2022

NSI Prospect

App Android

OLLIVIER CHRIS – Lopes Jorys – Chabrier Mathis

# Page de service

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Auteur** | **Date** | **Description** |
| 1.0 | OLLIVIER Chris | 18/01/2022 | Création mise en page + |
| 1.1 | CHABRIER Mathis | 06/05/2022 | Ajout documentation API Siret |

Table des matiÈres

[Page de service 1](#_Toc104301219)

[Objet du document 3](#_Toc104301220)

[Contexte 3](#_Toc104301221)

[Gestion des utilisateurs 3](#_Toc104301222)

[Infrastructure Réseau 4](#_Toc104301223)

[Maquette 5](#_Toc104301224)

[CONNEXION utilisateur 5](#_Toc104301225)

[MENU principal 5](#_Toc104301226)

[Consultation, modification et saisie d'un prospect 7](#_Toc104301227)

[Ajout d’un prospect 7](#_Toc104301228)

[Modification d’un prospect 8](#_Toc104301229)

[Backend 9](#_Toc104301230)

[API PHP 9](#_Toc104301231)

[Base de donnée 11](#_Toc104301232)

[FrontEnd 12](#_Toc104301233)

[Classes API 13](#_Toc104301234)

[API Prospect 13](#_Toc104301235)

[API User 14](#_Toc104301236)

[API données sociétés 15](#_Toc104301237)

[Gestion de la base de données 16](#_Toc104301238)

[Class Prospect 20](#_Toc104301239)

[Class User 21](#_Toc104301240)

[View 22](#_Toc104301241)

[Login 22](#_Toc104301242)

[SignIn 24](#_Toc104301243)

[accueil 25](#_Toc104301244)

[AddProspect 26](#_Toc104301245)

[DisplayProspect 27](#_Toc104301246)

# Objet du document

Ce document contient des éléments d'analyse, de conception et de réalisation d'un outil de gestion des visiteurs. L'objectif est de conserver la trace et documenter le premier contact avec un prospect potentiel.

# Contexte

Fondée en 1988, NSI (Networking Solutions Incorporated) est une ESN (Entreprise de Services du Numérique). NSI possède les compétences et l'expérience pour intervenir sur la totalité du cycle de vie des solutions applicatives de ses clients : de l'analyse-conception jusqu'à l'exploitation en passant par la réalisation, le déploiement et la maintenance.

NSI est organisé autour de trois pôles de compétence :

1. Le développement d'applications pour ses clients.
2. L'édition logicielle en particulier pour les mobiles sous Android.
3. La conception et implémentation d'infrastructures informatiques.

# Gestion des utilisateurs

Mark TWAIN, le Directeur commercial de NSI, fait un constat très simple :

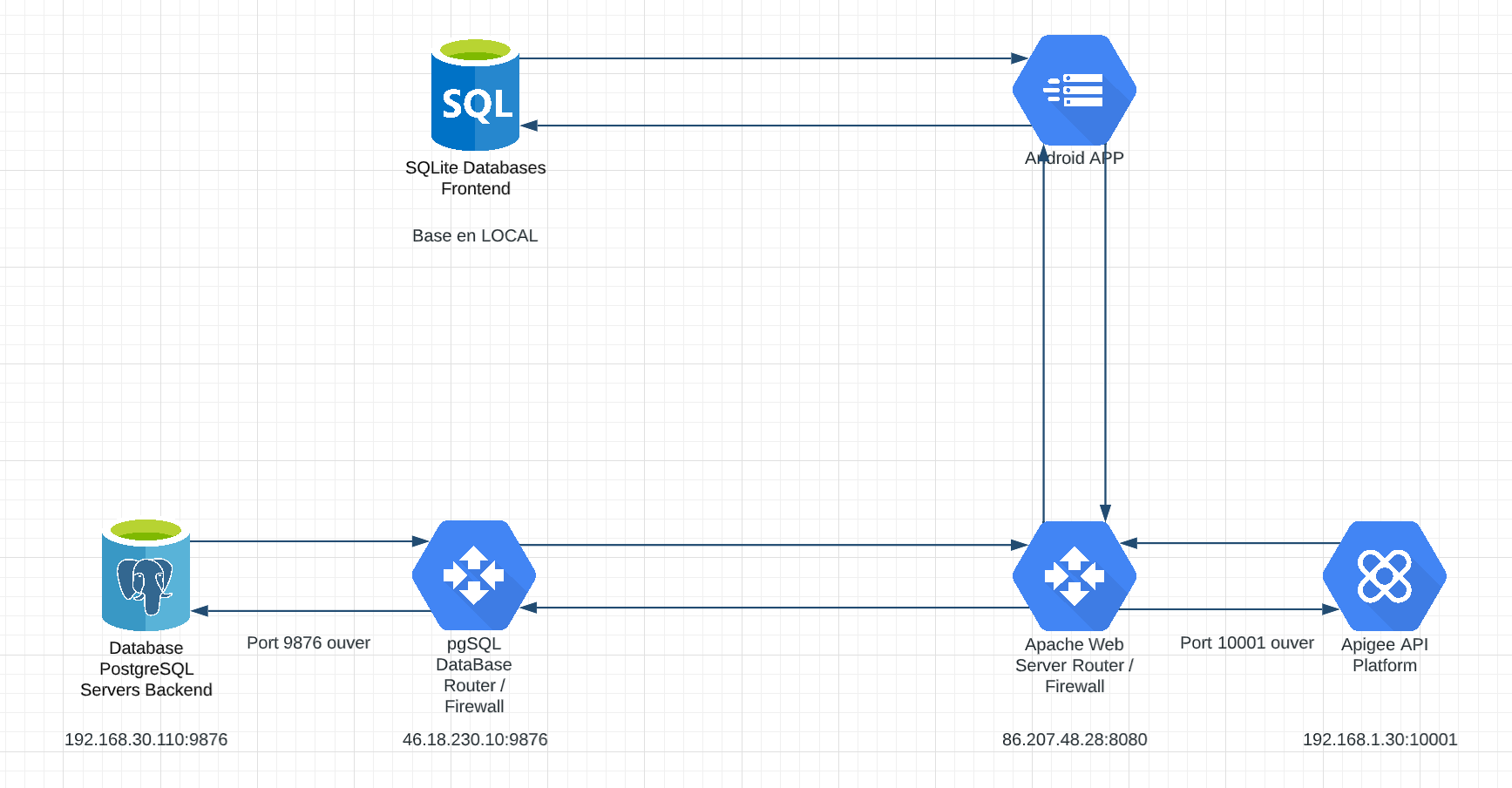
NSI expose son savoir-faire lors de salons professionnels. Ces salons, foires, expositions, congrès, forums, conférences sont l'occasion de rencontres informelles avec des représentants d'entreprises qui deviendront peut-être des prospects voire des clients.

