Compte rendu de la mission 2 : Développement de l'application Android de l'application GSB de suivi des frais

Florian Martin – BTS SIO option SLAM

Année 2019-2020

Sommaire

I) Contexte

Galaxy Swiss Bourdin (GSB) est un laboratoire pharmaceutique. Le suivi des frais des visiteurs-salariés du laboratoire est actuellement géré de plusieurs façons. On souhaite uniformiser cette gestion.

Ainsi, une application web a été développée, destinée aux visiteurs et aux comptables de l'entreprise, permettant d'enregistrer tous les frais engagés (événementiel, déplacement, restauration…). L'application permet un suivi daté des opérations menées par le service comptable (validation des demandes de remboursement, mise en paiement, remboursement effectué). Pour ce qui est de l'interface destinée aux visiteurs, elle permet à ces derniers de pouvoir saisir tous les frais qu'ils engendrent.

Suite à la demande des visiteurs, une application Android est également en cours de développement. Elle doit permettre aux visiteurs de saisir en direct leurs frais (forfaitisés ou hors forfaits). L'application est une sorte de mémo qui permet d'enregistrer l'information à tout moment pour le mois en cours. Les visiteurs peuvent ainsi consulter leur mobile pour voir ce qu'ils ont enregistré.

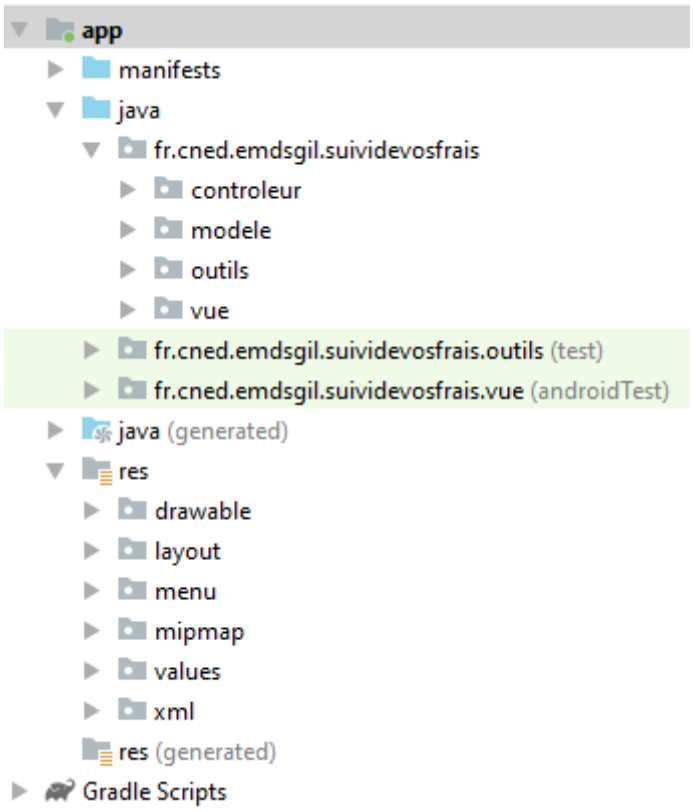
L'application Android doit notamment permettre l'envoi direct des informations saisies sur le serveur web qui mettra ainsi directement à jour la base de données, communes à l'application Android et l'application web, sans que le visiteur ait à ressaisir les données dans l'application web.

L'application comportait déjà un menu principal permettant d'accéder à la saisie des différentes catégories de frais ainsi que la vue portant sur la saisie des kilomètres et des frais hors forfait ainsi que le récapitulatif des frais hors forfait.

J'ai ainsi dû réaliser plusieurs tâches : coder la page de connexion de l'application, interdire la saisie direct dans la saisie des kms et autoriser la saisie uniquement avec les boutons + et – dans l'application, gérer la saisie des frais de repas, de nuitées et d'étapes, gérer la suppression de frais hors forfaits dans la vue affichant le récapitulatif des frais hors forfaits saisis, créer la connexion à la base de données, gérer l'enregistrement des différentes valeurs saisies par le visiteur et produire une documentation technique.

Cette application a été développée en Java. Le développement a été effectué grâce à l'IDE Android Studio. Concernant la gestion de version du projet, je me suis servi du logiciel de gestion de version Git et du service en ligne GitHub.

II) Architecture de l'application



Dans le dossier « manifests » on retrouve notamment un fichier de configuration pour l'application.

Dans le dossier java, on va retrouver plusieurs dossiers et sous-dossiers.

On retrouve tout d'abord le dossier « controleur » qui contient le contrôleur de l'application (l'application étant construite sur le patron de conception MVC).

Ensuite, on retrouve le dossier « modele » qui contient les modele de l'application.

Également, le dossier « vue » qui contient les différentes vues de l'application.

Vient également le dossier « outils » qui contient des classes comprenant des méthodes génériques partagées à l'ensemble de l'application et qui, de manière générale, peuvent être réutilisées dans d'autres projets.

Ensuite, on a les dossiers qui contiennent les différents tests de l'application (les dossiers contenants les tests contiennent entre parenthèses « test » et « androidTest »).

Ensuite, dans le dossier « res », on retrouve des sous-dossiers qui sont importants dans une application Android.

Dans le sous-dossier « drawable », on retrouve les images de l'application.

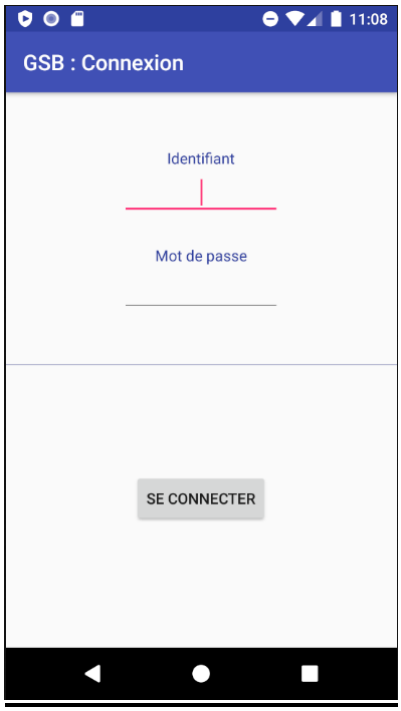
Dans le sous-dossier « layout », on retrouve les activity de l'application. Dans une application Android, les activity correspondent aux vues de l'application et permettent de gérer ces vues, notamment les boutons graphiques, les labels, les inputs etc. Ces activity sont des fichiers XML.

Le sous-dossier « menu » permet de gérer l'onglet menu en haut de chaque vue de l'application.

Ensuite, et pour terminer, le sous-dossier « values » contient des fichiers contenants des constantes globales réutilisables dans l'ensemble de l'application.

III) Connexion des utilisateurs

Ci-dessous, voici l'interface que j'ai développé pour gérer la connexion des utilisateurs :



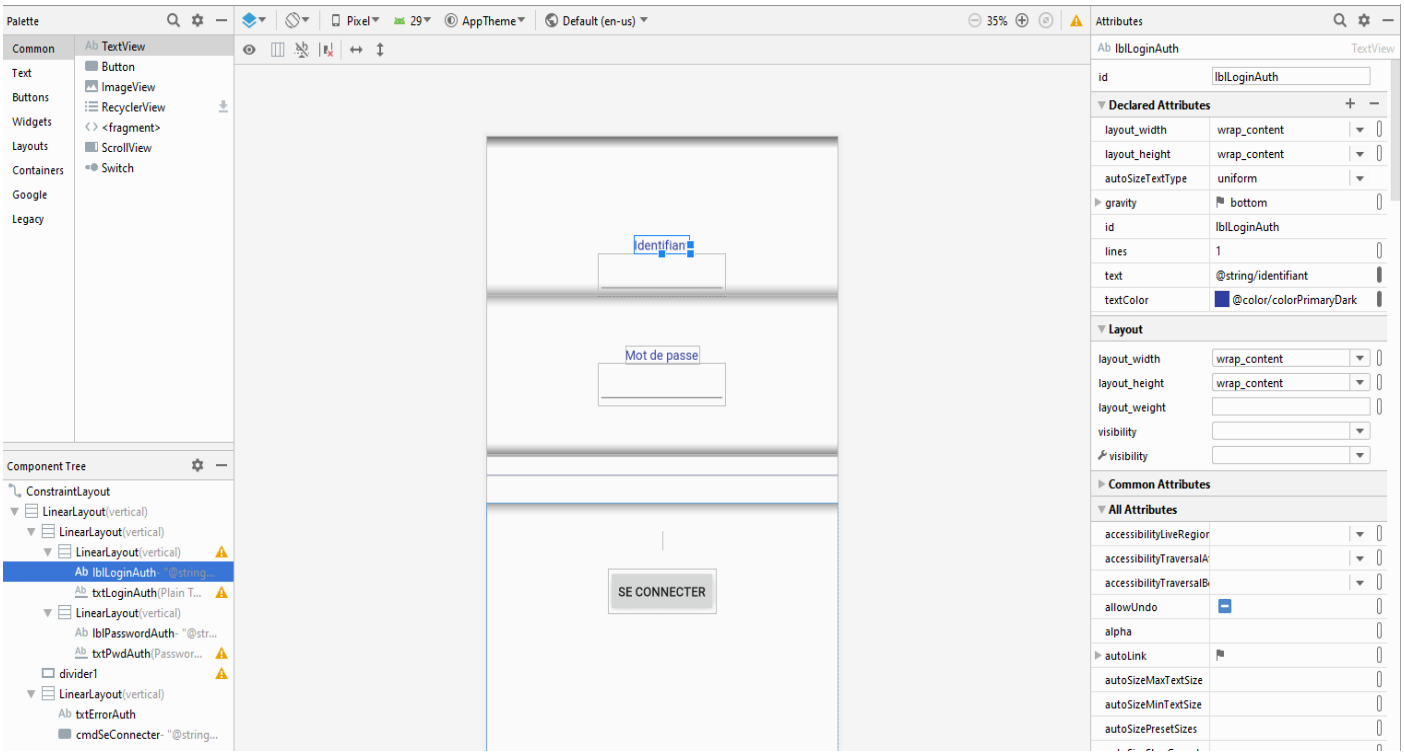
Pour créer une vue dans une application Android et pouvoir gérer tous les composants graphiques, Android Studio propose tous les outils et éléments nécessaires. Tout d'abord, il faut savoir qu'une vue est, sous Android Studio, appelée une activity. Une activity correspond à un fichier XML, donc sous forme de balises.

