Compte Rendu Projet SMB116

Conception et développement pour systèmes mobiles

TORTEVOIS Alexandre

RÉVISIONS				
Version	Nature de la modification	Auteur	Vérificateur	Date
01	Édition Originale	ALT	-	04/02/2020

Objectif :

Réaliser une application de système de « Ticket SAV » communicant avec une API REST (minimaliste) développée pour l'occasion en PHP/MySQL. La communication est bidirectionnelle et au format JSON.
La réponse JSON à une requête est toujours accompagné d'un code de « status » :

- = 0 : pas de réponses
- > 0 : succès
- < 0 : échec

Chaque code a sa signification propre pour être capable, notamment en cas d'échec de connaitre la raison de celui-ci.
Les schémas de l'API et de la base de données sont décrits en fin de document.

Description Technique :

L'application mobile nécessite une authentification, réalisée à l'aide d'un numéro de contrat et d'une clé.
Ce duplet est stocké en local sur le mobile grâce aux *SharedPreferences*.
L'authentification à l'API consiste à vérifier que le numéro de contrat et la clé correspondent avec un utilisateur enregistrer dans la base de données coté serveur.

Quelques identifiants disponibles :

id_customer	contract_id	contract_key	lastname	name
1	SAV0001	test_sav	Sav	Alexandre
2	CL01234	test_client1	Tortevois	Alexandre
3	CL01235	test_client2	Areco	Nico
4	CL01236	test_client3	Client	Frederic
5	CL01237	test_client4	Client	Android
6	CL01238	test_api	Client	Android

L'application développée utilise une *BottomNavigationView* et des *Fragment*.
A l'ouverture de l'application, la *MainActivity* charge les *SharedPreferences* stockées sur le mobile.
Si un contrat et une clé ont été enregistrés et sont valides, l'utilisateur est automatiquement logué sinon le premier écran qui apparait est login et les autres onglets sont verrouillés.

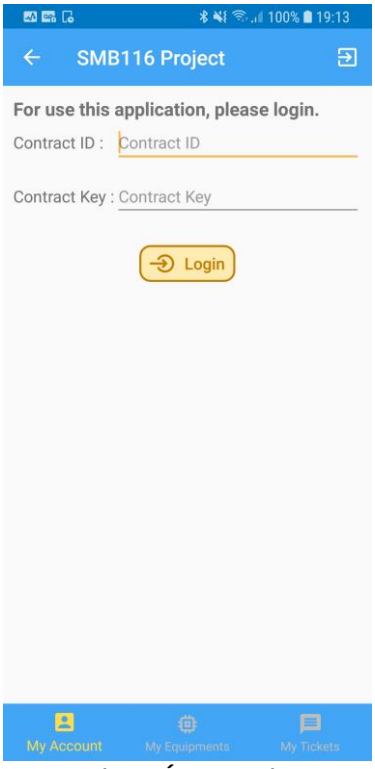


Fig.1 : Écran Login

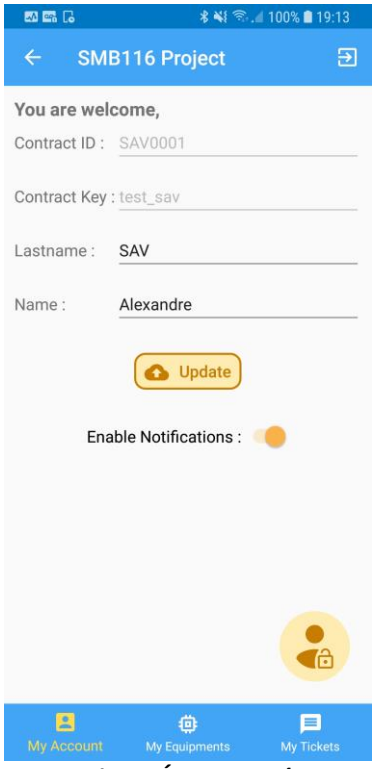


Fig.2 : Écran Logué

Un fois logué, l'utilisateur peut mettre à jour son Nom/Prénom et s'inscrire aux notifications sur les tickets.
Il peut également se déconnecter en cliquant sur l'icône en bas à droite.
L'utilisateur à ensuite la possibilité d'accéder à la liste de ses équipements :