Ces contacts sont purement informels et sous l'entière responsabilité du collaborateur qui réalise physiquement ce contact sans que jamais NSI n'en soit informé. Concrètement et dans le meilleur des cas, l'ingénieur commercial prend des notes personnelles dans un cahier ou une application locale type bureautique. Aucune trace n'est conservée au niveau de l'entreprise. Ce n'est que plus tard lorsque le contact deviendra un prospect ou un client que NSI commencera à prendre en charge la gestion des informations utiles au travers d'une solution de CRM (Customer Relationship Management – gestion de la relation client)

Ce qui vaut pour NSI vaut également pour toutes les entreprises et organisations qui souhaitent gérer le premier contact avec une personne. Il n'existe aucune application sur le marché prenant en charge ce besoin. Ce que l'on trouve sur le marché en matière de gestion des visiteurs se résume à la gestion de la sécurité ou de la billettique !

Fort de ce constat NSI développera une solution mobile en Java sous Android pour prendre en charge le premier contact avec une personne : par exemple un visiteur sur un salon, une exposition, un musée, un stade, etc…

# Infrastructure Réseau



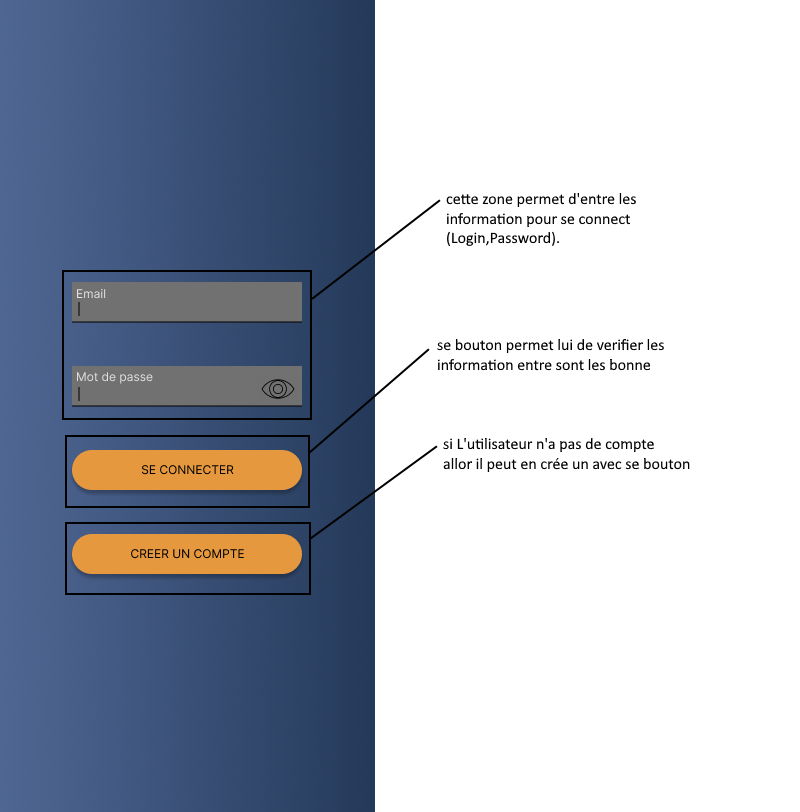
L’infrastructure pour l’application et plutôt simple avec un backend avec une API en PHP et Base de donnee postgreSQL et un frontend avec android pour l’application et une base de donnee SQLite en local.

# Maquette

Cette maquette représente l’interface utilisateur de l’application.

## CONNEXION utilisateur

## MENU principal

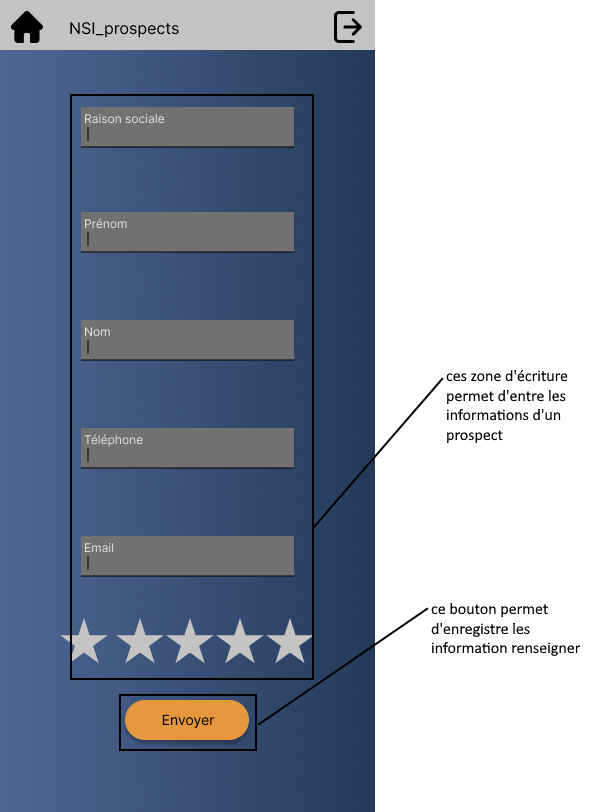
Le menu principal est constitué d’une liste des prospects présents dans la base de données et du header permettant de se déconnecter et de retourner au menu principal.