Voici, par exemple, un extrait du fichier XML ayant permis de créer l'activity de la page de connexion. On peut par exemple voir des balises <LinearLayout> qui en englobent d'autres. Les balises LinearLayout sont des sortes de conteneurs qui permettent de structurer le contenu d'une activité. Par exemple, on peut voir dans la capture d'écran ci-dessous qu'une balise LinearLayout englobe des labels et des inputs. Les balises <TextView> et <EditText> sont des composants graphiques. Dernier point important sur la capture d'écran suivante, c'est que chaque balise contient différentes propriétés permettant de pouvoir modifier leur comportement ou leur design. Par exemple on peut voir que la balise <TextView> sera de la couleur « colorPrimaryDark ». La couleur « colorPrimaryDark » est définie dans le fichier « colors.xml » dans le dossier « values » comme le montre la seconde capture d'écran ci-dessous et est partagée par l'ensemble de l'application :

```xml
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="0.4"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal|center_vertical"
            android:layout_weight="0.3"
            android:gravity="bottom|center_horizontal"
            android:orientation="vertical">

            <TextView
                android:id="@+id/lblLoginAuth"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:autoSizeTextType="uniform"
                android:gravity="bottom"
                android:lines="1"
                android:text="Identifiant"
                android:textColor="@color/colorPrimaryDark" />

            <EditText
                android:id="@+id/txtLoginAuth"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:ems="7"
                android:gravity="center"
                android:inputType="textPersonName" />
        </LinearLayout>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal|center_vertical"
            android:layout_weight="0.3"
            android:gravity="center_horizontal|center_vertical"
            android:orientation="vertical">
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

De la même manière que l'on peut configurer une activity dans un fichier XML en écrivant les balises XML directement, on peut aussi passer par une interface graphique, ce qui facilite la création, la disposition et la visualisation lors de la conception d'une activity. On peut voir sur la capture d'écran ci-dessous que l'on peut choisir les composants graphiques en haut à gauche de la capture d'écran et les faire glisser directement sur la vue centrale de la capture. Également, en bas à gauche, on retrouve l'architecture des différents composants de la vue avec les conteneurs, les labels, les inputs etc. Enfin, sur la partie de droite, on a accès aux différentes propriétés des différents composants graphiques qu'il est possible de modifier à souhait. Voici à quoi ressemble la création d'une activity par l'interface graphique d'Android Studio :



Une fois que l'interface de la vue est terminée, il reste à implémenter le code correspondant aux fonctionnalités et aux interactions des utilisateurs. Cela se fait dans des fichiers Java.

Tout d'abord, dans le dossier « vue », on fait correspondre un fichier java à chaque activity. Ici, on fait correspondre le fichier « MainActivity » à l'activity de la connexion. Par convention, le fichier et la vue contenant « main » dans le libellé est enfaite la vue sur laquelle démarre l'application. Voici, ci-dessous un extrait du fichier MainActivity. On peut tout d'abord voir qu'on déclare les différentes propriétés de la classe. Ensuite dans la fonction « onCreate » on s'occupe notamment à la ligne 32 de lier ce fichier Java à l'activity « activity_main » qui correspond à la vue de connexion. Ensuite, on valorise des propriétés et on appelle la procédure qui permet l'écoute sur le bouton « connexion » :

```
public class MainActivity extends AppCompatActivity {

    // informations affichées dans l'activité
    public static String login; // identifiant entré par le visiteur
    public static String passwordEntre; // mot de passe entré par le visiteur
    private Controle controle; // Contient l'unique instance du contrôleur
    public static TextView txtErrorAuth;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        setTitle("GSB : Connexion");
        // Créé le contrôleur
        this.controle = Controle.getInstance(null);
        Controle.context = this;
        this.txtErrorAuth = (TextView)findViewById(R.id.txtErrorAuth);
        // chargement des méthodes événementielles
        cmdSeConnecter_clic();
    }
```

Une fois que l'instanciation de la vue est faite, il faut gérer la connexion de l'utilisateur. Pour cela, tout se déroule dans la procédure « cmdSeConnecter_clic » qui est présentée sur la capture d'écran suivante. Cette procédure s'occupe d'écouter le bouton connexion. Au clic sur le bouton « connexion », on récupère le login et le mot de passe saisis par le visiteur. Une fois récupérée, on envoie au contrôleur les informations par l'intermédiaire de la fonction accesDonnees qui prend en premier paramètre l'opération à effectuer et en second paramètre les données à traiter, converties au préalable en tableau JSON grâce à la méthode « convertToJSONArray ». On peut remarquer que la méthode convertToJSONArray s'occupe d'encrypter le mot de passe entré. En effet, puisque dans la base de données les mots de passe sont cryptés, pour vérifier si le couple login/mot de passe entrés par le visiteur est correct, il faut au préalable que je hash le mot de passe ; Pour cela, j'ai créé deux méthodes dans la classe MesOutils permettant d'hasher une chaîne de caractère (présentent sur la seconde capture d'écran). La méthode hashString permet de hasher le mot de passe passé en paramètre grâce à l'algorithme de hashage « SHA-256 ». La méthode byteToHex permet quant à elle de transformer le tableau de byte, passé en paramètre, en chaîne de caractère (étant donné que la méthode hashString effectue des traitements sur des bytes) :

```
42      /**
43       * Sur le clic du bouton se connecter: on vérifie si le couple identifiant/mot de passe est
44       * correct. Si c'est correct, le visiteur a accès à son espace personnel, sinon, retour à
45       * l'activité de connexion.
46       */
47      private void cmdSeConnecter_clic() {
48          findViewById(R.id.cmdSeConnecter).setOnClickListener((v) → {
51                  // Récupération de l'identifiant et du mot de passe entrés
52                  login = ((EditText)findViewById(R.id.txtLoginAuth)).getText().toString();
53                  passwordEntre = ((EditText)findViewById(R.id.txtPwdAuth)).getText().toString();
54                  // Récupération dans la BDD de l'id du visiteur
55                  controle.accesDonnees( operation: "recupInfos", convertToJSONArray());
56          });
58      }
59
60      /**
61       * Conversion du login et du mdp entrés au format JSONArray
62       * @return login et mdp au format JSONArray
63       */
64      public JSONArray convertToJSONArray(){
65          List list = new ArrayList();
66          list.add(login);
67          list.add(MesOutils.hashString(passwordEntre));
68
69          return new JSONArray(list);
70      }
```

```java
    /**
     * Permet de hasher la chaîne de caractère fournie en paramètre grâce à l'algorithme de hashage
     * SHA-256
     *
     * @param chaine chaîne de caractère
     * @return la chaîne de caractère hashée
     */
    public static String hashString(String chaine) {
        byte[] hash = null;
        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        try {
            hash = md.digest((chaine).getBytes( charsetName: "UTF-8"));
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return byteToHex(hash);
    }

    /**
     * Permet d'effectuer une conversion d'un tableau de bits au format hexadécimale pour finalement
     * retourner le tableau de bits sous forme de chaîne de caractère String
     *
     * @param bits conversion au format hexadécimale puis au format String
     * @return les bits passés en paramètre au format String
     */
@   public static String byteToHex(byte[] bits) {
        if (bits == null) {
            return null;
        }
        StringBuffer hex = new StringBuffer( capacity: bits.length * 2);
        for (int i = 0; i < bits.length; i++) {
            if (((int) bits[i] & 0xff) < 0x10) {
                hex.append("0");
            }
            hex.append(Integer.toString( i: (int) bits[i] & 0xff,  radix: 16));
        }
        return hex.toString();
    }
```

Dans le contrôleur, on s'occupe ensuite d'appeler la méthode « envoi » sur l'instance de la classe AccesDistant avec les paramètres reçus :

