Développement de l'Application Mobile pour les visiteurs de GSB

SOMMAIRE

I PRESENTATION DU PROJET
1) Présentation……. page 3
2) Organisation du code……... page 4

II INTERDICTION DE SAISIE DIRECTE DES QUANTITÉS…….…..… page 5

III ENREGISTREMENT DES AUTRES CATEGORIES DE FRAIS FORFAITISÉS..…. page 6

IV SUPPRESSION DE FRAIS HORS FORFAIT…….……..… page 7

V SYNCHRONISATION AVEC LA BASE DE DONNÉES DISTANTE
1) Blocage des DatePicker……... page 9
2) Contrôle de la saisie des frais hors forfait……... page 11
3) Récapitulatif total des frais…….. page 12
4) L'interface de connexion
4.1) La classe accesHttp()……... page 14
4.2) La classe accesDistant()……...page 15
4.3) Le clic sur le bouton transfert…….. page 17
4.4) La page serveurGSB…… page 19
4.5) Remise à 0 du tableau des frais……... page 22

VI PRODUCTION DE LA DOCUMENTATION…….……...page 23

I PRESENTATION DU PROJET

1) Présentation

Développement :
Il s'agit d'une Application Android développée en Java, pour la mise à jour de leurs frais (forfaits et hors forfaits) par les visiteurs, reliée à une base de données distante.

Cadre :
Il s'agit de la Mission n°3 : Application mobile, proposée dans le cadre de PPE par le réseau Certa, dans le contexte professionnel des laboratoires pharmaceutiques Galaxy Swiss Bourdin.
Les tâches qui nous ont été demandées étaient d'une difficulté progressive.

Support :
Développé avec l'IDE Android Studio, avec un serveur local Wamp et l'IDE NetBeans pour la partie PHP.
Langages = Java et Php.

Détails du contexte : https://portfolio.lencodage.fr/docs/gsb_description_GSB.pdf

Accès aux productions :
- Code source : https://github.com/Steph-bts/GSB-AppliAndroid
- Lien vers le fichier apk : https://github.com/Steph-bts/GSB-AppliAndroid/releases
- Documentation du code : https://gsbdocumentationandroid.lencodage.fr/
- Un compte Visiteur médical : Identifiant = dandre, mdp = oppg5
- Lien vers l'appli web = https://gsb.lencodage.fr

Difficultés rencontrées :
Android évolue tellement vite qu'il est parfois difficile de trouver de l'aide via un moteur de recherche. C'est accentué par le fait que les applications sont de plus en plus souvent développées avec le langage Kotlin.

2) Organisation du code

Une partie du code nous était fournie : la page d'accueil de l'application, ainsi que la page des frais kilométriques, des frais hors forfait, et le récapitulatif des frais hors forfait. Le tout avec la possibilité de sérialiser/ désérialiser.

Contrairement à l'application Web, le code fourni ne respectait pas l'architecture MVC : j'ai donc considéré que cela faisait partie du contexte imposé, et je n'ai pas changé, je me suis contentée d'ajouter les pages tels que c'était fait.

J'ai fait le choix d'héberger la documentation du code générée sur un sous-domaine dédié, gsbdocumentationandroid.lencodage.fr.

II INTERDICTION DE SAISIE DIRECTE DES QUANTITES

Instructions : Modifier le code existant pour interdire la saisie directe des km dans l'activité correspondante : la quantité doit être obtenue uniquement en utilisant les touches + et -.

C'était la première tâche … et la plus facile.

A l'origine, dans l'application telle qu'elle nous était fournie, c'est un widget de type EditText qui servait à la saisie des frais. Il a suffit de le remplacer par un TextView, ainsi l'utilisateur ne peut plus saisir directement les frais.

III ENREGISTREMENT DES AUTRES CATEGORIES DE FRAIS FORFAITISES

Instructions : Créer les autres activités pour la saisie des frais forfaitisés, sur le modèle de la saisie des frais km, en respectant la présentation des interfaces données ci-dessus. Il faudra aussi faire en sorte que les informations soient enregistrées (comme elles le sont déjà pour les km).

