GSB Comptable Projet Application de Gestion de frais

-

Application Android

Sommaire Projet

I – Présentation

Pour ce projet j'ai dû créer une application Android pour le compte des laboratoires pharmaceutiques Galaxy Swiss Bourdin consistant à la gestion des notes de frais de salariés, destinées au comptable de l'entreprise. J'ai dans un premier temps créé l'application Web disponible ici : https://github.com/G-alexis-Fr/gsb_comptable.

J'ai développé cette application Android avec l'IDE Android Studio avec le langage Java, et une toute petite partie en PHP développé avec Visual Studio Code.

Dans un premier temps le travail a été effectué dans un environnement local, avec un serveur Wamp puis en fin de projet, j'ai migré vers la base de données sur mon Serveur VPS.

Vous retrouverez le contexte de ce projet ici : https://github.com/G-alexis-Fr/gsb_comptable-android/blob/master/GSB-AppliFrais-Fiche%20Descriptive.pdf

Accès au code :

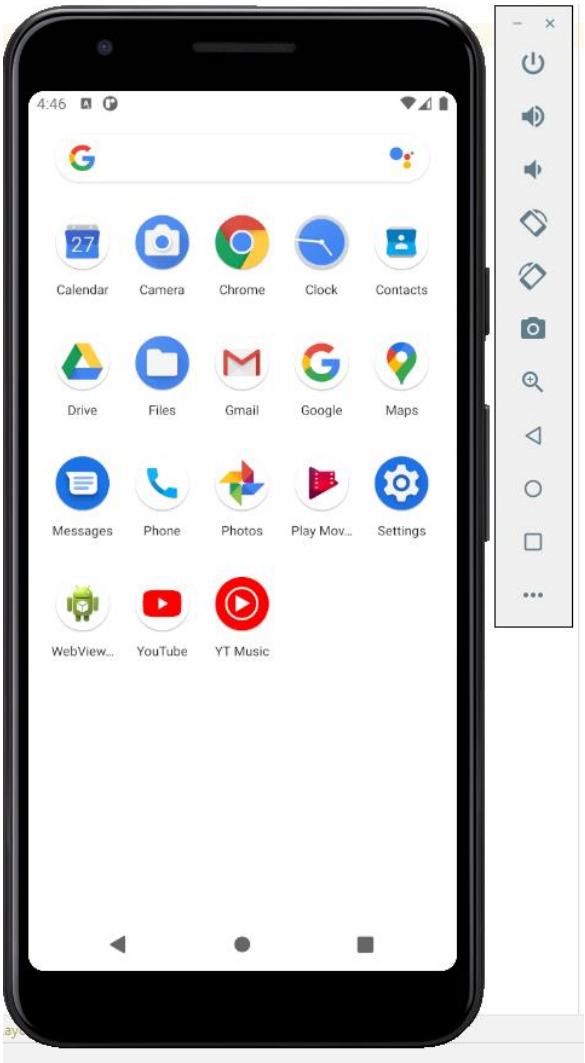
- Code source Github : https://github.com/G-alexis-Fr/gsb_comptable-android
- Accès partie visiteur :
    - Identifiant : dandre
    - Mot de passe : oppg5
- Documentation : http://gsbdocumentation-android.g-alexis.com

Pour ce projet, du code nous avait été fourni, comme la page d'accueil que vous venez de voir, mais également le récapitulatif des frais Hors Forfait, la page des frais Kilométriques. Les personnes ayant travaillés sur le projet sont mentionnées comme il se doit en haut de chaque fichier.

Vous retrouverez tout le code sur mon github, avec une release APK qui vous permettra d'essayer l'application.

Je précise également que j'ai travaillé avec un émulateur sous Android Studio avec les caractéristiques suivantes :

- Modèle téléphone : Pixel 3a
- Résolution : 1080 x 2220 : 440dpi
- API : 30
- Android V. : 11.0 (Google APIs)
- CPU : x86



II – Restriction des saisies

Nous devions dans un premier temps modifier la façon de saisir le nombre de Km qu'un utilisateur faisait, auparavant la personne saisissant un nombre, j'ai donc modifié ceci par un **TextView** au lieu d'un **EditText**. Donc plus la possibilité de changer manuellement ce nombre.

Donc en utilisant les boutons + / - vous pouvez augmenter le compteur pour les Km et ensuite le valider.