Graphical user interface, website

Description automatically generated

## Consultation, modification et saisie d'un prospect

### Ajout d’un prospect



### Modification d’un prospect

Lorsque l’on appuie sur un prospect dans la liste, on a la possibilité de consulter ses informations et de les modifier.



# Backend

Le back permet à l’application mobile de récupérer les données de la base de données distantes. Elle passe donc par une appli qui est développé en PHP. Qui va récupérer les données dans une base de données, poserait SQL.

## API PHP

Ce diagramme de classes présente la structure des classes de notre API. Elle se compose donc des trois classes dont Dbh qui permet de créer la connexion vers la base de données cette class est hérite par les classes User et Prospect qui sont deux classes qui permet des faire du CRUD dans la base de donnes.

class Dbh {  
 protected function connect() {  
 try {  
 return $dbh = new PDO("pgsql:host=46.18.230.10;port=9876;dbname=NSI\_prospect-Ollivier\_chris;"

,"postgres", "P@ssw0rdsio");  
 }catch (PDOException $exception){  
 print 'Error :' . $exception->getMessage() .' <br>';  
 die();  
 }  
 }  
}

La class Dbh n’a qu’une seul méthode qui permet de se connexion a la base de données, elle est héritée par toutes les classes qui ont le besoin de se connecte.

La class Prospect a donc 5 méthodes. Elles sont les suivent :

public function getOneProspect($raisonSocial){  
  
 $raisonSocial = strtolower($raisonSocial);  
  
 $stmt = $this->connect()->prepare('select \* from "Prospect" where "raisonSocial" = ?;');  
  
 if (!$stmt->execute(array($raisonSocial))) {  
 $stmt = null;  
 header("location: ./?error=404");  
 exit();  
 }  
 return $stmt->fetchAll(PDO::*FETCH\_ASSOC*);  
}

Cette méthode permet de récupère un prospect avec sa raison social. Elle lui retourne un array avec toutes les info.

public function getAllProspect(){  
 $stmt = $this->connect()->prepare('select \* from "Prospect";');  
  
 if (!$stmt->execute()) {  
 $stmt = null;  
 header("location: ./?error=404");  
 exit();  
 }  
 return $stmt->fetchAll(PDO::*FETCH\_ASSOC*);  
}

Cette méthode permet de récupère toutes les prospects. Elle lui retourne un array avec toutes les prospect et leurs info.

public function addOneProspect($id,$nom,$prenom,$siret,$score,$raisonSocial,$mail,$tel){  
 $stmt = $this->connect()->prepare('insert into "Prospect"(id,nom,prenom,siret,score,raisonSocial,mail,tel) values (?,?,?,?,?,?,?,?)');  
  
 if (!$stmt->execute(array($id,$nom,$prenom,$siret,$score,$raisonSocial,$mail,$tel))) {  
 $stmt = null;  
 header("location: ./?error=404");  
 exit();  
 }  
 $stmt = null;  
}

Cette méthode permet d’ajouter un prospect avec toutes les info nécessaire. Elle ne retourne rien.

public function updateOneProspect($id,$nom,$prenom,$siret,$score,$raisonSocial,$mail,$tel){  
 $stmt = $this->connect()->prepare('update "Prospect" set nom = ?,prenom = ?,siret = ?,score = ?,raisonSocial = ?,mail = ?,tel = ? where id = ?');  
  
 if (!$stmt->execute(array($nom,$prenom,$siret,$score,$raisonSocial,$mail,$tel,$id))) {  
 $stmt = null;  
 header("location: ./?error=404");  
 exit();  
 }  
 $stmt = null;  
}

Cette méthode permet de modifier un prospect avec toutes les info nécessaire.

public function checkIfExist($id){  
 $stmt = $this->connect()->prepare('select \* from "Prospect" WHERE id = ? ');  
  
 if (!$stmt->execute(array($id))){  
 $stmt = null;  
 header("location: ../frontend/php/index.php#popup-one?error=stmtFailed");  
 exit();  
 }  
  
 if ($stmt->rowCount() > 0){  
 $resCheck =false;  
 }else{  
 $resCheck =true;  
 }  
  
 return $resCheck;  
}

Cette méthode permet de savoir si un utilisateur existe si oui elle retourne un INT supérieur à un. La class User permet la même chose que la class Prospect mais avec les données.

## Base de donnée

Une image contenant texte, noir, capture d’écran, tableau de points

Description générée automatiquementLa base de données est très simple elle est la même pour le frontend car ont veut que les données soit les même de partout.

# FrontEnd

La partie d’infrastructure FrontEnd sont donc l’application Android et le serveur de base de données est SQLite. Ils sont désignés FrontEnd car ils sont en direct communication avec l’utilisateur. Ils servent donc IHM pour que l’utilisateur puisse utiliser l’application.

Le diagramme de class de toute la partie frontend est le suivant :

Graphical user interface, diagram

Description automatically generated

On y retrouve donc la class DataBaseHelper qui est intentiez dans toute les class de la view, mais aussi les classes User et Prospect qui sont intentiez dans les view qui interagie avec ceci et pour finir la petite particularité la class ProspectAdaptateur de la view qui utiliser une class et un interface interne.