Les vues (ou Activity) sous Android Studio, peuvent être générées/manipulées de 2 manières différentes : soit en utilisant la palette graphique, soit directement en format xml.

Afin de coller strictement à la demande, et donc d'obtenir des vues identiques (en adaptant les noms des widgets, des images, des libellés), j'ai donc recopié le fichier xml :

```
<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=".NuitActivity">

    <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">

            <ImageView
                android:id="@+id/imgNuitReturn"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_gravity="end"
                android:layout_weight="1"
                android:contentDescription="@string/icone_frais_nuitee"
                android:src="@drawable/frais_nuitee" />
        </LinearLayout>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="0.5"
            android:gravity="center"
            android:orientation="horizontal">

            <TextView
                android:id="@+id/txtTitleNuit"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:autoSizeTextType="uniform"
                android:gravity="start|center"
                android:lines="1"
                android:paddingRight="10dp"
                android:paddingLeft="10dp"
                android:text="Frais de nuitées"
                android:textColor="@color/colorPrimaryDark" />

        </LinearLayout>


    etc ...
```

J'ai procédé de même pour la page NuitAcivity.java, en modifiant simplement les id, je ne détaillerai pas ces classes ici puisqu'à ce stage, elles sont identiques à la classe KmAcivity fournie.

IV SUPRESSION DE FRAIS HORS FORFAIT

Instructions : Dans l'activité qui affiche le récapitulatif mensuel des frais hors forfait, rajouter un bouton pour la suppression d'une ligne (comme cela est montré dans la capture d'écran ci-dessus). Le bouton devra être ajouté dans le layout_list qui formate l'affichage d'une ligne du listview utilisé pour le récapitulatif. Coder le bouton pour que la ligne soit supprimée et que le frais correspondant soit supprimé dans l'enregistrement. Le codage va se faire dans l'"adapter" de la liste : FraisHfAdapter. Il faut donc dans un premier temps bien s'approprier le code existant pour comprendre le fonctionnement d'un listView.
Pour tester si l'enregistrement se fait correctement, il faut relancer l'émulateur.

Voici l'interface telle qu'elle se présente :

J'ai donc modifié, comme demandé, la page HfAdapter ainsi : en ajoutant le bouton cmdSuppHf dans la classe privée ViewHolder(), puis en lui attribuant un index, et enfin en ajoutant un évènement pour qu'en cas de clic sur ce bouton, la ligne disparaisse à l'affichage, et que le tableau Global.listFraisMois se mette à jour en conséquence :

```
/**
 * structure contenant les éléments d'une ligne
 */
private class ViewHolder {
	TextView txtListJour;
	TextView txtListMontant;
	TextView txtListMotif;
	ImageButton 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);
	// clic sur la croix pour supprimer le profil enregistré :
	holder.cmdSuppHf.setOnClickListener(new View.OnClickListener() {
		@Override
		public void onClick(View v) {
			int position = (int)v.getTag();
			// suppression de la liste à l'affichage
			lesFrais.remove(position);
			// suppression dans la liste des frais HF
			Serializer.serialize(Global.listFraisMois, inflater.getContext()) ;
			notifyDataSetChanged();
		}
	});
	return convertView;
}
```

Ainsi, un clic et la ligne de frais disparaît bien :

V SYNCHRONISATION AVEC LA BASE DE DONNEES DISTANTE

Instructions : Écrire le code derrière le bouton de synchronisation du menu principal, qui va permettre d'insérer dans la base distante tout ce qui a été enregistré en local.
Cette tâche est nettement la plus complexe et la plus importante. Vous avez appris à gérer une base distante avec le TP Android. Donc vous avez les connaissances pour le faire. Cependant il faut penser à la reconnaissance de la personne. Cela suppose que vous devez prévoir une authentification. Vous êtes libre de la méthode à utiliser, le but final étant que la base distante doit être capable de savoir à qui appartiennent les frais reçus pour les enregistrer popaoau bon endroit.