III – Nouvelles catégories de frais forfaitisés

Nous avions pour consignes de créer d'autres **Vues** pour les saisies de frais forfaitisés, pour rester dans le même design, j'ai dans un premier temps copie les parties XML puis par la suite adaptes les noms des vues, des widgets et de tous les contenus présents dans chaque vue.

Voici un aperçu du code du fichier activity_etp.xml :

(40 premières lignes du fichier qui en contient 170)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:orientation="vertical"
    android:weightSum="1"
    tools:context=".EtapeActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.25"
        android:orientation="horizontal"
        android:padding="5dp"
        android:weightSum="1"
        android:baselineAligned="false">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="end"
            android:layout_weight="0.5"
            android:orientation="horizontal">

            <TextView
                android:id="@+id/txtTitleEtp"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:autoSizeTextType="uniform"
```

Plusieurs autres pages ont étés créés de la même manière, encore une fois pour respecter le design mais également pour un gain de temps n'ayant pas tout à ressaisir.

IV – Retrait des frais hors forfait

Plusieurs choses nous était demandé dans cette partie du projet, il fallait dans un premier temps ajouter un bouton permettant de retirer une ligne de frais Hors Forfait.

Il fallait également utiliser un **ListView** pour afficher tous les frais Hors Forfait. Et bien sûr, si l'utilisateur supprime la ligne, et faut que l'affichage suive donc nous supprimons celle-ci a l'écran.

J'ai modifié le fichier FraisHfAdapter.java en ajoutant un bouton **cmdSuppHf** dans la classe privée ViewHolder()

```
private class ViewHolder {
   TextView txtListJour;
   TextView txtListMontant;
   TextView txtListMotif;
   ImageButton cmdSuppHf; // Bouton cmdSuppHf
}

@Override
public View getView(int index, View convertView, ViewGroup parent) {
   ViewHolder holder;
   if (convertView == null) {
      holder = new ViewHolder();
      convertView = inflater.inflate(R.layout.layout_liste, parent, false);
      holder.txtListJour = convertView.findViewById(R.id.txtListJour);
      holder.txtListMontant = convertView.findViewById(R.id.txtListMontant);
      holder.txtListMotif = convertView.findViewById(R.id.txtListMotif);
      holder.cmdSuppHf = convertView.findViewById(R.id.cmdSuppHf);
      convertView.setTag(holder);
   } else {
      holder = (ViewHolder) convertView.getTag();
   }
   holder.txtListJour.setText(String.format(Locale.FRANCE, "%d",
lesFrais.get(index).getJour()));
   holder.txtListMontant.setText(String.format(Locale.FRANCE, "%.2f",
lesFrais.get(index).getMontant()));
   holder.txtListMotif.setText(lesFrais.get(index).getMotif());
   holder.cmdSuppHf.setTag(index);
   // Suppression du profil enregistré :
   holder.cmdSuppHf.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
         int position = (int)v.getTag();
         // Suppression de la ligne
         lesFrais.remove(position);
         // Suppression de la liste des frais HF
         Serializer.serialize(Global.listFraisMois, inflater.getContext()) ;
         notifyDataSetChanged();
      }
   });
   return convertView;
}
```

Il a fallu attribuer à ce bouton un événement ainsi qu'un index pour pouvoir récupérer tous changement. Et donc faire disparaitre la ligne a l'écran mais également que le tableau Global.listFraisMois se mette à jour. (Partie grisée sur le code précèdent)

Voici 2 photos avant/après suppression :

V – Accès base de données

Pour cette étape du projet l'idée était de pouvoir envoyer tous ce que nous avions saisi en local à la base de données donc distante.

Ayant déjà créé cette base pour mon projet web, je réutiliserai donc la même.