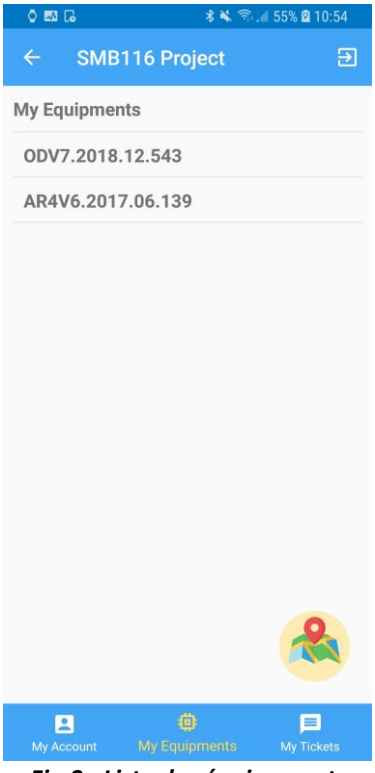


Fig.3 : Liste des équipements

Au clic sur un équipement il peut accéder à la fiche de celui-ci.
Il peut également voir une carte avec la position de ses équipements. Au clic sur un Marker on fait apparaître un popup avec le n° de série et le commentaire associé à l'équipement.

Fig.4 : Carte des équipements

Fig.5 : Fiche d'un équipement

Depuis la fiche d'un équipement, l'utilisateur peut changer le commentaire et mettre à jour la position de l'équipement, soit à partir de la position actuelle du mobile ou en définissant celle si sur la carte via l'icône en bas à droite sur la fiche.

Fig.6 : Carte de positionnement d'un équipement

Sur la carte de positionnement, l'utilisateur peux encore choisir la position actuelle du mobile comme référence.

Le troisième onglet permet d'accéder à la liste des tickets.

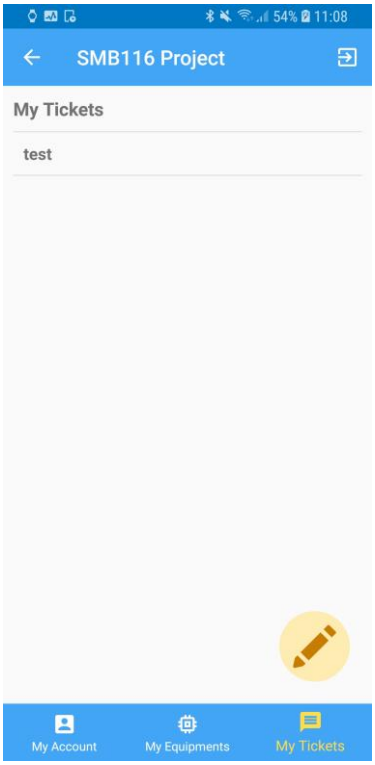


Fig.7 : Onglet tickets

L'utilisateur à la possibilité de créer un nouveau ticket pour déclarer un incident ou de cliquer sur un ticket pour accéder à la liste des messages de celui-ci.

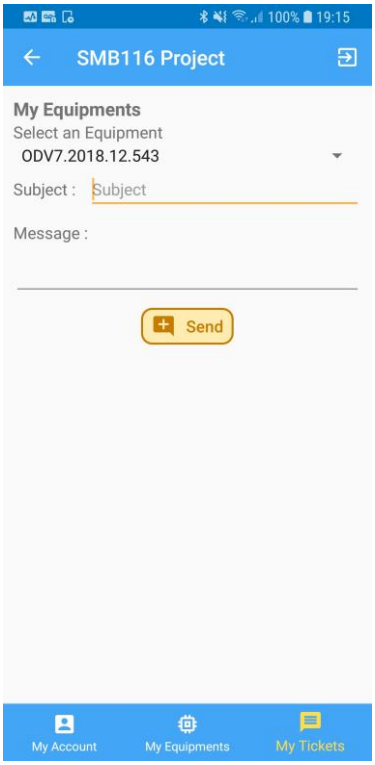


Fig.8 : Ouverture d'un ticket

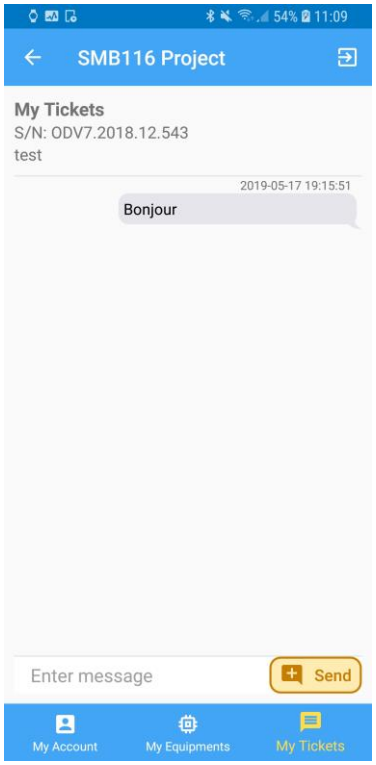


Fig.9 : Liste des messages

Le message est envoyé sur l'API et celle-ci envoie une notification aux abonnés du ticket (via Firebase Cloud Messaging)