```java
    /**
     * Mise à jour  ou récupération dans la BDD en fonction des paramètres fournis
     * @param operation opération à effectuer
     * @param lesDonnees données fournies
     */
    public void accesDonnees(String operation, JSONArray lesDonnees){
        accesDistant.envoi(operation, lesDonnees);
    }
```

Enfin, dans la classe AccesDistant, la méthode « envoi », appelée par le contrôleur est exécutée. Cette méthode s'occupe d'instancier un objet AccesHTTP pour permettre d'ajouter des paramètres à la requête HTTP en POST par l'intermédiaire de la méthode « addParam ». Un extrait de la méthode « addParam » de la classe AccesHTTP est présent sur la seconde capture d'écran. On peut notamment voir que l'on valorise la requête HTTP sous forme de clé/valeur avec les paramètres reçus. Également, la méthode « envoi » s'occupe d'exécuter la requête POST au serveur par l'intermédiaire de la méthode « execute » qui est une méthode de la classe AsyncTask dont hérite la classe AccesHTTP. La méthode « execute » s'occupe notamment d'appeler la méthode « doInBackground » redéfinie dans la classe AccesHTTP et dont le contenu est montré dans la 3ème capture d'écran. Cette méthode s'occupe entre autres de paramétrer la requête HTTP, d'envoyer au serveur la liste de paramètres au serveur et de récupérer le retour du serveur dans la variable « retourServeur » :

```
148         /**
149          * Envoi de données vers le serveur distant
150          * @param operation information précisant au serveur l'opération à exécuter
151          * @param lesDonneesJSON les données à traiter par le serveur
152          */
153     @   public void envoi(String operation, JSONArray lesDonneesJSON){
154             AccesHTTP accesDonnees = new AccesHTTP();
155             // lien avec AccesHTTP pour permettre à delegate d'appeler la méthode processFinish
156             // au retour du serveur
157             accesDonnees.delegate = this;
158             // ajout de paramètres dans l'enveloppe HTTP
159             accesDonnees.addParam( nom: "operation", operation);
160             accesDonnees.addParam( nom: "lesdonnees", lesDonneesJSON.toString());
161             // envoi en post des paramètres, à l'adresse SERVERADDR
162             accesDonnees.execute(SERVERADDR);
163         }
```

```
27      /**
28       * Construction de la chaîne de paramètres à envoyer en POST au serveur
29       * @param nom
30       * @param valeur
31       */
32      public void addParam(String nom, String valeur) {
33          try {
34              if (parametres.equals("")) {
35                  // premier paramètre
36                  parametres = URLEncoder.encode(nom,  enc: "UTF-8") + "=" + URLEncoder.encode(valeur,  enc: "UTF-8");
37              }else{
38                  // paramètres suivants (séparés par &)
39                  parametres += "&" + URLEncoder.encode(nom,  enc: "UTF-8") + "=" + URLEncoder.encode(valeur,  enc: "UTF-8");
40              }
41          } catch (UnsupportedEncodingException e) {
42              e.printStackTrace();
43          }
44      }
```

```java
/**
 * Méthode appelée par la méthode execute
 * permet d'envoyer au serveur une liste de paramètres en POST
 * @param urls contient l'adresse du serveur dans la case 0 de urls
 * @return null
 */
@Override
protected Long doInBackground(String... urls) {
    // pour éliminer certaines erreurs
    System.setProperty("http.keepAlive", "false");
    // objets pour gérer la connexion, la lecture et l'écriture
    PrintWriter writer = null;
    BufferedReader reader = null;
    HttpURLConnection connexion = null;

    try {
        // création de l'url à partir de l'adresse reçu en paramètre, dans urls[0]
        URL url = new URL(urls[0]);
        // ouverture de la connexion
        connexion = (HttpURLConnection) url.openConnection();
        connexion.setDoOutput(true);
        // choix de la méthode POST pour l'envoi des paramètres
        connexion.setRequestMethod("POST");
        connexion.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
        connexion.setFixedLengthStreamingMode(parametres.getBytes().length);
        // création de la requete d'envoi sur la connexion, avec les paramètres
        writer = new PrintWriter(connexion.getOutputStream());
        writer.print(parametres);
        // Une fois l'envoi réalisé, vide le canal d'envoi
        writer.flush();
        // Récupération du retour du serveur
        reader = new BufferedReader(new InputStreamReader(connexion.getInputStream()));
        retourServeur = reader.readLine();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        // fermeture des canaux d'envoi et de réception
        try{
            writer.close();
        }catch(Exception e){}
        try{
            reader.close();
        }catch(Exception e){}
    }
    return null;
}
```

A partir de là, la requête a été envoyée au serveur. Le serveur est en fait un fichier PHP hébergé sur le serveur ou l'application web de gestion des frais, développée pour la mission 1, est hébergée. Tout simplement, le fichier PHP s'occupe de récupérer le contenu de la requête qu'il reçoit, et, en fonction des paramètres et valeurs reçues, d'effectuer des traitements et de retourner si nécessaire une réponse HTTP à l'application Android. Voici un extrait du fichier PHP dans la capture d'écran ci-dessous. Tout d'abord, on peut voir qu'on s'occupe de récupérer le contenu de la requête pour savoir quel traitement est à effectuer. Dans notre cas, on avait envoyé en tant que clé/valeur « operation = recupInfos ». On rentre alors dans la condition if. La méthode « print » permet d'indiquer ce que l'on souhaite renvoyer en tant que réponse à l'application Android. Ici, on renvoie « recupInfos% ». C'est en fait un préfixe du contenu de la réponse qui est renvoyé à l'application Android pour savoir, côté Android, quel traitement sera à effectuer sur la réponse HTTP reçue (je reviendrai sur ce point un peu plus loin). Ensuite, on s'occupe de récupérer le login et le mot de passe envoyés par l'application Android. Puis, on exécute une requête SQL permettant de récupérer

le visiteur qui correspond au login et au mot de passe fourni. Si la requête retourne un résultat, c'est que le couple login/mot de passe est correct et donc on appelle de nouveau la fonction print en encodant le résultat pour l'envoyer à l'application Android. On pourra remarquer que tout le bloc est entouré d'un try, et qu'en cas d'erreur on capture l'erreur dans le bloc catch et on envoie l'erreur à l'application Android pour avoir des informations sur l'erreur :

```php
// Récupération des informations du visiteur
if ($_REQUEST["operation"]=="recupInfos") {

    try {
        print ("recupInfos%");
        // Récupération des données en post
        $lesdonnees = $_REQUEST["lesdonnees"];
        $donnee = json_decode($lesdonnees);
        $login = $donnee[0];
        $mdp = $donnee[1];

        $laRequete = "SELECT * FROM visiteur WHERE login ='";
        $laRequete.= $login ."' AND mdp ='" . $mdp . "'";

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();
        // Si la requête a réussi, on retourne les informations
        if ($ligne = $req->fetch(PDO::FETCH_ASSOC)) {
            print(json_encode($ligne));
        }

    }catch(PDOException $e){
        print "Erreur !%".$e->getMessage();
        die();
    }
```

Lorsque le serveur, en l'occurrence le fichier PHP, renvoie la réponse HTTP, côté Android, c'est la méthode « processFinish » présente dans la classe AccesDistant qui est appelée (par l'intermédiaire de la méthode « onPostExecute » présente dans la classe AccesHTTP). La méthode est décrite sur la capture d'écran ci-dessous. Le paramètre fourni à la méthode processFinish est d'abord découpé comme on peut le voir grâce au caractère de séparation « % ». C'est pour cela que dans le fichier PHP présent dans la capture d'écran ci-dessous on envoyait le préfixe « recupInfos% » qui permet ainsi de savoir quel traitement effectuer. Car en effet, on stocke dans la première case du tableau « message » le traitement à effectuer et dans la deuxième case les valeurs renvoyées par le serveur. On teste ensuite avec des conditions « if » le contenu de la première case du tableau et en fonction du contenu, on effectue les traitements correspondants comme on peut le voir à la ligne 45 sur la capture d'écran ci-dessous. Ainsi, à la ligne 45, puisque que le suffixe de la réponse renvoyée par le serveur était « recupInfos », on va ainsi rentrer dans cette condition. Dans cette condition, on s'occupe de récupérer l'identifiant et le mot de passe renvoyé par le serveur en manipulant un objet au format JSON et de valoriser différentes valeurs du contrôleur. Ensuite, on vérifie dans une dernière condition si le couple crypté login/mot de passe retourné par le serveur correspondent bien au couple crypté login/mot de passe entré par le visiteur. Le contenu de la méthode est détaillé dans la 2ème capture d'écran ci-dessous. Si la condition est valide, alors on redirige le visiteur sur la vue du menu principale grâce à la méthode

« accesMenuActivity » détaillée dans la 3ème capture d'écran qui s'occupe de fermer l'activity sur laquelle on se trouve et d'ouvrir l'activity que l'on passe en paramètre lorsqu'on instancie l'objet de type Intent :