```
MariaDB [gsb_restore]> show tables;
+-----------------------+
| Tables_in_gsb_restore |
+-----------------------+
| comptable             |
| etat                  |
| fichefrais            |
| fraisforfait          |
| lignefraisforfait     |
| lignefraishorsforfait |
| visiteur              |
+-----------------------+
7 rows in set (0.000 sec)

MariaDB [gsb_restore]>
```

Nous pouvons voir que nous avons la table visiteur qui contient tous les visiteurs avec leurs identifiants, coordonnées etc…

Voici de façon plus visuelle dirais-je d'un extrait de cette table sur MySQL :

id	nom	prenom	login	mdp	adresse	cp	ville	dateembauche
a131	Villechalane	Louis	lvillachane	jux7g	8 rue des Charmes	46000	Cahors	2005-12-21
a17	Andre	David	dandre	oppg5	1 rue Petit	46200	Lalbenque	1998-11-23
a55	Bedos	Christian	cbedos	gmhxd	1 rue Peranud	46250	Montcuq	1995-01-12

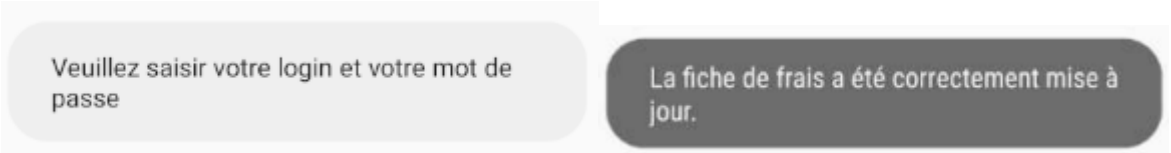
Nous utiliserons donc les champs **login et mdp** pour pouvoir se connecter. Quand je dis connecter j'entends par là, envoyer les informations que l'utilisateur vient de saisir.

Cette table visiteur nous avait été donnée pour nos deux projets. Et rempli pour une partie des informations qu'elle contient.

Un utilisateur peut se connecter à son compte sur l'application Web, mais sur l'application Android, tout le monde peut rentrer des informations en local, mais au moment de l'envoyer sur le serveur les identifiants seront demandés.

Une vérification est donc faite à ce moment et si les identifiants correspondent alors la fiche sera envoyée et la table se mettra à jour, et l'application se remettra à zéro.

Une notification (Toast) est affichée au moment de l'envoie de la fiche, pour indiquer à l'utilisateur si sa fiche a bien été envoyé ou pour lui indiquer qu'une erreur a eu lieu.



Etant donné que l'application Android fonctionne avec la même base de données que l'application WEB, il a fallu faire quelques modifications.

Dans un premier temps les datePicker qu'il a fallu paramétrer, car les frais forfaitisés ne requiert pas le jour du mois mais en revanche pour les frais Hors Forfait eux demandent de renseigner le mois ainsi que le jour, voici donc la fonction affichageDate() dans la classe Global :

```
public static void affichageDate(DatePicker datePicker, boolean afficheJours,
boolean recap) {
    try {
        Field f[] = datePicker.getClass().getDeclaredFields();
        for (Field field : f) {
            int daySpinnerId = Resources.getSystem().getIdentifier("day", "id",
"android");
            datePicker.init(datePicker.getYear(), datePicker.getMonth(),
datePicker.getDayOfMonth(), null);

            if (daySpinnerId != 0) {
                View daySpinner = datePicker.findViewById(daySpinnerId);
                if (!afficheJours && !recap) {
                    // pour les frais forfait, saisie autorisée uniquement sur le
mois en cours :
                    datePicker.setMinDate(System.currentTimeMillis() - 1000);
                    datePicker.setMaxDate(System.currentTimeMillis() + 30000);
                    daySpinner.setVisibility(View.GONE);
                } else if (!afficheJours && recap) {
                    // Pour le récap de frais HF, pas de blocage de dates
                    daySpinner.setVisibility(View.GONE);
                }
            }
        }
    } catch (SecurityException | IllegalArgumentException e) {
        Log.d("ERROR", e.getMessage());
    }
}
```

DatePicker Nous bloquons la date du DatePicker 1 sec avant son affichage et 30sec au maximum après son affichage. Et bien sûr nous n'affichons pas les jours pour les frais forfaitisés.

Pour celui du Hors forfait nous pouvons remonter un an en arrière voici donc une méthode se trouvant dans la classe ConnexionActivity :

```
private void setDateSaisieHF() {
   // instanciation d'une date :
   Calendar dateMinimum = Calendar.getInstance();
   // on remonte 11 mois en arrière
   dateMinimum.add(Calendar.MONTH, -11);
   // ensuite, on sélectionne le 1er jour du mois
   dateMinimum.set(Calendar.DAY_OF_MONTH, 1);
   // on attribue au DatePicker concerné cette date minimum