Cette fois-ci en effet, c'était nettement plus complexe.

Tout d'abord, il m'a fallu m'intéresser de près à ce que je voulais que l'utilisateur puisse faire ou pas : par exemple, telle que proposée, l'application permettait de saisir des frais sur n'importe quelle date, passée ou future.
Or, l'application Web ne fonctionne pas ainsi : le visiteur ne peut saisir que son mois en cours pour les frais forfaitisés, et pour les frais hors forfait certes il peut demander le remboursement de frais engagés jusqu'à 1 an en arrière, cependant, ils " passeront " sur la fiche de frais du mois en cours.
J'ai donc modifié le paramétrage des DatePicker, pour bloquer la saisie en conséquence.

De même, il était possible d'enregistrer des frais hors forfait avec un montant de 0,00 €, ou même sans aucun libellé : cela m'a paru, pour le moins, déraisonnable : imaginez la comptable au moment de la validation, qui se retrouve avec les lignes à 0, ou pire, des frais sans libellés !

En ce qui concerne les interfaces, j'ai choisi de proposer, au clic sur le bouton transfert qui se trouve sur la page d'accueil, un tableau qui fait le total de tout ce qui est saisi, et donc sur le point d'être transféré.
Un tel récapitulatif n'existait pas (il y avait seulement un récapitulatif des frais Hors Forfait – mais par mois et non pas au total), je me suis imaginée le pauvre utilisateur contraint d'aller d'entrer dans toutes les activity pour savoir ce qu'il " envoyait " à la base de données au total.

Ensuite, si le Visiteur valide les informations qu'il souhaite envoyer, un écran de connexion lui demandant ses login et mot de passe lui est proposé. Il peut ensuite cliquer sur un bouton pour transférer sa fiche, et un message de retour lui confirme que la fiche a été bien enregistrée (ou un message d'erreur approprié si ce n'est pas le cas).

Il n'était précisé, nulle part, si l'application Android devait supplanter purement et simplement l'application Web, ou si elle venait en complément : j'ai donc décidé que celle-ci venait en complément, et donc lors du transfert des frais, si certaines lignes sont déjà crées (et donc des frais déjà enregistrés), ce qui est transféré depuis l'appli mobile s'ajoute. Ainsi, le Visiteur Médical peut transférer les frais qu'il saisit sur l'appli mobile tous les mois, toutes les semaines … ou même tous les jours s'il le souhaite : la base de données est mise à jour au fur et à mesure, avec les informations transférées.

Enfin, si le transfert s'est bien passé, tous les frais qui avaient été enregistrés sont remis à 0.

1) Blocage des DatePicker

Les DatePicker font déjà l'objet d'un paramétrage particulier, selon s'il s'agit de frais forfait ou hors forfait : en effet, pour les frais forfaitisés, renseigner le jour du mois n'est pas nécessaire, alors que c'est le cas pour les frais non forfaitisés.

Pour bloquer le DatePicker de la saisie de frais forfait, j'ai donc modifié ainsi la méthode statique changeAfficheDate() de la classe Global :

```
public static void changeAfficheDate(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() + 60000);
                    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());
    }
}
```

A partir de là, le DatePicker des frais forfaitisés démarre au minimum 1 seconde avant qu'il s'affiche, et s'arrête 1 minutes maximum après l'affichage. Les jours ne sont pas affichés pour ce type de frais, il était donc inutile de chercher à remonter au 1er du mois jusqu'à la fin du mois.