## Classes API

La class a donc uniquement 3 attributs qui servent d’enum pour les adresses d’appels d’API.

private static final String *HTTP\_ROUTS\_PROSPECT* = "http://86.207.48.28:8080/Prospect-API/backEnd/API/";  
private static final String *HTTP\_ROUTS\_RAISONSOCIALE* = "https://entreprise.data.gouv.fr/api/sirene/v1/full\_text/";  
private static final String *HTTP\_ROUTS\_USERAPP* = "http://86.207.48.28:8080/Prospect-API/backEnd/API/";

### API Prospect

On a besoin d’accéder à nos prospect depuis notre application vers la base de données, pour ce faire, il existe une méthode getAllProspect() qui renvoi, sous le format JSON, la liste des prospects dans la base.

public static  
String getAllProspect() throws IOException, NetworkOnMainThreadException {  
 URL url = new URL(*HTTP\_ROUTS\_PROSPECT*+"GET/AllProspect.php");  
 try (BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream(), StandardCharsets.*UTF\_8*))) {  
 for (String line; (line = reader.readLine()) != null; ) {  
 return line;  
 }  
 }  
 return "Error";  
}

La méthode postProspect(), elle va ajouter un prospect à la base, lors de la création d’un prospect dans l’application.

public static  
int postProspect(String jsonBody) throws IOException {  
 URL obj = new URL(*HTTP\_ROUTS\_PROSPECT* + "POST/OneProspect.php");  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod("POST");  
 con.setRequestProperty("User-Agent", "Mozilla/5.0");  
 con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");  
 con.setDoOutput(true);  
 DataOutputStream wr = new DataOutputStream(con.getOutputStream());  
 wr.writeBytes(jsonBody);  
 wr.flush();  
 wr.close();  
 return con.getResponseCode();  
}

### API User

L’accès à l’API concernant les utilisateurs va être composé uniquement d’une méthode, qui permet d’accéder à tous les utilisateurs dans une chaîne de caractère dans le format JSON.

public static  
String getAllUserApp() throws IOException, NetworkOnMainThreadException {  
 URL url = new URL(*HTTP\_ROUTS\_USERAPP*+"GET/AllUser.php");  
 try (BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream(), "UTF-8"))) {  
 for (String line; (line = reader.readLine()) != null; ) {  
 return line;  
 }  
 }  
 return "Error";  
}

Cette méthode fait appel à l’api USER et la fonction AllUser.php dans les GET de celle-ci.

La méthode postUser(), elle va ajouter un user à la base, lors de la création d’un user dans l’application.

public static  
int postUser(String jsonBody) throws IOException {  
 URL obj = new URL(*HTTP\_ROUTS\_PROSPECT* + "POST/OneUser.php");  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestMethod("POST");  
 con.setRequestProperty("User-Agent", "Mozilla/5.0");  
 con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");  
 con.setDoOutput(true);  
 DataOutputStream wr = new DataOutputStream(con.getOutputStream());  
 wr.writeBytes(jsonBody);  
 wr.flush();  
 wr.close();  
 return con.getResponseCode();  
}

con.getResponseCode();  
}

### API données sociétés

Lors de l’entrée d’un prospect rencontré, on souhaite obtenir l’entreprise dans laquelle il est collaborateur, non seulement la raison sociale de celle-ci mais également son code SIRET. C’est pourquoi nous auront besoin de faire un appel à une API pour demander cette information.

L’API qui est utilisée est [SIRENE](https://entreprise.data.gouv.fr/api_doc/sirene), proposée par le gouvernement, elle permet d’accéder facilement au registre des société. L’API reçoit la raison sociale et renvoi toutes les informations (adresse, ) on va uniquement récupérer le code SIRET.

Dans la partie ‘API SIRET’ de la classe API, on commence par déclarer l’adresse de l’API (au cas où on décide de la changer par exemple, pour une modification simplifiée)

La méthode getDataFromRS() renvoi toutes les informations renvoyées par l’API lors de l’envoi de la raison sociale d’une entreprise. Le résultat est une chaine de caractère au format JSON.

public static String getDataFromRS(String RS) throws IOException, NetworkOnMainThreadException {  
 RS = RS.replaceAll(" ", "\_").toLowerCase();  
 URL url = new URL(*HTTP\_ROUTS\_RAISONSOCIALE*+RS);  
 try (BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream(), "UTF-8"))) {  
 for (String line; (line = reader.readLine()) != null; ) {  
 return line;  
 }  
 }  
 return "couldn't find data";  
}

La méthode getSiretFromRS() permet d’obtenir le SIRET de l’entreprise souhaitée en entrant la raison sociale, le résultat est une chaîne de caractère.

public static String getSiretFromRS(String RS) throws IOException, NetworkOnMainThreadException {  
 try {  
 JSONObject jsonObject = new JSONObject(*getDataFromRS*(RS));  
 return jsonObject.getJSONArray("etablissement").getJSONObject(0).getString("siret");  
 } catch (JSONException err) {  
 return "couldn't find siret";  
 }  
}

## Gestion de la base de données

La base de données regroupant les utilisateurs et les prospects est gérée depuis la classe DataBaseHelper.java qui regroupe des méthodes de création, et d’accès a des données.

La méthode onCreate, appelée lors du lancement de l’application, va, si elle n’existe pas déjà, créer la base de données avec les tables user et prospect.