((DatePicker)findViewById(R.id.datHf)).setMinDate(dateMinimum.getTimeInMillis());

   // instanciation d'une date maxi
   Calendar dateMaximum = Calendar.getInstance();
   // on ajoute 1 mois au mois en cours
   dateMaximum.add(Calendar.MONTH, 1);
   // on sélectionne le 1er jour du mois
   dateMaximum.set(Calendar.DAY_OF_MONTH, 1);
   // on attribue au DatePicker concerné cette date maximum de saisie

((DatePicker)findViewById(R.id.datHf)).setMaxDate(dateMaximum.getTimeInMillis());
}
```

Pour être sûre que l'utilisateur ne rentre pas zéro dans les Hors Forfait j'ai fait en sorte de modifier une fonction déjà existante en vérifiant seulement si le montant rentré était inférieur ou égal à zéro, si c'est le cas il retourne **False** sinon **True.** Nous contrôlons également que la personne a bien saisi un motif de Hors Forfait pour plus de clarté pour le comptable.

```
private boolean enregListe() {
   // Récupération des informations saisies
   Integer annee = ((DatePicker)findViewById(R.id.datHf)).getYear() ;
   Integer mois = ((DatePicker)findViewById(R.id.datHf)).getMonth() + 1 ;
   Integer jour = ((DatePicker)findViewById(R.id.datHf)).getDayOfMonth() ;
   Float montant =
Float.valueOf((((EditText)findViewById(R.id.txtHf)).getText().toString()));
   String motif = ((EditText)findViewById(R.id.txtHfMotif)).getText().toString() ;
   // Enregistrement dans la liste
   Integer key = annee*100+mois ;
   // Vérification que l'utilisateur a bien saisi toutes les informations
   if(montant <= 0 || motif.isEmpty()) {
      return false;
   } else {
      if (!Global.listFraisMois.containsKey(key)) {
         // creation du mois et de l'annee s'ils n'existent pas déjà
         Global.listFraisMois.put(key, new FraisMois(annee, mois)) ;
      }
      Global.listFraisMois.get(key).addFraisHf(montant, motif, jour) ;
      return true;
   }
}
```

Cette méthode sera donc appelée au moment de Valider via une autre fonction cmdAjouter_clic() dans la même classe.

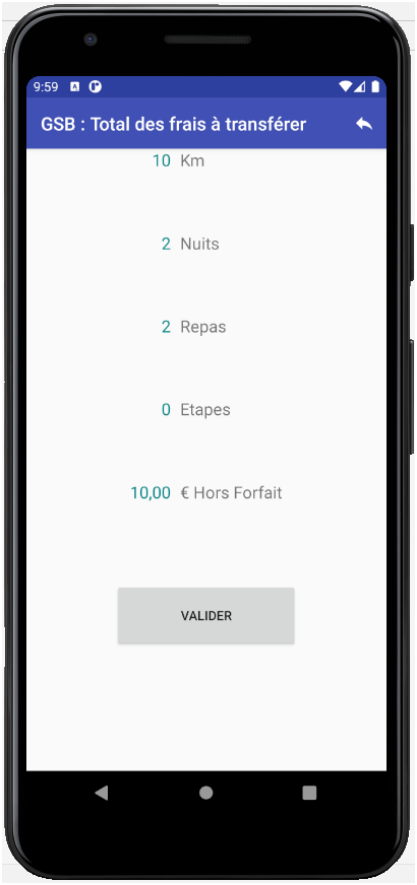
```
private void cmdAjouter_clic() {
    findViewById(R.id.cmdHfAjouter).setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v) {
            if(enregListe()) {
                Serializer.serialize(Global.listFraisMois,  context: HfActivity.this) ;
                retourActivityPrincipale() ;
            } else {
                Toast.makeText( context: HfActivity.this,
                        text: "Veuillez saisir un montant et un libellé",
                        Toast.LENGTH_LONG).show();
            }
        }
    }) ;
}
```

Si enregListe() retourne **True** alors nous passons dans ce **if** et validons notre Hors Forfait ou alors le **else** avec une petite notification nous annonçant de bien vouloir saisir un montant et un motif.

Une fois que les frais forfaitisés et Hors Forfait ont été saisis, nous souhaitons donc les envoyer sur la base de données, nous passerons d'abord par l'icone du bas pour valider ces frais et ensuite nous identifier, voici deux captures d'écran :

Icone permettant d'accéder à la dernière validation avant envoie.

Et voici donc la page récapitulative des frais à valider.



Voici 2 méthodes qui me sont utiles pour l'affichage des informations visibles au-dessus :