Le clic sur la notification ouvre le ticket correspondant.

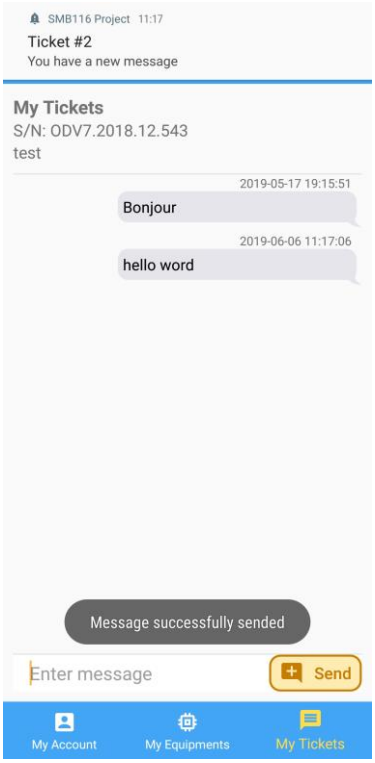


Fig.10 : Notification d'un nouveau message

Pour la partie SAV les fonctionnalités suivantes ont été ajoutées :

Pour les équipements :

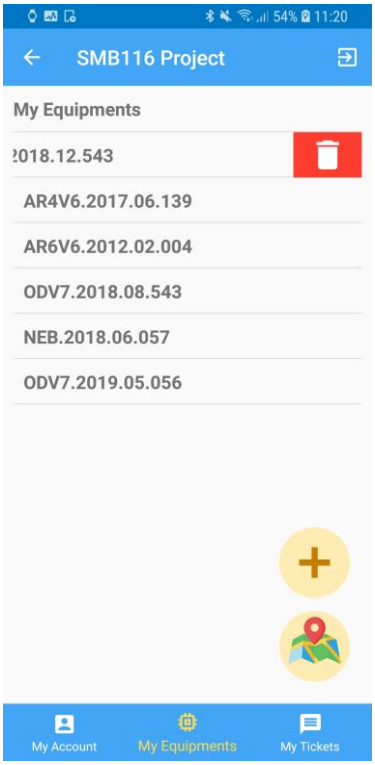


Fig.11 : Bouton Ajouter et Supprimer

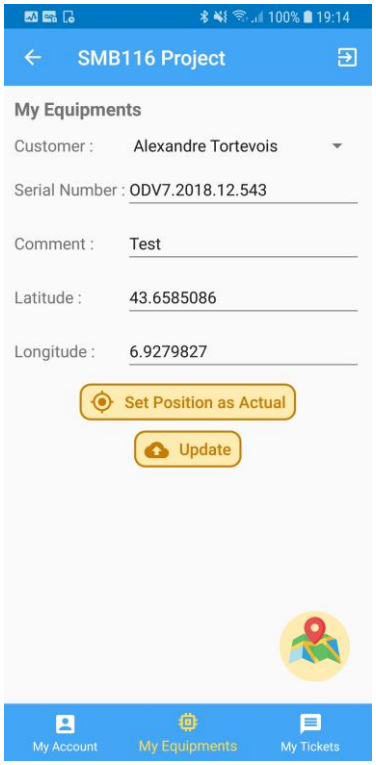


Fig.12 : Modification d'un équipement

Le SAV peut ajouter/modifier/supprimer un équipement à partir de la liste.