Pour le DatePicker des frais Hors Forfait, cela demande un paramétrage un petit peu plus fin, puisque selon la description précise du contexte : "*Les frais saisis peuvent remonter jusqu'à un an en arrière (au mois d'août 2021, on peut saisir des frais engagés de septembre 2020 à août 2021)*". Voici, en détail et avec les commentaires, comment j'ai paramétré ces dates de saisies possibles :

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

      // instanciation d'une date maxi
      Calendar dateMaxi = Calendar.getInstance();
      // on ajoute 1 mois au mois en cours
      dateMaxi.add(Calendar.MONTH, 1);
      // on sélectionne le 1er jour du mois
      dateMaxi.set(Calendar.DAY_OF_MONTH, 1);
      // on attribue au DatePicker concerné cette date maximum de saisie
      ((DatePicker)findViewById(R.id.datHf)).setMaxDate(dateMaxi.getTimeInMillis());
}
```

2) Contrôle de la saisie des frais hors forfait

Maintenant, empêchons les utilisateurs de transmettre des lignes de frais hors forfaits avec des montants à 0, ou sans libellés : j'ai … refactorisé d'une certaine manière une méthode qui existait déjà dans la classe HfActivity, la méthode enregListe(). Celle-ci, à l'origine, ne retournait rien, elle se contentait d'enregistrer la nouvelle ligne de frais dans le tableau.

Je l'ai transformée en méthode booléenne : elle contrôle si le montant est > à 0 et si le libellé n'est pas vide, si oui elle enregistre la ligne de frais concernée, et elle retourne true.

```
/**
 * Enregistrement dans la liste du nouveau frais hors forfait
 * Retourne false s'il manque une information (montant ou libellé)
 */
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;
	}
}
```

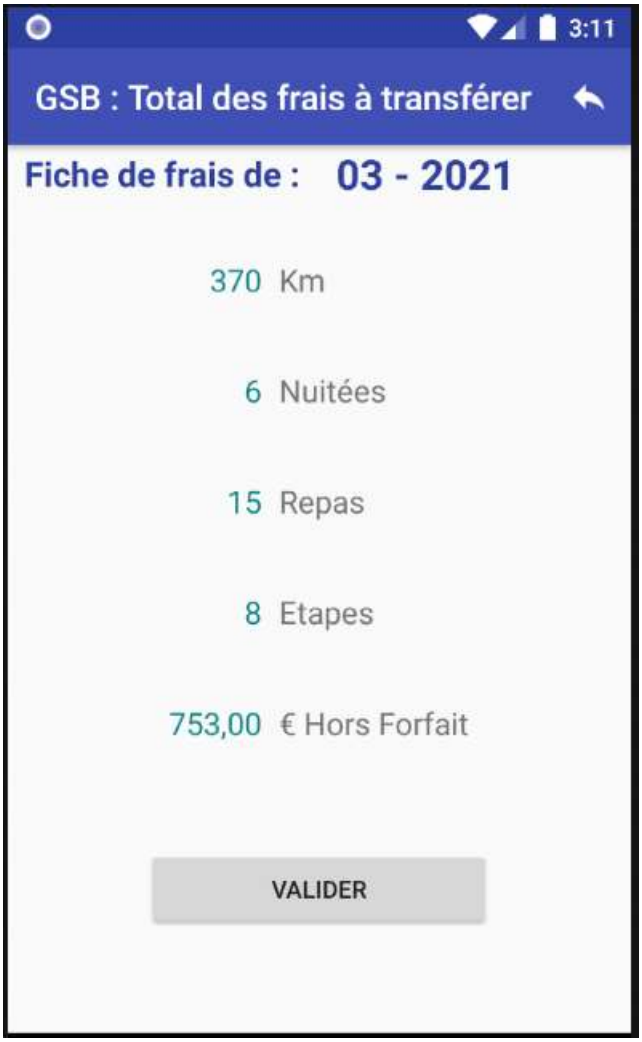
Cette méthode est donc appelée au moment du clic sur le bouton Ajouter :