```
private void calculForfait() {
    Integer key = annee*100+mois ;
    // Initialisation à 0 des frais
    km = 0 ;
    etape = 0;
    nuitee = 0;
    repas = 0;
    // récupération des frais :
    if (Global.listFraisMois.containsKey(key)) {
        km = Global.listFraisMois.get(key).getKm();
        etape = Global.listFraisMois.get(key).getEtape();
        nuitee = Global.listFraisMois.get(key).getNuitee();
        repas = Global.listFraisMois.get(key).getRepas();
    }
    // Maj des TextView concernés
    ((TextView)findViewById(R.id.txtKmMois)).setText(String.format(Locale.FRANCE,
"%d", km));
    ((TextView)findViewById(R.id.txtEtpMois)).setText(String.format(Locale.FRANCE,
"%d", etape));

((TextView)findViewById(R.id.txtNuitMois)).setText(String.format(Locale.FRANCE,
"%d", nuitee));

((TextView)findViewById(R.id.txtRepasMois)).setText(String.format(Locale.FRANCE,
"%d", repas));
}
```

Et cette fonction pour le calcul des Hors Forfait :

```
private void calculHF() {
    Integer key = annee*100+mois ;
    ArrayList<FraisHf> liste = new ArrayList<>();
    totalFraisHf = Float.valueOf(0);

    for(int m=0; m < 12; m++ ) {
        // calcul de la période concernée
        if((mois - m) >= 1) {
            key = annee*100+(mois-m);
        } else {
            key = (annee-1)*100+(mois + 12 - m);
        }
        // vérification si dans la liste de frais du mois déterminé, il y a des
frais
        if(Global.listFraisMois.containsKey(key)) {
            // si oui on met les frais dans une liste
            liste = Global.listFraisMois.get(key).getLesFraisHf();
            // et on ajoute chaque frais de la liste au montant total
            for(FraisHf frais : liste) {
                totalFraisHf += frais.getMontant();
            }
        }
    }
    // Maj du montant dans le TextView du récap total
    ((TextView)findViewById(R.id.txtHfMois)).setText(String.format(Locale.FRANCE,
"%.2f", totalFraisHf));
}
```

Durant mes deux années de BTS nous avons pu effectuer quelques petits TP en Java, j'ai donc décider de réutiliser 2 classes que nous avions utilisé pour ces TP l'une permettant l'accès au serveur en faisant une requête Asynchrone et l'autre permettant l'accès au serveur distant.

Une partie du code de cette classe :

```
public class AccesHTTP extends AsyncTask<String, Integer, Long> {

    // propriétés
    public static String ret=""; // information retournée par le serveur
    public AsyncResponse delegate=null; // gestion du retour asynchrone
    private String parametres = ""; // paramètres à envoyer en POST au serveur


    public AccesHTTP() {
        super();
    }
```

Il est intéressant de retenir que cette classe **AccesHTTP** a l'extension de AsyncTask qui signifie que la connexion s'établira dans un autre thread que le principal, si ceci n'était pas le cas alors l'application se bloquerait en attendant que le serveur renvoie la réponse. Il en convient que ceci ne serait pas vraiment agréable pour l'utilisateur.

Et voici la deuxième concernant l'accès à distance :

```
public class AccesDistant implements AsyncResponse {

    // Connexion au serveur via ce fichier PHP
    private static final String SERVERADDR = "http://192.168.0.165/gsb_visiteur/android/serveurGSB.php";
//    private static final String SERVERADDR = "http://gsbvisiteur.g-alexis.com/android/serveurGSB.php";
    /**
     * Constructeur
     */
    public AccesDistant() { super(); }