```
32      /**
33       * Retour du serveur HTTP
34       * @param output
35       */
36      @Override
37      public void processFinish(String output) {
38          // pour vérification, affiche le contenu du retour dans la console
39          Log.d( tag: "serveur", msg: "************" + output);
40          // découpage du message reçu
41          String[] message = output.split( regex: "&");
42          // contrôle si le retour est correct (au moins 2 cases)
43          if(message.length>1) {
44              // Récupération de l'id, du mdp et du login du visiteur qui se connecte
45              if (message[0].equals("recupInfos")) {
46                  try{
47                      // Récupération des informations
48                      JSONObject info = new JSONObject(message[1]);
49                      String identifiant = info.getString( name: "id");
50                      String passwordBdd = info.getString( name: "mdp");
51                      String login = info.getString( name: "login");
52                      controle.setIdentifiantVisiteur(identifiant);
53                      controle.setPasswordBdd(passwordBdd);
54                      controle.setLogin(login);
55                      if (controle.coupleLoginPwdCorrect(MainActivity.login, MainActivity.passwordEntre)){
56                          controle.accesMenuActivity();
57                      }
```

```
78      /**
79       * Vérification du login et du mdp entrés par le visiteur lors de l'authentification
80       *
81       * @param login login entré par le visiteur
82       * @param password mdp entré par le visiteur
83       * @return
84       */
85      public boolean coupleLoginPwdCorrect(String login, String password) {
86          // On crypte le mot de passe passé en paramètre
87          String passCrypte = MesOutils.hashString(password);
88
89          // Si le couple identifiantVisiteur/mdp crypté correspond au couple identifiantVisiteur/mdp crypté dans la
90          // bdd, on retourne true
91          if (login.equals(this.login) && passCrypte.equals(this.passwordBdd)) {
92              return true;
93          }
94          return false;
95      }
```

```
97      /**
98       * Permet de lancer la vue MenuActivity
99       */
100     public void accesMenuActivity() {
101         Intent intent = new Intent(this.context, MenuActivity.class);
102         intent.addFlags(intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
103         // startActivity(intent);
104         Controle.this.context.startActivity(intent);
105     }
```

A savoir que si le serveur ne trouve aucun enregistrement lorsqu'il effectue la requête SQL, alors il retourne seulement « recupInfos% ». Ainsi, on remplie la condition présente dans la capture d'écran ci-dessous, toujours dans la méthode « processFinish » qui s'occupe d'afficher un message d'erreur à l'utilisateur :

```
138         } else{
139             /* Si la demande envoyé au serveur était recupInfos et que rien n'est retourné, c'est
140              * que le mdp ou le login était incorrect donc on affiche un message à l'écran
141              */
142             if (message[0].equals("recupInfos") && message.length==1){
143                 MainActivity.txtErrorAuth.setText("Identifiant ou mot de passe incorrect");
144             }
145         }
```

IV) Accès au menu principal

Lorsque la connexion est réussie, le visiteur est donc redirigé sur la vue suivante qui lui permet notamment de naviguer sur les différentes saisies qu'il a à faire :

De la même façon que pour la page de connexion, cette vue possède un fichier XML qui permet de définir les différents composants graphiques à afficher et également un fichier Java permettant de gérer les différents traitements à effectuer. Côté Java, on a toujours la méthode « onCreate » qui est appelée lorsque l'activity MenuActivity est instanciée et qui s'occupe notamment de valoriser la vue à afficher à l'utilisateur et de permettre l'écoute sur les différents boutons du menu grâce à la méthode « cmdMenu_clic ». Comme on peut le voir sur la seconde capture d'écran, la méthode « cmdMenu_clic » s'occupe de créer une écoute sur chaque bouton du menu et, au clic, de lancer la vue correspondant au bouton sur lequel on a cliqué. Également, grâce à la méthode « recupFrais », on récupère les frais forfaits et hors forfaits, pour le mois en cours, du visiteur qui s'est connecté en envoyant une requête HTTP au serveur (comme le montre la 3ème capture d'écran). Côté PHP, on récupère grâce à des requêtes SQL les différents frais et on envoie les résultats à l'application Android (comme le montre la 4ème et 5ème capture d'écran) :

```java
public class MenuActivity extends AppCompatActivity {

    private Controle controle; // Contient l'unique instance du contrôleur

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_menu);
        setTitle("GSB : Suivi des frais");
        // chargement des méthodes événementielles
        cmdMenu_clic(((ImageButton)findViewById(R.id.cmdKm)), KmActivity.class);
        cmdMenu_clic(((ImageButton)findViewById(R.id.cmdHf)), HfActivity.class);
        cmdMenu_clic(((ImageButton)findViewById(R.id.cmdHfRecap)), HfRecapActivity.class);
        cmdMenu_clic(((ImageButton)findViewById(R.id.cmdNuitee)), NuiteeActivity.class);
        cmdMenu_clic(((ImageButton)findViewById(R.id.cmdEtape)), EtapeActivity.class);
        cmdMenu_clic(((ImageButton)findViewById(R.id.cmdRepas)), RepasActivity.class);
        // Récupération du contrôleur
        this.controle = Controle.getInstance(null);
        recupFrais();
    }
```

```java
    /**
     * Sur la sélection d'un bouton dans l'activité principale ouverture de l'activité correspondante
     */
    private void cmdMenu_clic(ImageButton button, final Class classe) {
        button.setOnClickListener((v) -> {
                // ouvre l'activité
                Intent intent = new Intent( packageContext: MenuActivity.this, classe);
                intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                startActivity(intent);
        });
    }
```

```java
    /**
     * Récupération des frais forfaits du visiteur
     */
    private void recupFrais(){
        // Récupération des frais forfaits et hors forfaits du visiteur
        controle.accesDonnees( operation: "getFraisHF", convertToJSONArray());
        controle.accesDonnees( operation: "getFraisForfait", convertToJSONArray());
    }
```

```php
} elseif ($_REQUEST["operation"]=="getFraisForfait") {

    try{
        // récupération des données en post
        print("getFraisForfait%");
        $lesdonnees = $_REQUEST["lesdonnees"];
        $donnee = json_decode($lesdonnees);
        $idVisiteur = $donnee[0];
        $mois = $donnee[1];

        if ($pdo->estPremierFraisMois($idVisiteur, $mois)) {
            $pdo->creeNouvellesLignesFrais($idVisiteur, $mois);
        };

        // Préparation de la requête
        $laRequete = "SELECT * FROM lignefraisforfait WHERE idvisiteur ='";
        $laRequete .= $idVisiteur . "' AND mois = '" . $mois . "'";

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();
        $ligne = $req->fetchAll();
        print(json_encode($ligne));

    }catch(PDOException $e){
        print "Erreur !%".$e->getMessage();
        die();
    }
```

```php
} elseif ($_REQUEST["operation"]=="getFraisHF") {

    try{
        // récupération des données en post
        print("getFraisHF%");
        $lesdonnees = $_REQUEST["lesdonnees"];
        $donnee = json_decode($lesdonnees);
        $idVisiteur = $donnee[0];
        $mois = $donnee[1];

        if ($pdo->estPremierFraisMois($idVisiteur, $mois)) {
            $pdo->creeNouvellesLignesFrais($idVisiteur, $mois);
        };

        // Préparation de la requête
        $laRequete = "SELECT * FROM lignefraishorsforfait WHERE idvisiteur ='";
        $laRequete.= $idVisiteur . "' AND mois = '" . $mois . "'";

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();
        $ligne = $req->fetchAll(PDO::FETCH_ASSOC);

        print(json_encode(
            $ligne,
            JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_SLASHES
        ));

    }catch(PDOException $e){
        print "Erreur !%".$e->getMessage();
        die();
    }
```

Côté Android, on récupère la réponse du serveur par l'intermédiaire de la méthode « processFinish » Comme on peut le voir sur la première capture d'écran ci-dessous, on s'occupe de manipuler le tableau reçu pour valoriser la variable locale lesFraisForfait pour ensuite l'envoyer au contrôleur en paramètre de la méthode « setLesFraisForfait ». Le contenu de la méthode « setLesFraisForfait » est très simple, elle s'occupe tout simplement de valoriser la propriété locale « lesFraisForfait » de la classe « controleur » qui est un dictionnaire qui contiendra tous les frais forfaits pour la session en cours. Ensuite, comme on le voit sur la seconde capture d'écran ci-dessous, on récupère grâce à une boucle for chaque frais hors forfaits et on stocke chaque frais hors forfaits dans un tableau de type « FraisHf » en instanciant dans chaque tour de boucle un objet de type « FraisHf ». La classe « FraisHf » est une classe modèle comme le montre la 3ème capture d'écran. Une fois que la boucle for est terminée, on valorise le tableau « lesFraisHF » dans la classe « controleur » qui contient tous les frais hors forfaits pour la session en cours par l'intermédiaire de la méthode « setLesFraisHorsForfait » présente à la ligne 101 sur la seconde capture d'écran.