@Override  
public void onCreate(SQLiteDatabase sqLiteDatabase) {  
 String CreateTableStatementUser = "create table user( Id INTEGER PRIMARY KEY AUTOINCREMENT,email text, password text, nom TEXT, prenom TEXT);";  
 String CreateTableStatementProspect = "create table prospect(Id INTEGER primary key autoincrement,nom text,prenom text,siret text,raisonSociale text, score integer, mail text,tel text);";  
  
  
 sqLiteDatabase.execSQL(CreateTableStatementUser);  
 sqLiteDatabase.execSQL(CreateTableStatementProspect);  
}

La méthode onUpgrade() permet de mettre a jour la base de données en supprimant les tables user et prospect et en les recréant.

@Override  
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {  
 String CreateTableStatementDeleteUser = "Drop table if exists user;";  
 String CreateTableStatementDeleteProspect = "Drop table if exists prospect;";  
  
 sqLiteDatabase.execSQL(CreateTableStatementDeleteUser);  
 sqLiteDatabase.execSQL(CreateTableStatementDeleteProspect);  
  
 onCreate(sqLiteDatabase);  
  
}

La méthode addNewUser() créé un nouvel utilisateur avec comme paramètre un objet User, dans la base de données.

public void addNewUser(User user) {  
 try {  
 user.setEmail(*encryption*.encrypt(user.getEmail()));  
 user.setPassword(*encryption*.encrypt(user.getPassword()));  
 user.setNom(*encryption*.encrypt(user.getNom()));  
 user.setPrenom(*encryption*.encrypt(user.getPrenom()));  
  
 SQLiteDatabase sqLiteDatabase = this.getWritableDatabase();  
 ContentValues values = new ContentValues();  
  
 values.put("email",user.getEmail());  
 values.put("password",user.getPassword());  
 values.put("nom", user.getNom());  
 values.put("prenom", user.getPrenom());  
 sqLiteDatabase.insert("user", null, values);  
 sqLiteDatabase.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
}

La méthode readAllUser() renvoi la liste des utilisateurs de la base de données sous une arrayList.

public ArrayList<User> readAllUser() {  
 SQLiteDatabase db = this.getReadableDatabase();  
  
 assert *encryption* != null;  
 Cursor cursorUser = db.rawQuery("SELECT \* FROM user", new String[]{});  
  
 ArrayList<User> userArrayList = new ArrayList<>();  
 if (cursorUser.moveToFirst()) {  
 do {  
 userArrayList.add(new User(  
 cursorUser.getInt(0),  
 *encryption*.decryptOrNull(cursorUser.getString(1)),   
 *encryption*.decryptOrNull(cursorUser.getString(2)),   
 *encryption*.decryptOrNull(cursorUser.getString(3)),   
 *encryption*.decryptOrNull(cursorUser.getString(4))   
 ));  
 } while (cursorUser.moveToNext());  
 }  
 cursorUser.close();  
 return userArrayList;  
}

La méthode readUserFromUser() renvoi, à partir d’un objet User, un seul utilisateurs.

public ArrayList<User> readUserFormUser(User user) {  
 SQLiteDatabase db = this.getReadableDatabase();  
  
 assert *encryption* != null;  
 Cursor cursorUser = db.rawQuery("SELECT \* FROM user WHERE email = ?", new String[]{*encryption*.encryptOrNull(user.getEmail())});  
  
 ArrayList<User> userArrayList = new ArrayList<>();  
 if (cursorUser.moveToFirst()) {  
 do {  
 userArrayList.add(new User(  
 cursorUser.getInt(0),  
 *encryption*.decryptOrNull(cursorUser.getString(1)),   
 *encryption*.decryptOrNull(cursorUser.getString(2)),   
 *encryption*.decryptOrNull(cursorUser.getString(3)),   
 *encryption*.decryptOrNull(cursorUser.getString(4))   
 ));  
 } while (cursorUser.moveToNext());  
 }  
 cursorUser.close();  
 return userArrayList;  
}

La méthode readNumberUserFromMail(), elle, renvoi 1 s’il existe un utilisateur ayant l’adresse mail mis en paramètre

public int readNumberUserFromMail(String mail) {  
 SQLiteDatabase db = this.getReadableDatabase();  
 *userResult* = 0;  
 assert *encryption* != null;  
 Cursor cursorUser = db.rawQuery("SELECT \* FROM user WHERE email = ?", new String[]{*encryption*.encryptOrNull(mail)});  
 if (cursorUser.moveToFirst()) {  
 do {  
 *userResult*++;  
 } while (cursorUser.moveToNext());  
 }cursorUser.close(); return *userResult*;}

La méthode addNewProspect créé un prospect dans la base de données à partir d’un objet Prospect

public void addNewProspect(Prospect prospect) {  
 try {  
 SQLiteDatabase sqLiteDatabase = this.getWritableDatabase();  
 ContentValues values = new ContentValues();  
 values.put("nom",prospect.getNom());  
 values.put("prenom",prospect.getPrenom());  
 values.put("siret",prospect.getSiret());  
 values.put("raisonSociale",prospect.getRaisonSociale());  
 values.put("score",prospect.getScore());  
 values.put("mail",prospect.getMail());  
 values.put("tel",prospect.getTel());  
 sqLiteDatabase.insert("prospect", null, values);  
 sqLiteDatabase.close();  
 } catch (Exception e) {  
 e.printStackTrace();}}

La méthode readAllProspect() renvoi une arrayList contenant les prospects.

public ArrayList<Prospect> readAllProspect() {  
 SQLiteDatabase db = this.getReadableDatabase();  
 Cursor cursorProspect = db.rawQuery("SELECT \* FROM prospect",new String[]{});  
 ArrayList<Prospect> prospectArrayList = new ArrayList<>();  
 if (cursorProspect.moveToFirst()) {  
 do {  
 prospectArrayList.add(new Prospect(  
 cursorProspect.getInt(0), //ID  
 cursorProspect.getString(1), //nom  
 cursorProspect.getString(2), //prenom  
 cursorProspect.getString(3), //siret  
 cursorProspect.getString(4), //RS  
 cursorProspect.getInt(5), //score  
 cursorProspect.getString(6), //tel  
 cursorProspect.getString(7) //mail  
 ));  
 } while (cursorProspect.moveToNext());  
 }  
 cursorProspect.close();  
 return prospectArrayList;  
}