    /**
     * Retour du serveur HTTP
     * @param output
     */
    @Override
```

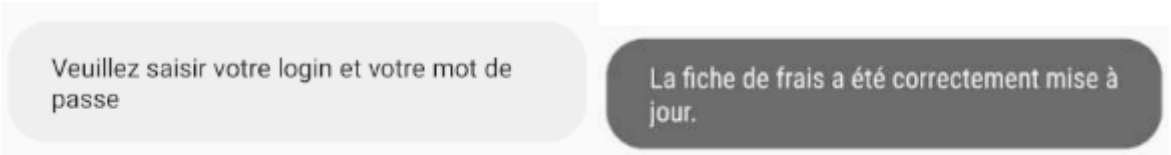
On définit une variable pour l'adresse du serveur, dans un premier temps en local et le lien commenté pour le fichier Php hébergé sur un VPS que je modifierai au moment de la release.

On remarque que l'adresse pointe vers un fichier PHP qui communique avec la base de données car au moment de l'envoie, si les identifiants sont bons alors nous faisons une requête au serveur pour insérer, il nous faut donc envoyer nos informations depuis Java vers Php et pour cela il nous faut donc convertir nos données en format JSON.

```
private JSONArray convertToJSONArray(Hashtable<Integer, FraisMois> listeFrais,
String login, String mdp) {
    List laListe = new ArrayList();
    laListe.add(login);
    laListe.add(mdp);
    Integer annee = Calendar.getInstance().get(Calendar.YEAR);
    // Attention Calendar indexe les mois à partir de 0, donc il faut rajouter 1
pour avoir le bon
    // n° de mois
    Integer mois = Calendar.getInstance().get(Calendar.MONTH) + 1;
    Integer key = annee * 100 + mois;
    if (listeFrais.containsKey(key)) {
        laListe.add(listeFrais.get(key).getKm());
        laListe.add(listeFrais.get(key).getNuitee());
        laListe.add(listeFrais.get(key).getRepas());
        laListe.add(listeFrais.get(key).getEtape());
    }
    for (int m = 0; m < 12; m++) {
        // calcul de la période concernée
        if ((mois - m) >= 1) {
            key = annee * 100 + (mois - m);
        } else {
            key = (annee - 1) * 100 + (mois + 12 - m);
        }
        List laListeHF = new ArrayList();
        // vérification si dans la liste de frais du mois déterminé, il y a des
frais
        if (listeFrais.containsKey(key)) {
            for (FraisHf n : listeFrais.get(key).getLesFraisHf()) {
                laListe.add(key);
                laListe.add(n.getJour());
                laListe.add(n.getMontant());
                laListe.add(n.getMotif());
            }
        }
    }
    return new JSONArray(laListe);
}
```

La plus importante ensuite étant celle lors du clic **Envoyer la fiche,** si les identifiants ont été saisit et que la connexion se fait correctement alors on transtype tout en String et on envoie. Si tout se passe bien durant la phase d'envoie nous avons une notification indiquant à l'utilisateur que la fiche a été mise à jour.

Je remets les 2 notifications en fonction du résultat :



Pour ce qui en est du fichier PHP **serveurGSB.php** celui-ci est hébergé donc sur un serveur et fait parti d'un sous dossier du projet web et plus précisément de la partie visiteur.

Le fait de faire ceci me permet de réutiliser les méthodes créées durant le projet Web.

La classe **PdoGsb** et le fichier où sont toutes mes fonctions **fct.inc.php** sont utilisées pour les 2 applications, donc j'ai simplement mis mon code pour la partie Android en bas de page et indiqué par un commentaire plus imposant que les autres pour pouvoir le visualiser plus facilement.

Peu de chose doive être ajouté pour que tout fonctionne correctement, seulement cette petite méthode au fichier **fct.inc.php** qui permet de retourner une date au format français jj/mm/aaaa

```
function convertitPeriodeEnDate(String $periode, int $jour)
{
    $annee = substr($periode, 0, 4);
    $mois = substr($periode, 4);
    return $jour . '/' . $mois . '/' . $annee;
}
```

Voici un exemple de ce qui circule en Json :

["dandre","oppg5",321,3,11,6,202102,1,277,"Frais courant",202101,18,452,"Frais du 18 janvier"]

Ceci étant la date du mois en cours.

202102,1

Et ceci étant la date pour les frais Hors Forfait.

202101,18

Pour revenir sur le fichier **serveurGSB.php**, on peut voir en début de fichier que nous avons besoins des fichiers fonctions et de la classe Pdo :

```
require '../includes/fct.inc.php';
require '../includes/class.pdogsb.inc.php';
$pdo = PdoGsb::getPdoGsb();
```

Puis nous contrôlons ensuite les données envoyées par java en décodant le Json.