```
62          // Récupération des frais forfaits du visiteur pour le mois actuel
63          if (message[0].equals("getFraisForfait")) {
64              try {
65                  // récupération des informations
66                  JSONArray lesInfos = new JSONArray(message[1]);
67                  Hashtable<String, Integer> lesFraisForfait = new Hashtable<>();
68                  for(int k=0 ; k<lesInfos.length() ; k++){
69                      JSONObject info = new JSONObject(lesInfos.get(k).toString());
70                      String idFrais = info.getString( name: "idfraisforfait");
71                      Integer quantite = info.getInt( name: "quantite");
72                      lesFraisForfait.put(idFrais, quantite);
73                  }
74                  // mémorisation des frais forfait
75                  controle.setLesFraisForfait(lesFraisForfait);
76              } catch (JSONException e) {
77                  e.printStackTrace();
78              }
```

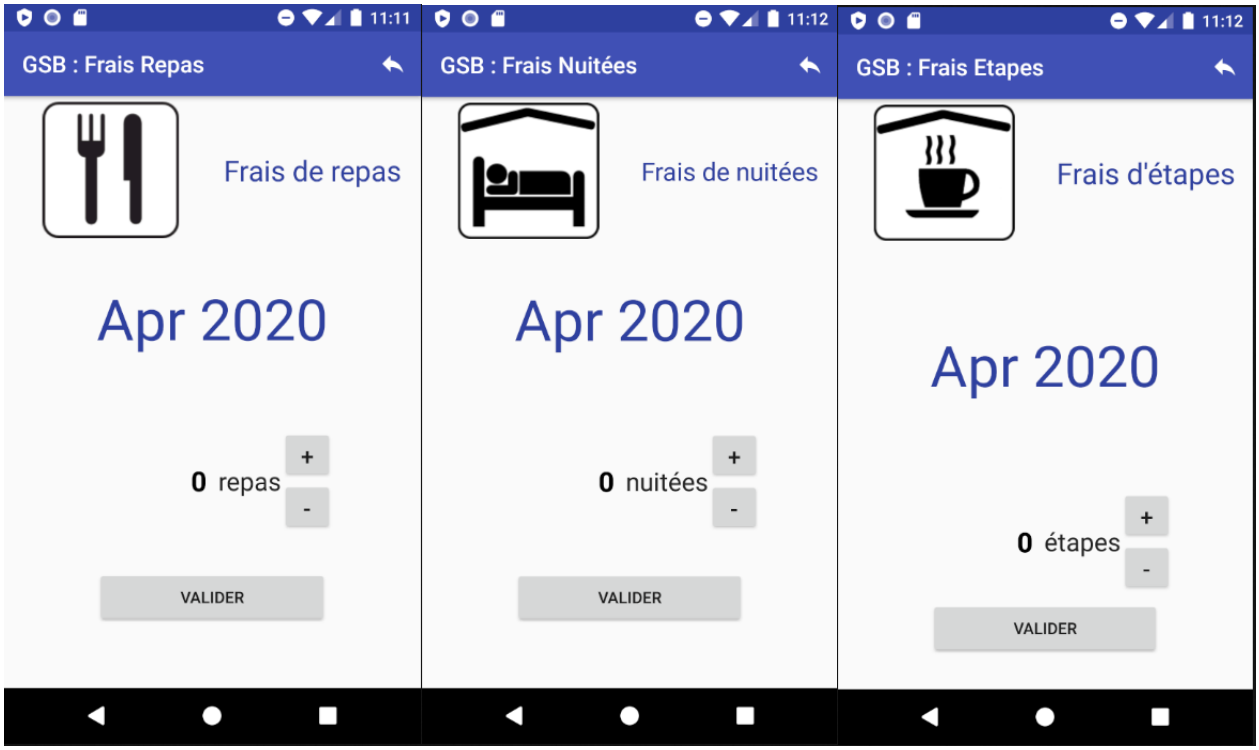
```
80          // Récupération des frais hors forfait du visiteur pour le mois actuel
81          if (message[0].equals("getFraisHF")) {
82              try {
83                  // récupération des informations
84                  JSONArray lesInfos = new JSONArray(message[1]);
85                  ArrayList<FraisHf> lesFraisHf = new ArrayList<>();
86                  int identifiant = 0;
87                  for(int k=0 ; k<lesInfos.length() ; k++){
88                      JSONObject info = new JSONObject(lesInfos.get(k).toString());
89                      Integer idFraisHF = info.getInt( name: "id");
90                      Double montant = info.getDouble( name: "montant");
91                      String libelle = info.getString( name: "libelle");
92                      String jour = (info.getString( name: "date")).substring(8, 10);
93                      FraisHf unFraisHF = new FraisHf(montant, libelle, jour, identifiant, idFraisHF);
94                      lesFraisHf.add(unFraisHF);
95                      // Mémorisation du dernier id frais HF ajouté dans la BDD
96                      controle.setDernierIdFraisHf(identifiant);
97                      identifiant++;
98                  }
99
100                 // mémorisation des frais hors forfait
101                 controle.setLesFraisHorsForfait(lesFraisHf);
102             } catch (JSONException e) {
103                 e.printStackTrace();
104             }
```

```java
/**
 * Classe métier contenant la description d'un frais hors forfait
 *
 */
public class FraisHf implements Comparable {

    private final Double montant;
    private final String libelle;
    private final String jour;
    private final int identifiant; // identifiant incrémenté de 1 à chaque frais HF ajoutés dans l'appli
    public Integer idFraisDansBdd; // id du frais HF dans la BDD

    public FraisHf(Double montant, String libelle, String jour, int identifiant, Integer idBdd) {
        this.montant = montant ;
        this.libelle = libelle ;
        this.jour = jour ;
        this.identifiant = identifiant;
        this.idFraisDansBdd = idBdd;
    }

    public double getMontant() { return montant; }

    public String getLibelle() { return libelle; }

    public String getJour() { return jour; }

    public int getIdentifiant() { return identifiant; }

    public Integer getIdFraisDansBdd() { return idFraisDansBdd; }
```

A partir de là, tous les frais forfaits et hors forfaits correspondant au visiteur connecté sont récupérés dans la base de données et sont manipulables par l'application Android.

V) Gestion des frais forfaits

La vue correspondante à la saisie des kilomètres effectués était déjà codée. J'ai donc codé les 3 autres vues correspondantes à la saisie des frais forfaits à savoir la saisie des nuitées, des frais d'étapes et des frais de repas. Ces différentes vues de saisies permettent de saisir les différents frais pour les visiteurs. Etant donné que l'application web et l'application Android partagent la même base de données, si vous entrez des valeurs dans l'application Android pour un visiteur donné et que vous allez ensuite sur l'application web en vous connectant avec le même visiteur, vous pourrez voir que les valeurs sont synchronisées. Voici des captures d'écran des différentes vues permettant les saisies :



Etant donné que les traitements et le code Java correspondant pour chacune des activity permettant la saisie des frais forfaits suivent la même logique et qu'uniquement le nom des propriétés changent, je ne vais détailler que la vue permettant la saisie des frais de repas, sachant que le code qui va être détaillé s'applique donc aux 3 autres vues (saisie de nuitées, d'étapes et de kilomètres).

Tout comme les autres vues déjà présentées précédemment, la vue permettant la saisie des frais de repas est reliée à un fichier XML et un fichier Java. Comme pour chaque vue, on a dans le fichier Java une méthode onCreate qui permet d'initialiser les différentes méthodes et propriétés de la vue. La méthode onCreate est détaillée dans la capture d'écran ci-dessous. On peut notamment voir qu'on récupère le mois et l'année en fonction de la date actuelle. On appelle également la méthode « valoriseProprietes » qui permet de valoriser les différentes propriétés à afficher au visiteur. Cette méthode est détaillée dans la seconde capture d'écran. On peut notamment voir qu'on valorise la date du jour qu'on affiche à l'utilisateur et qu'on affiche également la quantité de nuitées qu'on a déjà saisi lors de la dernière visite dans l'application et qui a au préalable était récupérée dans la base de données. Aussi, on place les différentes méthodes permettant l'écoute sur les différents composants graphiques de la vue. On a par exemple la méthode « cmdValider_clic » qui permet de créer une écoute sur le bouton valider de la vue et qui enregistre notamment dans la base de données la quantité de nuitées saisies par l'utilisateur. Le détail de cette méthode est montré dans la 3ème capture d'écran ci-dessous. La procédure est la même que d'habitude, on effectue un appel au serveur qui va s'occuper, par l'intermédiaire d'une requête SQL, de mettre à jour la base de données (dont le détail est présent sur la 4ème capture d'écran) :

```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_nuitee);
    setTitle("GSB : Frais Nuitées");
    this.txtDateNuitee = findViewById(R.id.txtDateNuitee);
    this.moisMMM = MesOutils.actualMonth(new Date());
    this.moisMM = MesOutils.actualMoisInNumeric(new Date());
    this.annee = MesOutils.actualYear(new Date());
    controle = Controle.getInstance(null);
    // valorisation des propriétés
    valoriseProprietes();
    // chargement des méthodes événementielles
    imgReturn_clic();
    cmdValider_clic();
    cmdPlus_clic();
    cmdMoins_clic();
}
```

