Compte rendu

Le code fourni ne contenait que certaines parties de l'interface, la première étape a donc été de rajouter les parties manquantes.

Pour cela il m'a suffi de dupliquer l'activité déjà existante et de la modifier pour correspondre à ce qui était demandé, autant au niveau du texte et des images que des variables et fonctions.

À l'origine, toutes les données sont stockées sous forme d'un Hashtable qui est en suite sérialisé dans un fichier stocké en local. J'ai donc repris la même méthode pour les nouvelles activités.

D'autres modifications ont été nécessaires. Comme celle de forcer la saisie de kilomètres par des boutons – et + qui s'est résumée à désactiver le champ d'entrée des km. Ou bien l'ajout d'un bouton de suppression de frais hors forfaits. Les frais hors forfaits étant sous forme de liste, un simple remove(key) suffit à supprimer une ligne.

La partie la plus difficile de la mission fut la synchronisation avec la base de donnée distante. Pour des raisons de sécurité évidentes, l'application n'interagir pas directement avec la base de donnée, j'ai donc choisi d'instaurer une simple API en php entre l'application et la base de donnée. Le but ici était d'abord de recevoir les données du serveur pour les afficher à l'utilisateur puis de lui permettre, après modifications, de les envoyer pour qu'elle soit insérée dans la base de données.

Le format de transfert des données choisi est le format JSON, qui est facile à envoyer en POST et à traiter coté serveur en php.

Pour cela j'ai utilisé la bibliothèque java Gson qui permet de facilement sérialiser un objet java en json de cette façon :

```
Gson gson = new Gson();
String jsonString = gson.toJson(object);
```

La difficulté a ensuite été d'envoyer ces données au serveur et plusieurs options se sont offerte à moi et parmi celles que j'ai retenues se trouvaient Volley et Retrofit (Apache HTTP est en « legacy » depuis Android 6.0, je l'ai donc exclue.). J'aurais pu aussi choisir d'utiliser la classe HttpURLConnection, mais utiliser une bibliothèque plus orientée API me semblait être plus intéressant. J'ai donc choisi d'utiliser Retrofit.

```java
public interface GsbAPI {

  String BASE_URL = "https://cned.francoislachese.fr/";

  @Headers("Content-Type: Application/json")
  @POST(BASE_URL+"api/auth.php")
  Call<ResponseBody> login(
        @HeaderMap Map<String, String> headers,
        @Query("login") String login,
        @Query("password") String password
  );

  @POST(BASE_URL+"api/write.php")
  Call<ResponseBody> post(
        @Body RequestBody body,
        @Query("login") String login,
        @Query("password") String password
  );

  @POST(BASE_URL+"api/read.php")
  Call<Hashtable<Integer, FraisMois>> read(
        @HeaderMap Map<String, String> headers,
        @Query("login") String login,
        @Query("password") String password
  );

}
```

Retrofit s'utilise à l'aide d'une interface qui permet de définir les requêtes. Il suffit après d'utiliser cette interface lors d'appels asynchrones.

Les données sont transférées en json et à chaque requête le login et mot de passe sont envoyés à nouveau.

L'API n'aura donc pas à gérer de session (façon « stateless »), ce qui rend aussi l'application mobile plus simple et plus sécurisée (si les données sont bien transférées en https bien sûr.).

L'application mobile fonctionnant indépendamment de la base de données cela veut dire aussi que l'API peut être modifiée profondément sans redemander de déploiement de l'application mobile.

Le protocole SOAP a été mis de côté très rapidement car beaucoup trop lourd pour des besoins aussi simples. Avec plus de recul, je pense néanmoins que j'aurais dû concevoir l'application autour de l'utilisation d'une API REST, même si ce que j'ai conçu s'en rapproche. Pour simplifier l'évolution du code serveur et pour que le fonctionnement soit plus normalisé.

La façon dont sont traitées les données dans l'application est discutable et nécessite probablement des modifications pour rester performante. Dans l'état actuel, elle télécharge l'entièreté des données utilisateur et les renvoie aussi dans leur intégralité. Ce n'est pas un fonctionnement optimal. Dans l'idéal, l'application ne devrait par exemple ne télécharger que les données du mois en cours et ne renvoyer que ce qui a été modifié ou ajouté.

L'application mobile fonctionne donc conformément au cahier des charges, néanmoins elle reste perfectible et bénéficierait grandement d'un retour utilisateur quant à sa praticité et à son ergonomie en situation réelle.