La méthode readNumberPropectFromNomPrenomSiret() permet de compter le nombre de prospect dans la table Prospect, pour vérifier si les informations de prospect existent dans la table.

public int readNumberProspectFromNomPrenomSiret(String nom ,String prenom ,String siret) {  
 SQLiteDatabase db = this.getReadableDatabase();  
 *userResult* = 0;  
 Cursor cursorUser = db.rawQuery("SELECT \* FROM prospect WHERE nom = ? And prenom = ? And siret = ?", new String[]{nom,prenom,siret});  
  
 if (cursorUser.moveToFirst()) {  
 do {  
 *userResult*++;  
 } while (cursorUser.moveToNext());  
 }  
 cursorUser.close();  
 return *userResult*;  
}

## Class Prospect

Le classe prospect se composent de plusieurs attribue qui sont les suivants :

private int Id;  
private String nom;  
private String prenom;  
private String siret;  
private String raisonSociale;  
private int score;  
private String mail;  
private String tel;

L'objet peut se construire grâce à 2 différents constructeurs qui sont overlead Le constructeur par défaut tous les attributs et le 2e n'a pas l’id :

public Prospect(int id, String nom, String prenom, String siret, String raisonSociale, int score, String mail, String tel) {  
 this.Id = id;  
 this.nom = nom;  
 this.prenom = prenom;  
 this.siret = siret;  
 this.raisonSociale = raisonSociale;  
 this.score = score;  
 this.mail = mail;  
 this.tel = tel;  
}

Tous ces attributs ont bien évidemment leur guide récepteur qui permettent de récupérer ou de positionner des informations lorsqu'un objet est instancié.

Enfin l'objet prospect a 2 types de méthodes différents qui permettent les 2 de Récupérer des informations dans un Json relatif à un prospect

La première type de méthode permet donc te retrouver le nombre de prospects dans le Json en question , cette méthode s'apparente donc énormément à un getSize d'un array :

public static  
int getJsonArraySize(String data)throws IOException {  
 try {  
 JSONObject jsonObject = new JSONObject(data);  
 return jsonObject.getJSONArray("Prospect").length();  
 }catch (JSONException err){  
 return 0;  
 }  
}

Le 2e type de méthode permet-elle de récupérer l'information, elle s'apparente donc à un getter mais pour les informations en Json relatif au prospect :

public static @NotNullString getProscpectNom(String data, int i) throws IOException {  
 try {  
 JSONObject jsonObject = new JSONObject(data);  
 return jsonObject.getJSONArray("Prospect").getJSONObject(i).getString("nom");  
 }catch (JSONException err){  
 return "";  
 }}

## Class User

Cette classe est composée de 5 attributs qui sont les suivants :

private int Id;  
private String email;  
private String password;  
private String nom;  
private String prenom;

En ce qui concerne les méthodes elles sont relativement identiques à celle de la class Prospect mais relatif à un utilisateur.

# View

La view est un partie plutôt diffèrent du reste du code dans Android, car elles sont directement liées a un fichier XML qui permet de réaliser leurs visuelle. Elles sont donc là pour permettre l’interaction entre le code et le visuelle.

## Login

La premier page qui nous est présenté en lancent l’application est la page de login qui nous permet donc de nous connecte à l’application

Comme toutes autre view elle a une méthode OnCreate qui est obligatoire avec l’héritage. Elle permet d’effectuer l’initialisation de toutes ces « attribue» comme un constructeur que l’on appellera qu’une fois et des récupère les id des fixture de la page. Dans celle si on exécute aussi la méthode GetAllDataFromAPI .

**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.*loginactivity*);  
 dataBase = **new** DataBaseHelper(**this**);  
 InputLogin = (EditText) findViewById(R.id.*inputLogin*);  
 InputPassword = (EditText) findViewById(R.id.*inputPassword*);  
 Button btnLogin = (Button) findViewById(R.id.*BtnLogin*);  
 btnLogin.setOnClickListener(eventBtnLogin);  
 Button btnSignIn = findViewById(R.id.*BtnSignIn*);  
 btnSignIn.setOnClickListener(eventBtnSignIn);  
 StrictMode.ThreadPolicy policy = **new** StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.*setThreadPolicy*(policy);  
 Log.*v*("API Status","" + getAllDataFromAPI());  
}

Les méthodes OnClick permet de réagir à l’évènement d’un click de bouton. Celui si permet de se connecte si le login et mot de passe sont correcte.

**public** View.OnClickListener eventBtnLogin = v -> {  
 User user = **new** User(InputLogin.getText().toString(), InputPassword.getText().toString());  
 ArrayList<User> allUser = dataBase.readUserFormUser(user);  
 **try** {  
 **if** (user.getPassword() != **null** && allUser.get(0).getPassword() != **null** && user.getPassword().equals(allUser.get(0).getPassword())) {  
 Log.*v*("connexion", "Connexion effectuée : " + allUser.get(0).getPassword());  
 Intent connexion = **new** Intent(LoginActivity.**this**, AccueilActivity.**class**);  
 startActivity(connexion);  
 } **else** {  
 InputPassword.setError("Password and username didn't match");  
 Log.*v*("Error", "connection failed");  
 }  
 } **catch** (Exception e) {  
 Log.*v*("Error", "connection failed" + e);  
 InputPassword.setError("DO NOT EXISTE");  
 }  
};

cette methode permet de passe verre la page SignIn.