```java
/**
 * Valorisation des propriétés
 */
private void valoriseProprietes() {
    // Affichage de la date actuelle
    String date = moisMMM + " " + annee;
    this.txtDateNuitee.setText(date);
    // récupération de la quantité pour ce frais
    qte = controle.getUnFraisForfait(idFrais);
    ((TextView)findViewById(R.id.txtNuitee)).setText(String.format(Locale.FRANCE, format: "%d", qte));
}
```

```java
/**
 * Sur le clic du bouton valider : enregistrement du frais forfait dans la BDD
 */
private void cmdValider_clic() {
    findViewById(R.id.cmdNuiteeValider).setOnClickListener((v) -> {
            controle.accesDonnees( operation: "updateFraisForfait", convertToJSONArray());
            retourActivityPrincipale();
    });
}
```

```php
} elseif ($_REQUEST["operation"]=="updateFraisForfait") {

    try{
        // récupération des données en post
        print("updateFraisForfait%");
        $lesdonnees = $_REQUEST["lesdonnees"];
        $donnee = json_decode($lesdonnees);
        $idVisiteur = $donnee[0];
        $mois = $donnee[1];
        $idFrais = $donnee[2];
        $quantite = $donnee[3];

        if ($pdo->estPremierFraisMois($idVisiteur, $mois)) {
            $pdo->creeNouvellesLignesFrais($idVisiteur, $mois);
        };


        // Préparation de la requête
        $laRequete = "UPDATE lignefraisforfait SET quantite =" .$quantite;
        $laRequete .= " WHERE mois = '" . $mois . "' AND idvisiteur = '" ;
        $laRequete .= $idVisiteur . "' AND idfraisforfait = '" . $idFrais . "'";

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();
        print(json_encode($laRequete));


    }catch(PDOException $e){
        print "Erreur !%".$e->getMessage();
        die();
    }
```

VI) Gestion des frais hors forfaits

La vue correspondant à la saisie des frais hors forfaits permet au visiteur de saisir des frais hors forfaits. De la même façon que pour les vues permettant la saisie des frais forfaits, lorsque qu'un visiteur saisi des frais hors forfaits, cela sera synchronisé avec l'application web étant donné que les deux applications partagent la même base de données. Voici l'aperçu de la vue qui permet de saisir des frais hors forfaits :

Cette vue ainsi que les fonctionnalités correspondantes étaient déjà présentes dans le code source mis à part l'appel à la base de données pour enregistrer le nouveau frais hors forfaits saisis. Ainsi, au clic sur le bouton « ajouter », on fait appel à la méthode « cmdAjouter_clic » (présentée dans la première capture d'écran ci-dessous) qui s'occupe d'appeler le serveur distant et d'effectuer une requête SQL permettant d'insérer le frais hors forfaits saisi (présentée dans la seconde capture d'écran) :

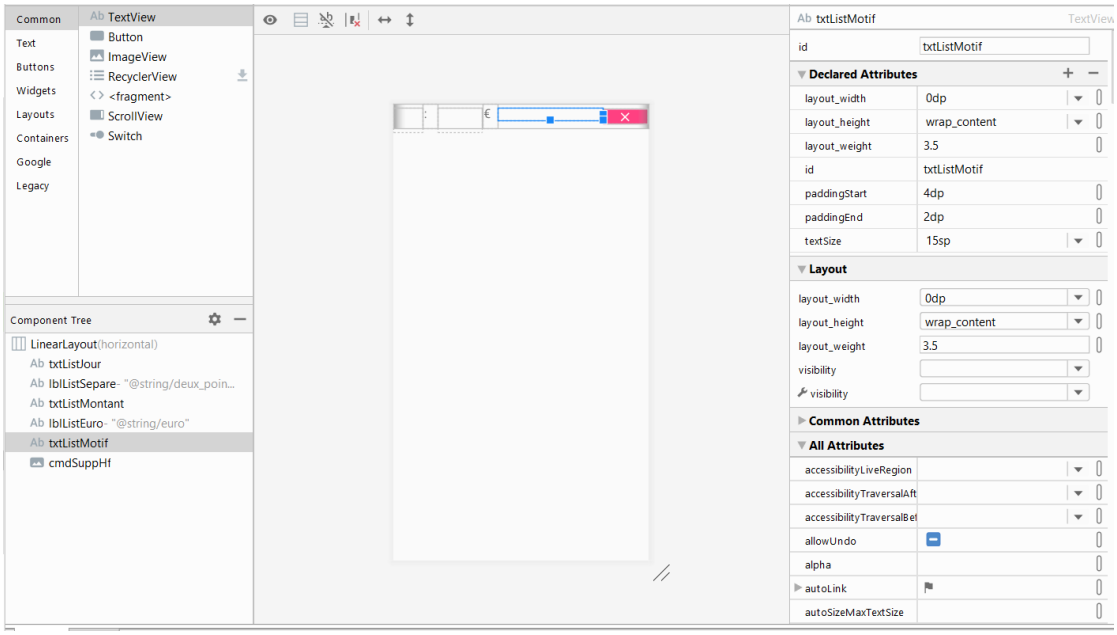
```java
/**
 * Sur le clic du bouton ajouter : enregistrement dans la liste des fraisHf di frais HF ajouté
 * et ajout du frais HF dans la BDD
 */
private void cmdAjouter_clic() {
    final String lib = this.libelle;
    findViewById(R.id.cmdHfAjouter).setOnClickListener((v) -> {
            enregListe();
            controle.accesDonnees( operation: "ajoutFraisHF", convertToJSONArray());
            retourActivityPrincipale();
    }) ;
}
```

```php
} elseif ($_REQUEST["operation"]=="ajoutFraisHF") {

    try{
        // récupération des données en post
        print("ajoutFraisHF%");
        $lesdonnees = $_REQUEST["lesdonnees"];
        $donnee = json_decode($lesdonnees);
        $idVisiteur = $donnee[0];
        $mois = $donnee[1];
        $libelle = $donnee[2];
        $date = $donnee[3];
        $montant = $donnee[4];

        if ($pdo->estPremierFraisMois($idVisiteur, $mois)) {
            $pdo->creeNouvellesLignesFrais($idVisiteur, $mois);
        };

        // Préparation de la requête
        $laRequete = "INSERT INTO lignefraishorsforfait (idvisiteur, ";
        $laRequete .= "mois, libelle, date, montant) ";
        $laRequete .= "VALUES ('".$idVisiteur."', '".$mois."', '".$libelle;
        $laRequete .= "', '".$date. "', ".$montant.")";

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();

        /* Dès que l'ajout est effectué, on récupère le dernier id du frais
         * hors forfait de la table lignefraishorsforfait et on le retourne
         */
        $laRequete = "SELECT idMax, libelle, date, montant ";
        $laRequete ."FROM (SELECT MAX(id) AS 'idMax', libelle, date, montant ";
        $laRequete .= "FROM lignefraishorsforfait ";
        $laRequete .= "GROUP BY libelle, date, montant) AS req ";
        $laRequete .= "WHERE idMax = (SELECT MAX(id) ";
        $laRequete .= " FROM lignefraishorsforfait)";

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();
        $ligne = $req->fetch(PDO::FETCH_ASSOC);
        print(json_encode($ligne));
```

VII) Récapitulatif des frais hors forfaits

Cette vue qui permet d'afficher la liste de tous les frais hors forfaits saisis pour le mois en cours par le visiteur était déjà présents dans le code source mis à part la possibilité de supprimer une ligne.

Cette vue présente une particularité : elle possède un list adapter. Un list adapter permet de pouvoir dérouler une liste comme lorsqu'on est sur un smartphone et qu'on déroule une liste avec le glissement du doigt vers le haut ou le bas. Pour gérer cela, il faut créer un modèle de ligne du list adapter à insérer dans l'activity qui affiche le récapitulatif. Voici le modèle créé en interface graphique :



Ainsi, lorsqu'on instancie la vue du récapitulatif des frais hors forfaits, on instancie également ce list adapter dans la méthode « onCreate » qui appelle la méthode « afficheListe » et qui s'occupe de d'instancier et d'afficher le list adapter :