```
// Conversion du JSONarray
$donnees = json_decode($lesdonnees);
// Mois en cours, au format aa/aa/mm
$mois = getMois(date('d/m/Y'));
```

Mais également les identifiants du visiteur.

```
$login = $donnees[0];
$mdp = $donnees[1];
// Vérification des identifiants :
$leVisiteur = $pdo->getInfosVisiteur($login, $mdp);
if ($leVisiteur) {
    print("Authentification_OK%");
    $idVisiteur = $leVisiteur['id'];
```

Si les identifiants sont corrects alors on commence le traitement.

```
if ($leVisiteur) {
    print("Authentification_OK%");
    $idVisiteur = $leVisiteur['id'];

    // Récupération du dernier mois de saisie des frais
    $dernierMoisSaisi = $pdo->dernierMoisSaisi($idVisiteur);

    // Test si les lignes de frais forfait sont déjà créées pour ce mois
    if ($dernierMoisSaisi == $mois) {
        // Récupération des valeurs déjà entrées pour ce visiteur et ce mois
        $fraisForfaitEnCours = $pdo->getLesFraisForfait($idVisiteur, $mois);
        // Affectation dans des variables locales des quantités déjà enregistrées
        $etp = intval($fraisForfaitEnCours[0]['quantite']);
        $km = intval($fraisForfaitEnCours[1]['quantite']);
        $nuit = intval($fraisForfaitEnCours[2]['quantite']);
        $repas = intval($fraisForfaitEnCours[3]['quantite']);


        // Création d'un array, avec l'idFrais en clé et le total (montant
        // déjà enregistré en base + ce qui est envoyé par l'appli Android)
        $fraisAAjouter = array(
            'KM' => $donnees[2] + $km,
            'NUI' => $donnees[3] + $nuit,
            'REP' => $donnees[4] + $repas,
            'ETP' => $donnees[5] + $etp
        );
    } else {
```

```
        } else {
            // Si les lignes de frais forfaits n'existent pas pour la période,
            // il faut donc les créer :
            $pdo->creeNouvellesLignesFrais($idVisiteur, $mois);

            // Création de l'array, avec l'idFrais en clé et le total (montant
            // qui est envoyé par l'appli Android)
            $fraisAAjouter = array(
                'KM' => $donnees[2],
                'NUI' => $donnees[3],
                'REP' => $donnees[4],
                'ETP' => $donnees[5]
            );
        }
        // Mise à jour des lignes de frais forfait dans la base
        $pdo->majFraisForfait($idVisiteur, $mois, $fraisAAjouter);