```
/**
 * Sur le clic du bouton ajouter : enregistrement dans la liste et sérialisation
 */
private void cmdAjouter_clic() {
	findViewById(R.id.cmdHfAjouter).setOnClickListener(new Button.OnClickListener() {
		public void onClick(View v) {
			if(enregListe()) {
				Serializer.serialize(Global.listFraisMois, HfActivity.this) ;
				retourActivityPrincipale() ;
			} else {
				Toast.makeText(HfActivity.this,
				"vous devez saisir un montant et un libellé",
				Toast.LENGTH_LONG).show();
			}

		}
	}) ;
}
```

3) Le récapitulatif total des frais

Je l'ai donc présenté de cette façon : le mois de la fiche de frais concernée (qui est forcément le mois en cours, puisque c'est ainsi qu'a été développée l'application web), les frais forfaitisés, ainsi que le montant total des frais hors forfait saisis :



Je ne m'étendrait pas sur la création de l'interface graphique, qui n'est pas très compliquée.

Concernant la classe liée, TotalRecapActivity(), voici un extrait des méthodes les plus importantes, avec les commentaires appropriés pour une parfaite compréhension du code :

```
/**
 * Affiche le mois en cours dans le TextView approprié, format MM - AAAA
 */
private void afficheMois() {
    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
    mois = Calendar.getInstance().get(Calendar.MONTH) + 1;
    if(mois < 10) {
        ((TextView) findViewById(R.id.txtMoisAnnee)).setText("0" + mois.toString() + " - " +
annee.toString());
    } else {
        ((TextView) findViewById(R.id.txtMoisAnnee)).setText(mois.toString() + " - " +
annee.toString());
    }

}
```

```
/**
 * Récupère les informations de la liste de frais forfait du mois en cours,
 * et les affiche dans les TextView dédiés
 */
private void valoriseProprietesForfait() {
    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));
}
```

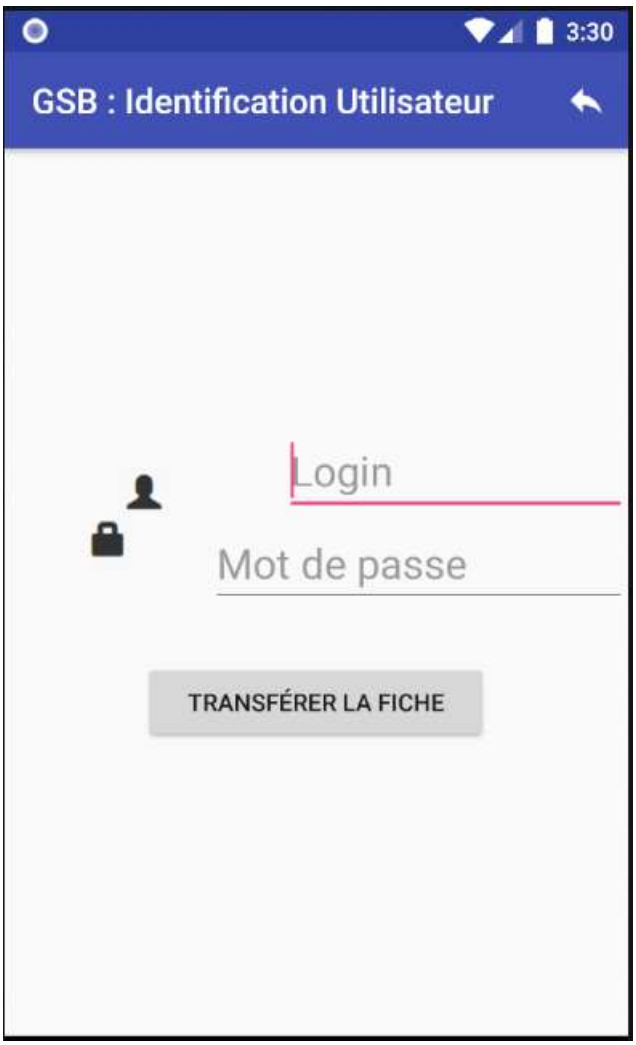
Pour les frais hors forfait, qui peuvent concernés des dates antérieures, c'est un tout petit peu plus complexe :