```
62      /**
63       * Affiche la liste des frais hors forfaits de la date sélectionnée
64       */
65      private void afficheListe() {
66          ListView listView = (ListView)findViewById(R.id.lstHfRecap);
67          Collections.sort(controle.getLesFraisHF(), Collections.<~>reverseOrder());
68          FraisHfAdapter adapter = new FraisHfAdapter( context: HfRecapActivity.this, controle.getLesFraisHF());
69          listView.setAdapter(adapter);
70      }
```

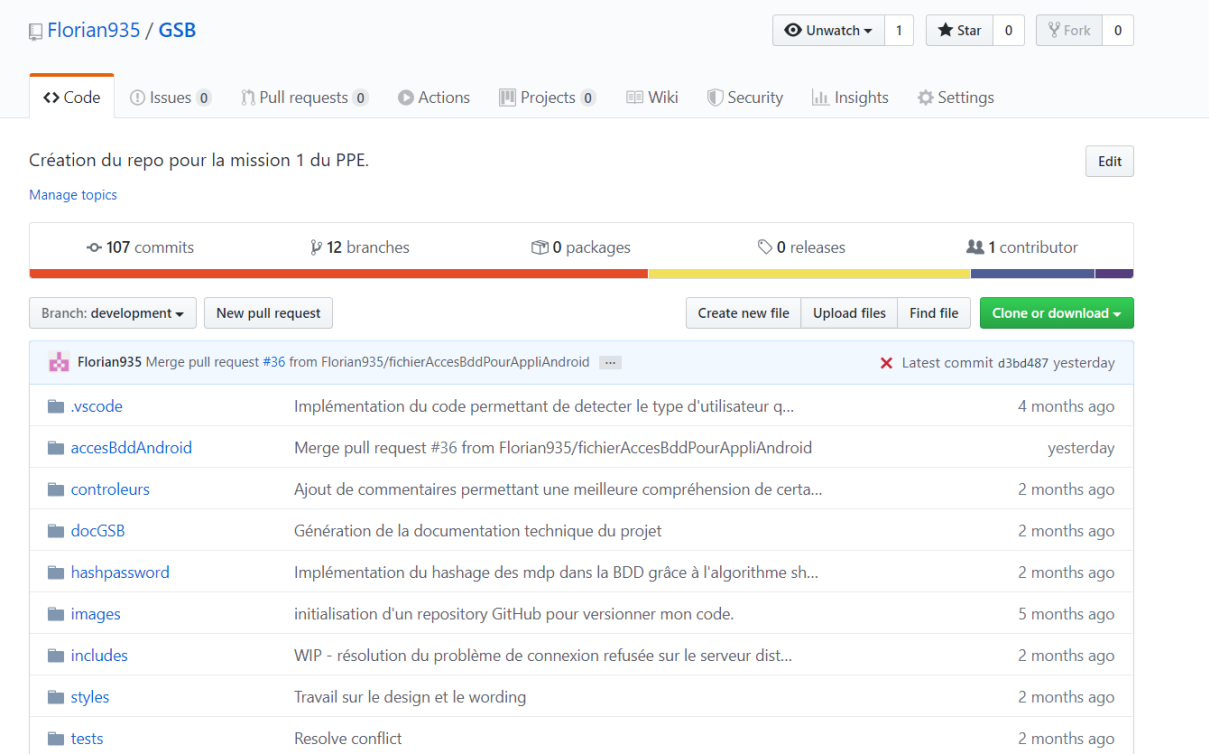
Dans la classe « FraisHfAdapter » qui s'occupe donc de gérer la liste des différents frais hors forfaits, on a notamment la méthode « getView », présente dans la capture d'écran ci-dessous, qui s'occupe spécifiquement d'afficher chaque ligne. Ainsi, sur chaque ligne on place un listener sur le bouton supprimer (qui correspond à la croix blanche sur fond rouge). Ainsi, lors du clic sur la croix rouge, on s'occupe d'appeler le serveur distant qui supprime l'enregistrement correspondant dans la base de données (comme montré dans la seconde capture d'écran). La méthode « notifyDataSetChanged » à la ligne 98 de la première capture d'écran s'occupe de mettre à jour en direct la liste des frais hors forfaits, c'est-à-dire d'afficher la liste désormais sans le frais supprimés :

```
65      /**
66       * Affichage dans la liste
67       */
68      @Override
69      public View getView(int index, View convertView, ViewGroup parent) {
70          final FraisHf FraisHfActuel = (FraisHf)this.getItem(index);
71          final Integer idFraisHF = FraisHfActuel.getIdFraisDansBdd();
72          ViewHolder holder;
73
74          if (convertView == null) {
75              holder = new ViewHolder();
76              convertView = inflater.inflate(R.layout.layout_liste, parent, attachToRoot: false);
77              holder.txtListJour = convertView.findViewById(R.id.txtListJour);
78              holder.txtListMontant = convertView.findViewById(R.id.txtListMontant);
79              holder.txtListMotif = convertView.findViewById(R.id.txtListMotif);
80              holder.cmdSuppHf = convertView.findViewById(R.id.cmdSuppHf);
81              convertView.setTag(holder);
82          }else{
83              holder = (ViewHolder)convertView.getTag();
84          }
85          holder.txtListJour.setText(lesFraisHf.get(index).getJour());
86          holder.txtListMontant.setText(String.format(Locale.FRANCE, format: "%.2f", lesFraisHf.get(index).getMontant()));
87          holder.txtListMotif.setText(lesFraisHf.get(index).getLibelle());
88          holder.cmdSuppHf.setTag(index);
89          // gestion de l'événement clic sur le bouton de suppression
90          holder.cmdSuppHf.setOnClickListener((v) → {
93                  int indice = (int)v.getTag();
94                  // Suppression du frais HF sur lequel on clic dans la BDD et dans la liste
95                  controle.supprFraisHF( operation: "suppressionFraisHF", convertToJSONArray(idFraisHF), FraisHfActuel);
96                  // Rafraichi la liste visuelle
97                  notifyDataSetChanged();
98          });
100         return convertView;
101     }
```

```php
} elseif ($_REQUEST["operation"]=="suppressionFraisHF") {

    try{
        // récupération des données en post
        print("suppressionFraisHF%");
        $lesdonnees = $_REQUEST["lesdonnees"];
        $donnee = json_decode($lesdonnees);
        $idFraisHF = $donnee[0];

        $laRequete = "SELECT id FROM lignefraishorsforfait ";
        $laRequete .= "WHERE id = " . $idFraisHF;

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();
        $ligne = $req->fetch(PDO::FETCH_ASSOC);
        print(json_encode($ligne));

        // Préparation de la requête
        $laRequete = "DELETE FROM lignefraishorsforfait ";
        $laRequete .= "WHERE id = " . $idFraisHF;

        // Execution et envoi de la requête
        $cnx = PdoGsb::getMonPdo('PdoGsb');
        $req = $cnx->prepare($laRequete);
        $req->execute();


    }catch(PDOException $e){
        print "Erreur !%".$e->getMessage();
        die();
    }
}
```

VIII) Gestion de version du projet

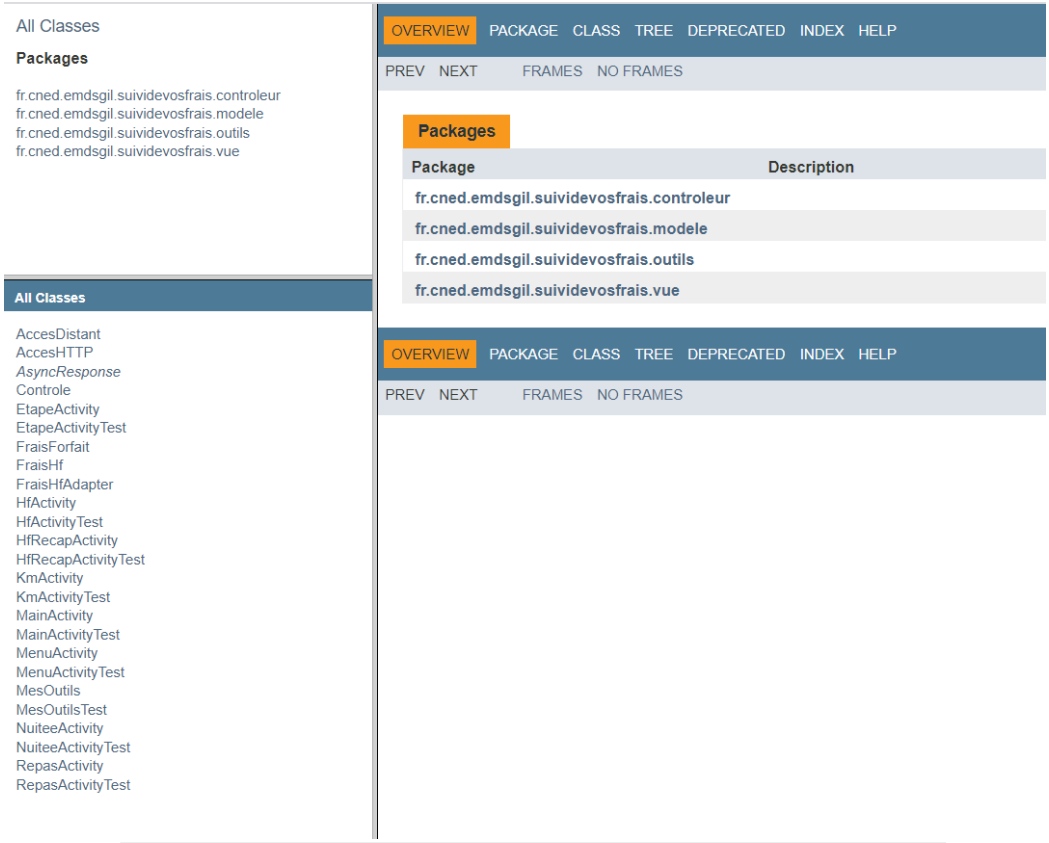
Ce projet a été réalisé avec Git qui permet la gestion du versionning de l'application, en association avec le service en ligne GitHub. Ainsi, cela m'a permis de gérer les différentes versions de l'application, de pouvoir faire des retours en arrière en cas d'erreurs et de surtout sécuriser mon projet ou cas où j'aurais eu des problèmes de matériel par exemple.



IX) Documentation technique

La documentation technique a été générée directement par l'intermédiaire de l'IDE Android Studio qui génère automatiquement la documentation à partir du moment ou les commentaires et annotations sur les différentes méthodes sont correctement implémentées.

Ci-dessous, un extrait de la documentation :



Method Detail

processFinish