        // Insertion des lignes de frais HF :
        // test s'il y a des frais hors forfait :
        if(count($donnees) > 6) {
            for ($i = 6; $i <= count($donnees) - 4; $i += 4) {
                $pdo->creeNouveauFraisHorsForfait(
                    $idVisiteur,
                    $mois,
                    $donnees[$i + 3],
                    convertitPeriodeEnDate($donnees[$i], $donnees[$i + 1]),
                    $donnees[$i + 2]
                );
            }
        }
        print("La fiche de frais a été correctement mise à jour.");
    } else {
        print("Echec%" . "Echec de l'authentification");
    }
} catch (PDOException $e) {
    print "Erreur !%" . $e->getTraceAsString();
}
```

N'oublions pas que si tout a bien été envoyé et mis à jour alors nous devons remettre à zéro les frais enregistrés sur le téléphone pour être sûr de ne pas renvoyer plusieurs fois les mêmes informations à la base de données.

Donc nous appelons la fonction **processFinish()** se situant dans la classe AccesDistant pour d'une part remettre à zéro le tableau (listFraisMois) et également de le sérialiser ainsi que la variable context.

Sans oublier le petit Toast pour indiquer à l'utilisateur que le transfert a réussi.

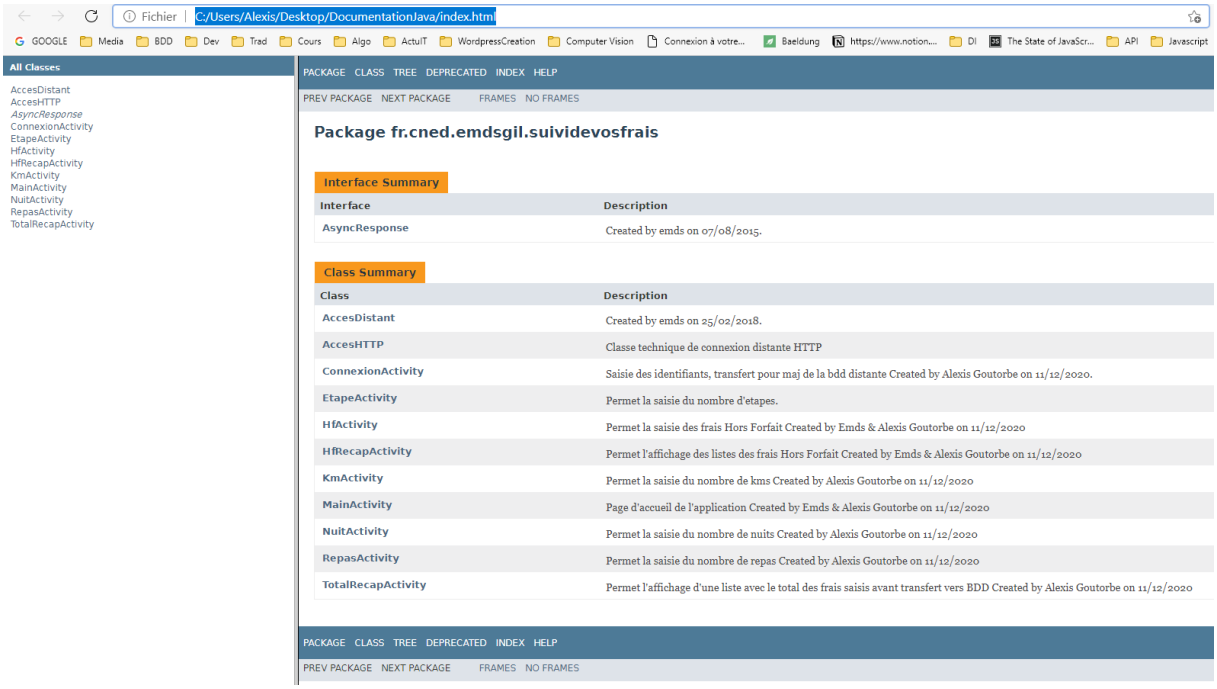
```
public void processFinish(String output) {
    // Vérification du contenu du retour dans la console
    Log.d("serveur", "************" + output);
    // découpage du message reçu
    String[] message = output.split("%");
    // Contrôle si le retour est correct (au moins 2 cases)
    if(message.length>1){
        if(message[0].equals("Echec")){
            Log.d("Echec","***************"+message[1]);
            Toast.makeText(Global.context, message[1], Toast.LENGTH_LONG).show();
        }else if(message[0].equals("Authentification_OK")){
            Log.d("Authentification","***************"+message[1]);
            Toast.makeText(Global.context, message[1], Toast.LENGTH_LONG).show();
            // Puisque le transfert s'est bien passé, remise à 0 du tableau de
frais :
            Global.listFraisMois.clear();
            Serializer.serialize(Global.listFraisMois, Global.context);
        }else if(message[0].equals("Erreur !")){
            Log.d("Erreur !","***************"+message[1]);
            Toast.makeText(
                    Global.context,
                    "Connexion impossible ! Veuillez contacter votre service
informatique",
                    Toast.LENGTH_LONG).show();
        }
    }
}
```

VI - Documentation

Pour générer une documentation Java sous Android Studio rien de plus simple, en seulement 3 clics, vous obtenez une documentation propre telle que vous voyez quand vous naviguez sur le site d'oracle.

https://docs.oracle.com/

Donc en cliquant sur Tool -> Generate JavaDoc -> Ok vous obtenez une doc au format html comme vous pouvez le voir ci-dessous.



Documentation du projet à retrouver ici :

http://www.gsbdocumentation-android.g-alexis.com

VII - Créer un Apk

Lorsque l'on lance notre application sur Android Studio, celui-ci crée un Build et donc génère un fichier Apk, donc à proprement parler Android Studio crée l'Apk pour moi à chaque fois que je lance l'application, je n'ai donc plus qu'à renommer ce fichier et le partager pour que l'on puisse l'utiliser sur un smartphone physique et non plus utiliser l'émulateur d'Android S.

Voici le chemin de l'Apk :