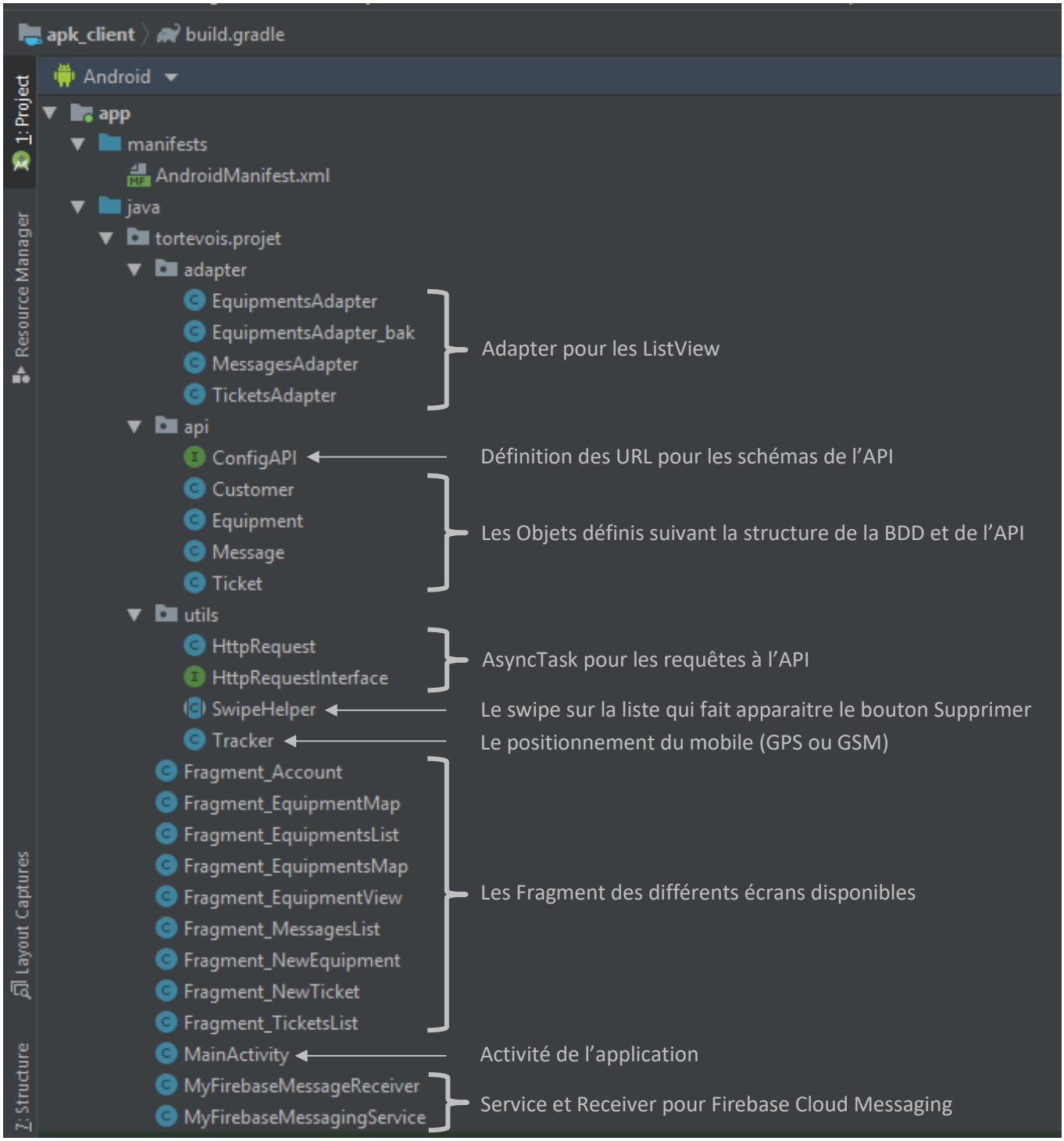
Pour les tickets :

Fig.13 : Bouton Fermer

Fig.14 : Bouton Ouvrir

Le SAV peut modifier le statut du ticket (Ouvert/Fermé) ce qui influe sur la possibilité d'ajouter un nouveau message.

Structure du Projet :



Auto-critique du projet :

Pour l'implémentation d'une *BottomNavigationView* j'ai utilisé ce tutorial OpenClassrooms et celui-ci de SupInfo.
Le passage d'un *Fragment* à l'autre ce fait grâce au *FragmentManager*. Les variables sont définies comme « globales » dans le *MainActivity* et sont donc partagées par les différents *Fragment*.
Le passage d'un *Fragment* à l'autre n'entraîne pas la création d'une nouvelle activité contrairement à ce qu'on a vu dans le cours, je ne sais pas si c'est une bonne méthode et une bonne pratique de développement de ce type d'application mais tous les tutoriaux que j'ai trouvé sur Internet expose tous cette méthode.
Je n'ai vu qu'à la fin du projet, les méthode *setArguments* et *getArguments* qui permettent de passer un *Bundle* d'un *Fragment* à l'autre et qui m'aurais permis de me passer des variables globales.

