreenapp .

Le contrôle de saisie sur les données entrée par l'utilisateur est réalisé par la méthode check() qui vérifie que l'utilisateur à rempli les champs nécessaire pour la connexion à son compte créé et affiche un toast pour avertir l'utilisateur:

```
public boolean check(String name, String pass) {
    if (name.trim().equals("")) {
        Toast.makeText(context: this, text: "Please type your username", Toast.LENGTH_SHORT).show();
    }
    else if (pass.trim().equals("")) {
        Toast.makeText(context: this, text: "Please type your password", Toast.LENGTH_SHORT).show();
    }

    else {
        return true;
    }

    return false;
}
```

Pour réaliser le login une requête http est envoyée vers le serveur qui à travers le fichier PHP correspondant vérifie l'existence de ce compte dans la table user de la base de données :

La méthode doInBackground() :

```
@Override  
protected String doInBackground(String... strings) {  
    HashMap<String, String> map=new HashMap<String, String>();  
    map.put("nom", user.getText().toString());  
    map.put("password", pass.getText().toString());  
    success = loginHttpRequest(map);  
    return null;  
}
```

La méthode loginHttpRequest() :

```
public int loginHttpRequest(HashMap<String, String> map) {  
    JSONObject object_login=parser.makeHttpRequest(  
        url: "https://omarandnour.000webhostapp.com/and/loqinqin.php", method: "GET", map);  
    try {  
        success=object_login.getInt( name: "success");  
  
    } catch (JSONException e) {  
        e.printStackTrace();  
    }  
    return success ;
```

La méthode loginStatusToaster() qui, selon le succès du login affiche un Toast sur l'écran :

```
public void loginStatusToaster(int success) {  
    if(success==1)  
    {  
        Intent i ;  
        Toast.makeText( context: LoginScreenApp.this, text: "Login Successful",Toast.LENGTH_LONG).show();  
        i = new Intent( packageContext: LoginScreenApp.this,Splashscreen.class);  
        i.putExtra( name: "user",user.getText().toString());  
        startActivity(i);  
        finish();  
    }  
    else  
    {  
        Toast.makeText( context: LoginScreenApp.this, text: "Login Failed",Toast.LENGTH_LONG).show();  
    }  
}
```

Si l'utilisateur veux quitter l'app il appuie sur le bouton de retour du téléphone une boîte de dialogue s'affiche sur l'écran

S'il appuie sur le bouton yes il quitte l'application sinon la boîte de dialogue se faire et il reste dans l'interface du login

Cette fonctionnalité est assurée par la redéfinition de la méthode onBackPressed() :

```
@Override
public void onBackPressed() {
    AlertDialog.Builder builder= new AlertDialog.Builder( context: this);
    builder.setTitle("Leaving The App :( ?")
        .setPositiveButton( text: "Yes", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {
                android.os.Process.killProcess(android.os.Process.myPid());
            }
        })
        .setNegativeButton( text: "No", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialogInterface, int i) {

            }
        });
    builder.create();
    builder.show();
}
```

Et finalement pour passer un appel téléphonique:

```
private void makePhoneCall() {
    String number = list_contacts.get(itemToDeletePosition).getNum();
    if (number.trim().length() > 0) {
        if (ContextCompat.checkSelfPermission( context: ShowContacts.this,
            Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions( activity: ShowContacts.this,
                new String[]{Manifest.permission.CALL_PHONE}, REQUEST_CALL);
        } else {
            String dial = "tel:" + number;
            startActivity(new Intent(Intent.ACTION_CALL, Uri.parse(dial)));
        }
    } else {
        Toast.makeText( context: ShowContacts.this, text: "Type a Number", Toast.LENGTH_SHORT).show();
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CALL) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            makePhoneCall();
        } else {
            Toast.makeText( context: this, text: "Permission DENIED", Toast.LENGTH_SHORT).show();
        }
    }
}
```

“Education is learning what you didn’t even know you didn’t know.”

—Daniel Boorstin