```
/**
 * Récupère et additionne tous les frais hors forfait enregistrés depuis 1 an, et valorise
 * le TextView dédié
 */
private void valoriseProprietesHorsForfait() {
    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));
}
```

4) L'interface de connexion

Le Visiteur Médical a donc vu le total des frais qu'il a saisi dans l'application Android, si ça ne lui convenait pas il a eu tout le loisir de revenir sur les écrans concernés pour faire les modifications souhaitées, il peut donc maintenant transférer les informations. Pour cela, lui n'a qu'à entrer ses login et mot de passe :



Là encore, je ne m'étendrait pas sur l'interface graphique : il y a 2 EditText et un bouton, rien qui nécessite d'explications détaillées.

Par contre pour le transfert des informations … un certain nombre d'étapes sont nécessaires.

4.1) La classe accesHttp()

Cette classe technique nous avait déjà été fournie lors d'un TP au cours de l'année scolaire.
J'ai choisi de ne pas ré-inventer la roue, et de l'intégrer telle quelle dans mon projet.

Le plus important, c'est qu'il s'agit d'une AsyncTask, c'est à dire que la connexion va s'établir dans un thread distinct du thread d'exécution principal : c'est indispensable, on peut raisonnablement pas bloquer le fonctionnement de l'application en attendant que le serveur distant réponde à la demande, il faut que cette tâche puisse s'éxécuter en tâche de fond, sans gêner l'utilisateur.

En voici un petit extrait :

```
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

	/**
	 * Constructeur : ne fait rien
	 */
	public AccesHTTP() {

		super();
	}

	/**
	 * Construction de la chaîne de paramètres à envoyer en POST au serveur
	 * @param nom
	 * @param valeur
	 */
	public void addParam(String nom, String valeur) {
		try {
			if (parametres.equals("")) {
				// premier paramètre
				parametres = URLEncoder.encode(nom, "UTF-8") + "=" +
URLEncoder.encode(valeur, "UTF-8");
			}else{
				// paramètres suivants (séparés par &)
				parametres += "&" + URLEncoder.encode(nom, "UTF-8") + "=" +
URLEncoder.encode(valeur, "UTF-8");
			}
		} catch (UnsupportedEncodingException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Méthode appelée par la méthode execute
	 * permet d'envoyer au serveur une liste de paramètres en GET
	 * @param urls contient l'adresse du serveur dans la case 0 de urls
	 * @return null
	 */
	@Override
	protected Long doInBackground(String... urls) {

		etc ...
```

4.2) La classe AccesDistant()

Celle-ci a aussi été importée d'un TP fait en cours d'année, mais il a fallu l'adapter un peu, puisque c'est elle qui détermine ce qu'on fait avec la réponse du serveur : voici donc la méthode concernée :

```
/**
 * Retour du serveur HTTP
 * @param output
 */
@Override
public void processFinish(String output) {
    // pour vérification, affiche le 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();
        }
    }
```

Quand on la regarde, ça a l'air tout simple non ? Ce n'est pourtant pas le cas, loin de là, et pour une bête raison : envoyer un Toast (un petit message d'information à l'utilisateur), cela nécessite normalement d'être dans une Activity. Or, là nous sommes dans une méthode qui s'exécute sur un autre thread.

J'avoue avoir eu des difficultés à résoudre ce problème, qui m'a paru insoluble, et j'ai dû demander de l'aide.

Finalement, la solution trouvé est la suivante : la création d'une variable globale de type Context pour pouvoir récupérer le contexte de l'activity concernée. Bonus, c'est aussi grâce à ce context que je peux remettre le tableau des frais à 0 dans l'hypothèse où le transfert s'est déroulé correctement.