```
public void processFinish(java.lang.String output)
```

Retour du serveur HTTP

Specified by:

`processFinish` in interface `AsyncResponse`

Parameters:

`output` -

envoi

```
public void envoi(java.lang.String operation,
                  org.json.JSONArray lesDonneesJSON)
```

Envoi de données vers le serveur distant

Parameters:

`operation` - information précisant au serveur l'opération à exécuter

`lesDonneesJSON` - les données à traiter par le serveur

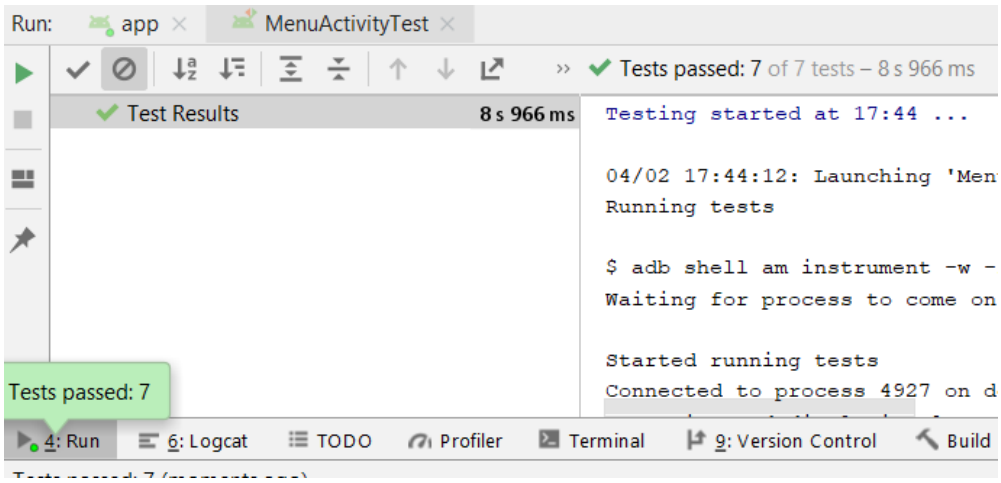
All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static java.lang.String	**actualDayOfMonth**(java.util.Date uneDate) Retourne le jour du mois de la date passée en paramètre au format numérique
static java.lang.String	**actualMoisInNumeric**(java.util.Date uneDate) Retourne le mois de la date passée en paramètre au format numérique
static java.lang.String	**actualMonth**(java.util.Date uneDate) Retourne le mois de la date passée en paramètre au format littéral
static java.lang.String	**actualYear**(java.util.Date uneDate) Retourne l'année de la date passée en paramètre
static java.lang.String	**byteToHex**(byte[] bits) Permet d'effectuer une conversion d'un tableau de bits au format hexadécimale pour finalement retourné le tableau de bits sous forme de chaîne de caractère String
static java.lang.String	**convertDateToString**(java.util.Date uneDate) Conversion d'une date au format Date vers le format String
static java.lang.String	**convertIntDayToStringDay**(java.lang.Integer day) Permet de convertir un entier correspondant à un jour d'un mois au format String sous la forme dd
static java.lang.String	**hashString**(java.lang.String chaine) Permet de hasher la chaîne de caractère fournie en paramètre grâce à l'algorithme de hashage SHA-256

X) Tests

Des tests ont été écrit pour l'application afin de tester différentes valeurs mais aussi différentes activity. Voici des extraits :

Ici, on peut voir que tous les tests sur la classe MenuActivityTest sont passés (première capture d'écran). Voici un extrait, sur les captures d'écran qui suivent la première, de la classe MenuActivityTest qui permet de simuler et de tester, au clic sur les différents boutons graphiques du menu, si les vues correspondantes s'ouvrent :



```java
public class MenuActivityTest {

    // Activity qui sera lancée pour chaque tests
    @Rule
    public ActivityTestRule<MenuActivity> menuActivityActivityTestRule = new ActivityTestRule<~>(MenuActivity.class);

    private MenuActivity menuActivity = null;

    // Valorisation des monitor permettant de réaliser la simulation des ouvertures des Activity souhaitées
    Instrumentation.ActivityMonitor monitorEtape = getInstrumentation().addMonitor(EtapeActivity.class.getName(), result: null, block: false);
    Instrumentation.ActivityMonitor monitorNuitee = getInstrumentation().addMonitor(NuiteeActivity.class.getName(), result: null, block: false);
    Instrumentation.ActivityMonitor monitorKm = getInstrumentation().addMonitor(KmActivity.class.getName(), result: null, block: false);
    Instrumentation.ActivityMonitor monitorRepas = getInstrumentation().addMonitor(RepasActivity.class.getName(), result: null, block: false);
    Instrumentation.ActivityMonitor monitorFraisHf = getInstrumentation().addMonitor(HfActivity.class.getName(), result: null, block: false);
    Instrumentation.ActivityMonitor monitorFraisHfRecap = getInstrumentation().addMonitor(HfRecapActivity.class.getName(), result: null, block: false);

    /**
     * Méthode appelée avant chaque test
     * @throws Exception
     */
    @Before
    public void setUp() throws Exception {
        menuActivity = menuActivityActivityTestRule.getActivity();
    }
```

```java
    /**
     * Teste si à la création de l'activity menu, tous les objets graphiques sont bien créés
     */
    @Test
    public void onCreate() {
        ImageButton cmdKm = menuActivity.findViewById(R.id.cmdKm);
        ImageButton cmdRepas = menuActivity.findViewById(R.id.cmdRepas);
        ImageButton cmdNuitee = menuActivity.findViewById(R.id.cmdNuitee);
        ImageButton cmdEtape = menuActivity.findViewById(R.id.cmdEtape);
        ImageButton cmdHf = menuActivity.findViewById(R.id.cmdHf);
        ImageButton cmdHfRecap = menuActivity.findViewById(R.id.cmdHfRecap);

        assertNotNull(cmdKm);
        assertNotNull(cmdRepas);
        assertNotNull(cmdNuitee);
        assertNotNull(cmdEtape);
        assertNotNull(cmdHf);
        assertNotNull(cmdHfRecap);
    }
```

```java
    /**
     * Teste si au clic sur le bouton permettant d'accéder à la vue des frais d'étapes, l'activity
     * correspondante s'ouvre
     */
    @Test
    public void testLaunchOfEtapeActivityOnButtonClick() {
        assertNotNull(menuActivity.findViewById(R.id.cmdEtape));

        onView(withId(R.id.cmdEtape)).perform(click());
        Activity etapeActivity = getInstrumentation().waitForMonitorWithTimeout(monitorEtape, timeOut: 5000);

        assertNotNull(etapeActivity);

        etapeActivity.finish();
    }
```

Avec la même logique, des tests ont été effectués sur les autres vues.

Ensuite, des tests unitaires ont été écrits pour tester les valeurs de retours de certaines méthodes de la classe « MesOutils » :

```java
public class MesOutilsTest {

    String date = (new SimpleDateFormat( pattern: "yyyy-MMM-dd")).format(new Date());
    String moisActuelInNumber = (new SimpleDateFormat( pattern: "MM")).format(new Date());

    @Before
    public void setUp() throws Exception {
    }

    @Test
    public void convertDateToString() {
        String dateRetournee = MesOutils.convertDateToString(new Date());

        assertEquals(date, dateRetournee);
    }

    @Test
    public void actualDayOfMonth() {
        String jourAttendu = date.substring(11, 13);
        String jourRetourne = MesOutils.actualDayOfMonth(new Date());

        assertEquals(jourAttendu, jourRetourne);
    }

    @Test
    public void actualMonth() {
        String moisAttendu = date.substring(5, 8);
        String moisRetourne = MesOutils.actualMonth(new Date());

        assertEquals(moisAttendu, moisRetourne);
    }

    @Test
    public void actualYear() {
        String anneeAttendue = date.substring(0, 4);
        String anneeRetournee = MesOutils.actualYear(new Date());

        assertEquals(anneeAttendue, anneeRetournee);
    }
```

Et on peut ainsi voir que les tests ont réussi :