Les méthodes communes pour les différents *Fragments* sont également définies dans la *MainActivity* pour éviter la redondance de code. Là encore, une réflexion à postériori me fait dire qu'il aurait été peut-être mieux de mettre ces méthodes dans une classe « Objets » de l'application qui auraient regroupée les variables qui ont été définie comme globales. Ainsi les objets *appCustomer*, *appEquipments*, *appTickets*, auraient pu être encapsulées dans une classe « Objets » qu'on aurait pu mettre dans un *Bundle* pour passer de *Fragment* en *Fragment*, et tous les Accesseurs, Getter et Setter aurait été défini dans les méthodes de cette classe.

Pour toutes les requêtes à l'API et la transformation JSON, j'ai trouvé (trop tard) une librairie qui m'aurait simplifié (ou pas) le code : https://square.github.io/retrofit/
Néanmoins l'utilisation d'une *AsyncTask* et de la méthode *doHttpRequest* (dans *MainActivity*) ne m'a pas paru complexe à mettre en œuvre et est un bon exemple d'application du cours.

Conclusion :

C'était un projet ambitieux et intéressant à mettre en place dans le cadre de cette UE, même si la partie API qui sort un peu du cadre de l'UE. Mais dans le monde professionnel, nous devons avoir cette capacité à mettre en œuvre différente technologie ensemble.
Il y a encore pleins de choses qui sont perfectibles et de TODO dans le code, mais le manque de temps en cette fin d'année scolaire a eu raison de mes ambitions.
J'aimerai avoir, si possible, un retour sur les bonnes méthodes à appliquer dans la structure d'un tel projet.

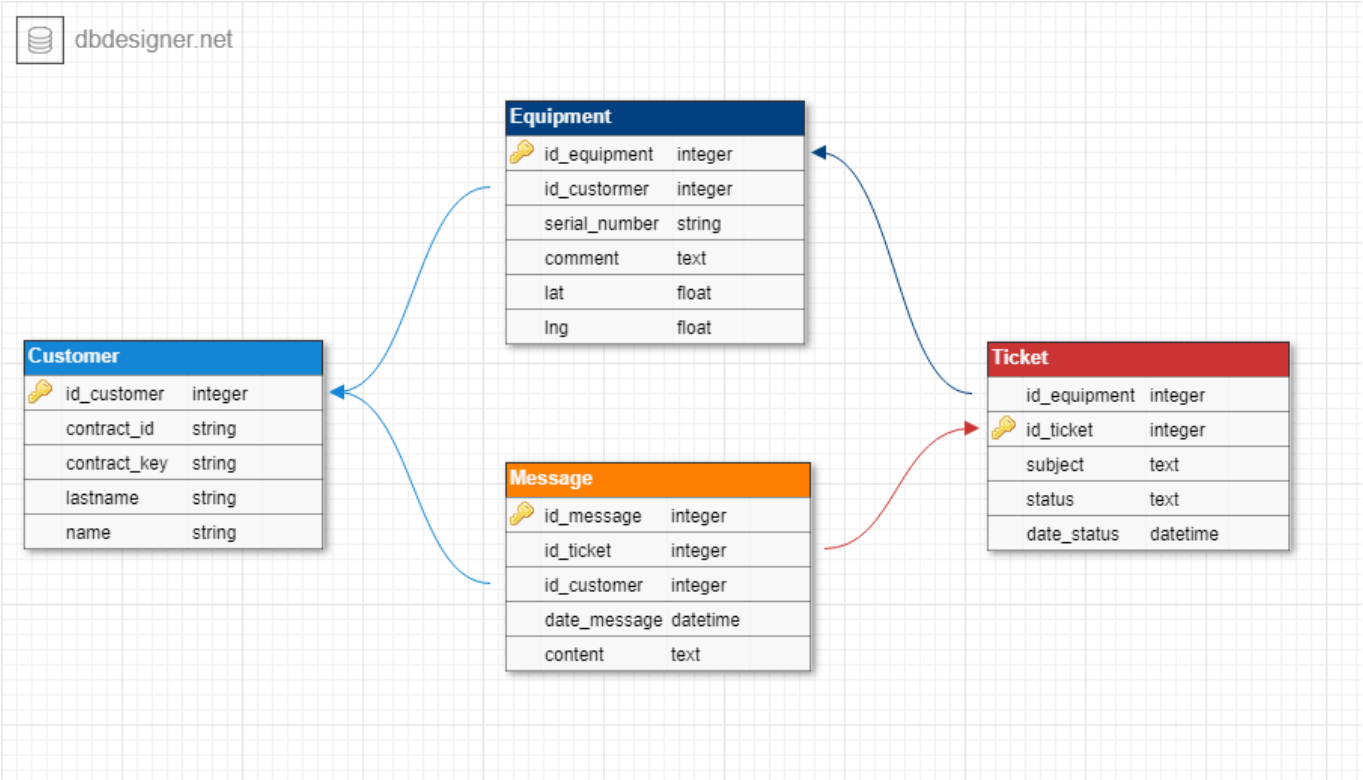
Bibliographie :
Les liens ont été mis en commentaires dans les sources.