```
public static Context context;
```

Le context donc, je le récupère dans la classe à laquelle le Toast est destiné, c'est à dire ma classe ConnexionActivity :

```
/**
 * Retourne l'instance de ConnexionActivity, afin de pouvoir l'utiliser dans une autre classe
 * pour envoyer un Toast
 * @return
 */
public ConnexionActivity getInstance() {
    return this;
}
```

Et puis je le valorise au moment du clic sur le bouton transférer, puisque c'est à ce moment que je souhaite que mon Toast s'affiche : c'est ainsi qu'ensuite je peux tranquillement utiliser le bon contexte pour l'envoi de mon Toast :

```
Toast.makeText(Global.context, message[1], Toast.LENGTH_LONG).show();
```

Parfois, derrière des petits détails, se cachent des concepts complexes.

Une capture de mon Toast, juste pour le plaisir des yeux :

4.3) Le clic sur le bouton transfert – conversion en JSONArray

Pour mettre à jour la base de données distante, il faut donc : que l'appli mobile envoie des informations nécessaires à une page PHP, et ensuite la page PHP, qui sera développée avec les outils nécessaires, manipulera la base de données.

Avant tout, il faut donc convertir les formats en format JSON, un format que PHP connaît et sait manipuler.

```
/**
 * Méthode qui permet de convertir le <hashtable>listFraisMois en JSONArray pour transfert
 * vers la page PHP
 * @param listeFrais : Hashtable<Integer, FraisMois>
 * @param login : String
 * @param mdp : String
 * @return JSONArray : [login, mdp, nbreKm, nbreNuitee, nbreRepas, nbreEtapes, périodeHF, jourHF,
 *                      montantHF, motifHF]
 */
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);
}
```

Tout est dit dans les commentaires du code en fait. A partir de là, il n'y a plus qu'à faire l'envoi : je récupère les login/mot de passe que l'utilisateur a saisi, je les convertit en format String, et s'ils ne sont pas vides, j'envoie les données au serveur. J'ai fait en sorte que le clavier du téléphone, qui apparaît quand on clique sur les EditText, disparaisse lorsque l'on clique sur le bouton " tranférer la fiche " : ainsi, comme vu dans l'écran présenté au-dessus, le Toast d'information apparaît sur fond blanc.

```
/**
 * Click sur le bouton "transférer la fiche" : récupération et transtypage des login et mdp
 * saisis par l'utilisateur, envoie d'un Toast si ces champs sont vides, sinon envoi
 * des données au format JSON
 */
private void cmdTransfert_clic() {
    findViewById(R.id.cmdTransfert).setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v) {
            txtLogin = (EditText)findViewById(R.id.txtLogin);
            login = txtLogin.getText().toString();
            txtMdp = (EditText)findViewById(R.id.txtMdp);
            mdp = txtMdp.getText().toString();
            if(login.isEmpty() || mdp.isEmpty()) {
                Toast.makeText(
                        ConnexionActivity.this,
                        "vous devez saisir votre login et votre mot de passe",
                        Toast.LENGTH_LONG
                ).show();
            } else {
                AccesDistant accesDistant = new AccesDistant();
                Global.context = ConnexionActivity.this;
                accesDistant.envoi("enreg", convertToJSONArray(Global.listFraisMois, login, mdp));
                // pour qu'au clic sur le bouton transférer, le clavier disparaisse pour que
                // l'utilisateur puisse correctement voir le Toast :
                InputMethodManager imm =
(InputMethodManager)getSystemService(Context.INPUT_METHOD_SERVICE);
                imm.hideSoftInputFromWindow(v.getWindowToken(), 0);
            }
        }
    }) ;
}
```

Concrètement, dans le log d'Android Studio, si je lui demande un affichage d'un JSONArray envoyé :

D/JSONArray: **********["lvillachane","jux7g",370,6,15,8,202103,1,297,"Frais courant",202102,23,456,"Frais du 23 février"]

Dans la 1ère case (case n°0) : le login, puis le mdp du visiteur.
Ensuite, les 4 cases suivantes se trouvent les frais forfaitisés, dans cet ordre : km, nuitées, repas, étapes.
Et enfin, s'il y en a, des lignes de frais forfait, chaque ligne utilise 4 cases : la période du frais au format AAAAMM, le jour du mois, le montant, et le motif.

Il est donc temps, maintenant qu'on a les bonnes informations dans le bon format, et qu'on sait identifier comment les trouver, de passer à la page PHP.

4.4) La page ServeurGSB

Concrètement, sur le serveur, cette page est placée dans un sous-dossier que j'ai appelé accesAndroid, le tout dans l'application web visiteurs qui était fournie.

Cela me permet de ré-utiliser toutes les méthodes et les fonctions déjà utilisées pour l'appli web.

Pour faciliter la maintenance, sur les 2 pages utilisées (la classe PdoGsb et la page fct.inc.php), j'ai bien démarqué le code qui serait développé uniquement pour l'appli mobile de cette manière :