**public** View.OnClickListener eventBtnSignIn = v -> {  
 Intent connexion = **new** Intent(LoginActivity.**this**, SignInActivity.**class**);  
 startActivity(connexion);  
};

la methode GetAllDataFromAPI permet de recupere toutes les data user et prospect de l’API.

**private boolean** getAllDataFromAPI(){  
 **try** {  
 String JsonUser = API.*getAllUserApp*();  
 **for** (**int** i = 0; i < User.*getJsonArraySize*(JsonUser); i++) {  
 **if** (dataBase.readNumberUserFromMail(User.*getAppUserMail*(JsonUser, i)) == 0) {  
 dataBase.addNewUser(**new** User(  
 User.*getAppUserMail*(JsonUser, i),  
 User.*getAppUserPassword*(JsonUser, i),  
 User.*getAppUserNom*(JsonUser, i),  
 User.*getAppUserPrenom*(JsonUser, i)  
 ));  
 }  
 }  
 String JsonProspect = API.*getAllProspect*();  
 **for** (**int** i = 0; i < Prospect.*getJsonArraySize*(JsonProspect); i++) {  
 **if** (dataBase.readNumberProspectFromNomPrenomSiret(Prospect.*getProscpectNom*(JsonProspect, i), Prospect.*getProscpectPrenom*(JsonProspect, i), Prospect.*getProscpectSiret*(JsonProspect, i)) == 0) {  
 dataBase.addNewProspect(**new** Prospect(  
 Prospect.*getProscpectNom*(JsonProspect, i),  
 Prospect.*getProscpectPrenom*(JsonProspect, i),  
 Prospect.*getProscpectSiret*(JsonProspect, i),  
 Prospect.*getProscpectRaisonSocial*(JsonProspect, i),  
 Prospect.*getProscpectScore*(JsonProspect, i),  
 Prospect.*getProscpectMail*(JsonProspect, i),  
 Prospect.*getProscpectTel*(JsonProspect, i)  
 ));  
 }  
 }  
 **return true**;  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 **return false**;  
 }  
}

## SignIn

Cette view permet aux utilisateur de se créer un compte.

La méthode created permet de crée un compte est de l’ajoute dans la base de donnée et a l’API.

**public** View.OnClickListener eventBtnCreated = v -> {  
 **if** (!SignInActivity.*isEmpty*(InputFName)&&!SignInActivity.*isEmpty*(InputLName)&&!SignInActivity.*isEmpty*(InputLogin)&&!SignInActivity.*isEmpty*(InputPassword)) {  
 dataBase.addNewUser(**new** User(InputLogin.getText().toString(), InputPassword.getText().toString(), InputFName.getText().toString(), InputLName.getText().toString()));  
 **try** {  
 JSONObject jsonBody = **new** JSONObject();  
 jsonBody.put("UserApp", **new** JSONArray());  
 JSONObject UserJson = **new** JSONObject();  
 **for** (User user : dataBase.readAllUser()) {  
 UserJson.put("id", user.getId());  
 UserJson.put("email", user.getEmail());  
 UserJson.put("password", user.getPassword());  
 UserJson.put("nom", user.getNom());  
 UserJson.put("prenom", user.getPrenom());  
 jsonBody.accumulate("UserApp", UserJson);  
 }  
 Log.*d*("json for POST", jsonBody.toString());  
 API.*postUser*(jsonBody.toString());  
 } **catch** (IOException | JSONException e) {  
 e.printStackTrace();  
 }  
 }**else** {  
 Toast.*makeText*(SignInActivity.**this**,"Des informations sont incompletes ou manquantes",Toast.*LENGTH\_LONG*).show();  
 }  
};

La methode BackLogin permet de retourner verre la page de login

**public** View.OnClickListener eventBtnBackLogin = v -> {  
 Intent connexion = **new** Intent(SignInActivity.**this**, LoginActivity.**class**);  
 startActivity(connexion);  
};

## accueil

L’accueil est l’endroit où est diriger l’utilisateur une fois qu’il s’est connecter il peut aussi y accéder à l’aide du bouton accueil s’il est sur une autre page.

Ces deux méthodes sont réutilisées dans d’autre view elle permet respectivement de se déconnecte ou de retourner verre la l’accueil.

**public** View.OnClickListener backbutton = v -> {  
 Intent intent = **new** Intent(AccueilActivity.**this**, AccueilActivity.**class**);  
 startActivity(intent);  
};  
  
**public** View.OnClickListener logout = v -> {  
 Intent intent = **new** Intent(AccueilActivity.**this**, LoginActivity.**class**);  
 startActivity(intent);  
};

Cette méthode permet d’aller vers la page d’ajoute d’un prospect

**public** View.OnClickListener addprospect = v -> {  
 Intent intent = **new** Intent(AccueilActivity.**this**, AddProspectActivity.**class**);  
 startActivity(intent);  
};

La méthode setRecyclerView() initialisent la liste de prospect dans une box scrollable.

private void setRecyclerView() {  
 recycler\_View.setHasFixedSize(true);  
 recycler\_View.setLayoutManager(new LinearLayoutManager(this));  
 adaptateur = new ProspectAdaptateur(this, dataBase.readAllProspect());  
 recycler\_View.setAdapter(adaptateur);  
}

Enfin la méthode OnProspectClick permet pour chaque prospect dans la box de devenir un bouton cliquable, Elle permet après se click aller sur la page pour visualiser / modifier un prospect.