Quelques détails sur l'API

Adresse : http://alexandre.tortevois.fr/api/

Le script PHP se contente de modifier la base de données MySQL. On se contente de vérifier que les champs ne sont pas vides et on se protège des injections XSS grâce aux fonctions *htmlspecialchars* et *strip_tags*.
Les requêtes sont formattées avec la classe PDO pour rester dans l'esprit du traitement DAO proposé avec Java.
Le script PHP exécute les requêtes SQL et retourne les résultats en JSON.
Les schémas de l'API sont décrits ci-après. Tous ceux qui sont présentés dans ce présent document, n'ont pas été implémentés dans l'application Android.

La base de données MySQL utilisée pour les besoins de l'application :



.htaccess	URL Rewriting et filtrage des requêtes
index.php	
error.php	Définition des codes erreurs
req_get.php	Traitement des requêtes GET
req_post.php	Traitement des requêtes POST
req_put.php	Traitement des requêtes PUT
req_delete.php	Traitement des requêtes DELETE
fcm_push.php	Envoie des notifications sur Firebase Cloud Messaging
tortevois.sql	Création de la base de données

Schémas de l'API :

GET **~/auth/*$contract_id*/*$contract_key***

Retourne l'utilisateur, si la clé d'authentification est correcte, dans un tableau.

Requête :

```
{
}
```

Réponse :

```
{
   "status":"",
   "msg": [
      {
         "id_customer":"",
         "contract_id":"",
         "contract_key":"",
         "lastname":"",
         "name":""
      }
   ]
}
```

POST **~/customers**

Ajoute un utilisateur.

Requête :

```
{
   "contract_id":"",
   "contract_key":"",
   "lastname":"",
   "name":""
}
```

Réponse :

```
{
   "status":"",
   "msg": {
      "id_customer":""
   }
}
```

GET **~/customers**

Retourne la liste de tous les utilisateurs dans un tableau.

Requête :

```
{
}
```

Réponse :

```
{
   "status":"",
   "msg": [
      {
         "id_customer":"",
         "contract_id":"",
         "contract_key":"",
         "lastname":"",
         "name":""
      },
      ...
   ]
}
```

GET **~/customers/*$id*/equipements**

Retourne la liste des équipements d'un utilisateur dans un tableau.

Requête :

```
{
}
```

Réponse :

```
{
   "status":"",
   "msg": [
      {
         "id_equipment":"",
         "id_customer":"",
         "serial_number":"",
         "comment":"",
         "lat":"",
         "lng":""
      },
      ...
   ]
}
```

GET **~/customers/*$id*/tickets**

Retourne la liste des tickets d'un utilisateur dans un tableau.

Requête :

```
{
}
```

Réponse :

```
{
   "status":"",
   "msg": [
      {
         "id_ticket":"",
         "id_equipement":"",
         "title":"",
         "status":"",
         "date_satus":"",
      },
      ...
   ]
}
```

PUT **~/customers/*$id***

Mettre à jour un utilisateur.

Requête :

```
{
   "id_customer":"",
   "lastname":"",
   "name":"",
   "key":""
}
```

Réponse :

```
{
   "status":""
   "msg": {
      "id_customer ":""
   }
}
```

DELETE **~/customers/*$id***

Supprimer un utilisateur.