```
    }

/*************************************************************************
*           Ajouts pour utilisation appli ANDROID
*************************************************************************/
```

Sauf que …. il n'y a rien en fait.

Du moins, dans la classe PdoGsb, tout ce dont j'avais besoin pour mettre à jour la base de données était déjà développé. Ce qui paraît assez naturel, puisque l'application mobile fait le même travail que l'application web.

Dans les fonctions, j'ai simplement ajouté une petite fonction qui prend une période et une date, et qui transforme le tout en date au format français. C'est pour mes frais hors forfait, puisque dans le JSONArray qui va être traité pour la date à laquelle les frais ont été engagés, c'est la période et ensuite le jour qui sont transmis :
["lvillachane","jux7g",370,6,15,8,202103,1,297,"Frais courant",202102,23,456,"Frais du 23 février"]

Hors, dans la fiche de frais, ces lignes hors forfait sont enregistrées sur la fiche du mois en cours, mais à la 'bonne' date d'engagement des frais. C'est un traitement qui aurait pu se faire sous Android, mais ce n'était pas plus long de le faire à l'arrivée sur la page de connexion au serveur :

```
/*************************************************************************
 *          Ajouts pour utilisation appli ANDROID
 *************************************************************************/

/**
 * Retourne une date au format français à partir de la période concernée
 * et d'un jour donné
 * @param String $periode
 * @param int $jour
 * @return 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;
}
```

Maintenant, il n'y a plus qu'à présenter la page ServeurGsb :

```php
<?php

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


if (isset($_POST["operation"])){
   if(isset($_REQUEST['operation']) == "enreg") {
      try {
         $lesdonnees = $_REQUEST["lesdonnees"];

         // conversion du JSONarray
         $donnees = json_decode($lesdonnees);
         // mois en cours, au format aaaamm
         $mois = getMois(date('d/m/Y'));

         $login = $donnees[0];
         $mdp = $donnees[1];
         // Vérification des login et mdp :
         $leVisiteur = $pdo->getInfosVisiteur($login, $mdp);
         if ($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 {
               // 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();
    }
  }
}
```

Je pense avoir correctement commenté mon code, je ne vois pas grand-chose à ajouter.

4.5) Remise à 0 du tableau de frais

Maintenant, les informations ont été correctement transférées au serveur, il ne reste plus qu'à remettre à 0 les frais qui ont été enregistrés dans l'application mobile.

Pour ça j'utilise ma méthode processFinish dans accesDistant (déjà vue au-dessus), en lui précisant de vider le tableau si le transfert s'est bien passé, et surtout … ne pas oublier de le sérialiser. Ainsi on repart de 0 et on ne risque pas de transmettre plusieurs fois les mêmes frais.

```
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);
```

VI PRODUCTION DE LA DOCUMENTATION

Il ne reste qu'à générer la documentation du code. Android Studio a l'outil nécessaire pour produire une documentation Java, que vous pourrez trouver à l'adresse suivante : : https://gsbdocumentationandroid.lencodage.fr/