@Override  
**public void** onProspectClick(**int** position) {  
 Intent intent = **new** Intent(**this**, DisplayProspectActivity.**class**);  
 intent.putExtra("ProspectId", prospectsList.get(position).getId());  
 intent.putExtra("ProspectNom", prospectsList.get(position).getNom());  
 intent.putExtra("ProspectPrenom", prospectsList.get(position).getPrenom());  
 intent.putExtra("ProspectRS", prospectsList.get(position).getRaisonSociale());  
 intent.putExtra("ProspectSiret", prospectsList.get(position).getSiret());  
 intent.putExtra("ProspectScore", prospectsList.get(position).getScore());  
 intent.putExtra("ProspectMail", prospectsList.get(position).getMail());  
 intent.putExtra("ProspectTel", prospectsList.get(position).getTel());  
 startActivity(intent);  
 **this**.finish();  
}

## AddProspect

La class AddProspectActivity est une view qui permet aux utilisateur de créer un nouvelle prospect avec toutes les information qui le décrit.

La methode Submit permet d’ajoute le prospect a la base de donnée front et back end.

**public** View.OnClickListener eventBtnSubmit = v -> {  
 **try** {  
 siret = API.*getSiretFromRS*(InputRaisonSocial.getText().toString());  
 } **catch** (Exception ignored) {}  
 dataBase.addNewProspect(**new** Prospect(  
 InputNom.getText().toString(),  
 InputPrenom.getText().toString(),  
 siret,  
 InputRaisonSocial.getText().toString(),  
 (**int**) InputStarRating.getRating(),  
 InputMail.getText().toString(),  
 InputTel.getText().toString()));  
 **try** {  
 JSONObject jsonProspect = **new** JSONObject();  
 jsonProspect.put("Prospect", **new** JSONArray());  
 ArrayList<Prospect> listOfAllProspect = dataBase.readAllProspect();  
 **for** (**int** i = 0; i < dataBase.readAllProspect().size(); i++) {  
 JSONObject ProspectJson = **new** JSONObject();  
 ProspectJson.put("id", listOfAllProspect.get(i).getId());  
 ProspectJson.put("nom", listOfAllProspect.get(i).getNom());  
 ProspectJson.put("prenom", listOfAllProspect.get(i).getPrenom());  
 ProspectJson.put("siret", listOfAllProspect.get(i).getSiret());  
 ProspectJson.put("score", listOfAllProspect.get(i).getScore());  
 ProspectJson.put("raisonsocial", listOfAllProspect.get(i).getRaisonSociale());  
 ProspectJson.put("mail", listOfAllProspect.get(i).getMail());  
 ProspectJson.put("tel", listOfAllProspect.get(i).getTel());  
 jsonProspect.accumulate("Prospect", ProspectJson);  
 }  
 Log.*v*("Post status", "" + API.*postProspect*(jsonProspect.toString()));  
 } **catch** (Exception e) {  
 Log.*v*("error", e.getMessage());  
 }  
 Intent connexion = **new** Intent(AddProspectActivity.**this**, AccueilActivity.**class**);  
 startActivity(connexion);  
};

## DisplayProspect

La class DisplayProspectActivity est la dernier view est-elle permet aux utilisateurs de voire les information des se même prospect on peut aussi modifier les information du prospect si elle ne correspond plu.

La premiere methode permet a l’utilisateur d’activer la modification sur les input.

**public** View.OnClickListener btnModifier = v -> {  
 Button btn\_Enregistre = findViewById(R.id.*Button\_Enregistrer*);  
 btn\_Enregistre.setEnabled(**true**);  
 TextNom.setEnabled(**true**);  
 TextPrenom.setEnabled(**true**);  
 TextRS.setEnabled(**true**);  
 TextTel.setEnabled(**true**);  
 TextMail.setEnabled(**true**);  
 StareRatting.setEnabled(**true**);  
};

le methode Enregistre permet d’enregistrer le modification soit d’update les informations du prospect dans la base de données Frontend et Backend.

**public** View.OnClickListener btnEnregistre = v -> {  
 **try** {  
 String siret = API.*getSiretFromRS*(TextRS.getText().toString());  
 prospectapp.setNom(TextNom.getText().toString());  
 prospectapp.setPrenom(TextPrenom.getText().toString());  
 prospectapp.setRaisonSociale(TextRS.getText().toString());  
 prospectapp.setSiret(siret);  
 prospectapp.setScore((**int**) StareRatting.getRating());  
 prospectapp.setMail(TextMail.getText().toString());  
 prospectapp.setTel(TextTel.getText().toString());  
 dataBase.updateProspectFromProspect(prospectapp);  
 } **catch** (Exception ignored) {}  
 **try** {  
 JSONObject jsonProspect = **new** JSONObject();  
 jsonProspect.put("Prospect", **new** JSONArray());  
 ArrayList<Prospect> listOfAllProspect = dataBase.readAllProspect();  
 **for** (**int** i = 0; i < dataBase.readAllProspect().size() ; i++) {  
 JSONObject ProspectJson = **new** JSONObject();  
 ProspectJson.put("id", listOfAllProspect.get(i).getId());  
 ProspectJson.put("nom", listOfAllProspect.get(i).getNom());  
 ProspectJson.put("prenom", listOfAllProspect.get(i).getPrenom());  
 ProspectJson.put("siret", listOfAllProspect.get(i).getSiret());  
 ProspectJson.put("score", listOfAllProspect.get(i).getScore());  
 ProspectJson.put("raisonsocial", listOfAllProspect.get(i).getRaisonSociale());  
 ProspectJson.put("mail", listOfAllProspect.get(i).getMail());  
 ProspectJson.put("tel", listOfAllProspect.get(i).getTel());  
 jsonProspect.accumulate("Prospect", ProspectJson);  
 }  
 Log.*v*("Post status", "" + API.*postProspect*(jsonProspect.toString()));  
 } **catch** (Exception e) {Log.*v*("error", e.getMessage());}  
 };