Requête :
```
{
}
```

Réponse :
```
{
   "status":""
   "msg": {
      "id_customer":""
   }
}
```

POST **~/equipments**

Ajoute un équipement.

Requête :
```
{
   "id_customer":"",
   "serial_number":"",
   "comment":"",
   "lat":"",
   "lng":""
}
```

Réponse :
```
{
   "status":"",
   "msg": {
      "id_equipment":""
   }
}
```

GET **~/equipments**

Retourne la liste de tous les équipements dans un tableau.

Requête :
```
{
}
```

Réponse :
```
{
   "status":"",
   "msg": [
      {
         "id_equipment":"",
         "id_customer":"",
         "serial_number":"",
         "comment":"",
         "lat":"",
         "lng":""
      },
      ...
   ]
}
```

GET **~/equipments/*$id***

Retourne l'équipement dans un tableau.

Requête :
```
{
}
```

Réponse :
```
{
   "status":"",
   "msg": [
      {
         "id_equipment":"",
         "id_customer":"",
         "serial_number":"",
         "comment":"",
         "lat":"",
         "lng":""
      }
   ]
}
```

PUT **~/equipments/*$id***

Mettre à jour un équipement.

Requête :
```
{
   "id_customer":"",
   "serial_number":"",
   "comment":"",
   "lat":"",
   "lon":""
}
```

Réponse :
```
{
   "status":"",
   "msg": {
      "id_equipment":""
   }
}
```

DELETE **~/equipments/*$id***

Supprimer un équipement.

Requête :
```
{
}
```

Réponse :
```
{
   "status":"",
   "msg": {
      "id_equipment":""
   }
}
```

POST **~/tickets**

Ajoute un ticket.

Requête :

```
{
   "id_equipment":"",
   "title":"",
   "status":"",
   "date_satus":""
}
```

Réponse :

```
{
   "status":"",
   "msg": {
      "id_ticket":""
   }
}
```

GET **~/tickets**

Retourne la liste de tous les tickets dans un tableau.

Requête :

```
{
}
```

Réponse :

```
{
   "status":"",
   "msg": [
      {
         "id_ticket":"",
         "id_equipement":"",
         "title":"",
         "status":"",
         "date_satus":"",
      },
      ...
   ]
}
```

GET **~/tickets/$id/equipments**

Retourne la liste des tickets d'un équipement dans un tableau.

Requête :

```
{
   "id_equipment ":""
}
```

Réponse :

```
{
   "status":"",
   "msg": [
      {
         "id_ticket":"",
         "id_equipement":"",
         "title":"",
         "status":"",
         "date_satus":"",
      },
      ...
   ]
}
```

GET **~/tickets/*$id*/messages**

Retourne la liste de tous les messages d'un ticket dans un tableau.

Requête :
```
{
}
```

Réponse :
```
{
   "status":"",
   "msg": [
      {
         "id_ticket":"",
         "id_customer":"",
         "date_message":"",
         "content":""
      },
      ...
   ]
}
```

PUT **~/tickets/*$id***

Mettre à jour un ticket.

Requête :
```
{
   "id_equipment":"",
   "title":"",
   "status":"",
   "date_satus":""
}
```

Réponse :
```
{
   "status":""
   "msg": {
      "id_ticket":""
   }
}
```

DELETE **~/tickets/*$id***

Supprimer le ticket.

Requête :
```
{
}
```

Réponse :
```
{
   "status":""
   "msg": {
      "id_ticket":""
   }
}
```

POST **~/messages**

Ajoute un message.

Requête :

```
{
   "id_ticket":"",
   "id_customer":"",
   "date_message":"",
   "content":""
}
```

Réponse :

```
{
   "status":"",
   "msg": {
      "id_message":""
   }
}
```

GET **~/messages/*$id***

Retourne le message dans un tableau.

Requête :

```
{
}
```

Réponse :

```
{
   "status":"",
   "messages": [
      {
         "id_ticket":"",
         "id_customer":"",
         "date_message":"",
         "content":""
      }
}
```

PUT **~/messages/*$id***

Mettre à jour le message.

Requête :

```
{
   "id_ticket":"",
   "id_customer":"",
   "date_message":"",
   "content":""
}
```

Réponse :

```
{
   "status":"",
   "msg": {
      "id_message":""
   }
}
```

DELETE **~/messages/*$id***

Supprimer le message.

Requête :

```
{
}
```

Réponse :

```
{
   "status":""
   "msg": {
      "id_message":""
   }
}
```