```
title : Specifications Projet Android
author : Sylvestre Massalaz
```

...

Résumé

Application permettant de signaler des Alertes par rapport à une position géographique et de tenir informé les gens de l'évolution de ces alerte.

Fonctionnalités

Utilisateurs

- S'enregistrer sur l'application
- Se connecter

Evènements

- Obtenir la liste des alertes autours d'une position
- Publier un évènement
- Ajouter soi même un évènement
- Mettre à jour un évènement que l'on a ajouté
- Recevoir des notifications par rapport aux alerte qui ont été mis à jour.

Points techniques

- Notifications concernant les alertes
- Recherche géographique dans les alertes

Modèle de données

Utilisateur

Un utilsateur est composé des informations suivante :

- un code pin
- un numéro de téléphone
- un email : sert de login
- un nom

- un prénom

Evènement

- un auteur
- une position géographique
- un timestamp de création
- Une liste de votes
  - Un utilisateur
  - La valeur du vote (booléen) : validation de l'évènement

Back-End

Infrastructure serveur

Afin de permettre d'accéder aux informations et gérer les opérations de mise à jour et ajout, une api REST sera mise à disposition des utilisateurs. Cette dernière permettra d'effectuer l'ensemble des opérations.

L'infrastructure sera composée d'un serveur (J2EE/ Node.JS / Play/ Python) accompagné d'une base (MongoDB?) permettant de faire des recherches géographiques nativemement, afin de contourner le problème (qui n'est pas notre problématique principale).

Proposition : Pour aller rapidement : partir sur un django (Python) + Spatialite (intégré à Django)

Notifications

Afin de permettre de recevoir des notifications lors de l'update d'un évènement où d'un ajout, il est nécessaire de disposer d'un système fonctionnel de notification.

Google propose gratuitement le `Google Cloud Messaging` qui semble être notre potentielle solution sans trop de code.

Il faut que vous consultiez la documentation GCM afin d'avoir toutes les informations vous permettant de connecter le client et traiter les méssages reçus.

Utilisation au niveau client

Au niveau du client, l'idée est de s'abonner à des topics (voir la documentation de Google) correspondants aux évènements à proximités. A chaque mise à jour des événements autours de nous, il faudra s'inscrire aux nouvelles alertes et se désincrire aux alertes en dehors de notre périmètre. Il faudra également s'assurer d'être inscrit au canal de notre utilisateur actuel afin d'être tenu au courant des informations liées à l'apparition de nouvelles alertes près de notre position.

Informations importantes

Sender ID : 191695741360

Server Key : AIzaSyByMfvnFv-akvkt89vudqvoH_Mm1sWXOzo

Utilisation des topics

L'application fera principalement usage du système de topic qui me semble plus simple car indifférent de l'id obtenu temporairement par le client (si jamais vous trouvez une méthode plus simple, n'hésitez pas), utilsier un topic permet de s'abstenir de maintenir l'id client. Les noms des topics sont gérés de la manière suivante :

- user-[USER_iD] : Topic dédié aux notifications d'un certain user
- alert-[ALERT_ID]: Topic dédié aux notifications d'une certaine alerte (notification de fin)

Je vous laisse étudier la documentation GCM pour savoir comment s'incrire et traiter les messages reçus. Les données seront transmises sous forme de json dans le champs data des messages

Messages reçus sur le topic d'un utilisateur

Via le topic d'un utilisateur, vous pourrez recevoir un type de message : l'apparition d'une alerte.

```
{
    ...
    "data": {
        "id": "ALERT_ID",
        "lat": "latitude",
        "long": "longitude"
    }
}
```

Attention, cependant. Il convient de vérifier ce que le **/alerts/getlist** vous renvois, car les notifications sont générées à partir de votre derniere position connue lors de l'ajout d'une nouvelle alerte. Le **/alerts/getlist** met à jour cette position à partir de la latitude et longitude que vous lui envoyez, ainsi, il est possible qu'il y ait de nouvelles artles dans ce que vous recevrez.

Basiquement, ces notifications vous permetent d'être tenu au courant des alertes apparaissant autours de votre dernière position avant que vous actualisiez manuelle votre liste auprès du serveur.

Messages reçus sur le topic d'une alerte

Via le topic d'une alerte, vous pourrez recevoir le message suivant vous signalant la fin de cette dernière

```
{
    ...
```

```
    "data" : {
        "isActive": false
    }
}
```

API REST

L'API mise à disposition des utilisateurs utilise le format json pour formater les arguments et résultats

Vous trouverez le serveur déployé à l'adresse suivante : `http://vps253024.ovh.net/`

Cas généraux

Succès

Retour d'un code HTTP 200 (OK)

Erreurs

Une requête peut rencontrer plusieurs types d'érreurs :

- Erreur liée à l'état de la connexion (session invalide, pas de session)
- Erreur liée aux arguments passés à la requête
- Problème de droit
- Erreur système

Une réponse contenant une erreur contient un champs error. la détection d'un champ error doit faire ignorer le reste de la réponse et annuler l'action en cours au niveau de l'application.

Les codes d'erreur HTTP peuvent être les suivants :

- UNAUTHORIZED (401) : Erreur lors de la connexion (mauvais couple login/pass) ou mauvaise session (ou pas de dession) donnée en argument des appels
- BAD REQUEST (400) : Erreur de syntaxe pour la requête : url invalide ou arguments malformés, compte uasis le cas où l'utilisateur ne peux pas faire l'action demandée (hors problème de droits)
- INTERNAL SERVER ERROR (500) : Erreur lors de l'exécution du code lié à la requête

Les codes d'erreur émis par l'application peuvent être les suivants :

- **APP_MALFORMED_JSON** (1) : Le json envoyé n'a pas pu être parsé où aucun json n'a été envoyé
- **APP_MISSING_DATA** (2): Le json envoyé ne contient pas tous les champs nécessaires pour la requête
- **APP_INVALID_DATA** (3) : Le json envoyé était complet mais son contenu ne permettait pas l'exécution correct de la requête
- **APP_LOGIN_FAILED** (4): Les identifiants fournis n'ont pas permi de se connecter
- **APP_SESSION_EXPIRED** (5): La session a expiré, il faut se reconnecter pour continuer à travailler

- **APP_SESSION_NOT_FOUND** (6): La sessinon donné en paramètre n'existe pas
- **APP_ALERT_NOT_FOUND** (7): L'alerte donnée en paramètre n'existe pas
- **APP_ALERT_ALREADY_VOTED** (8): L'utilisateur a déjà voté pour cette alerte

En plus du code d'erreur, plusieurs informations complémentaires sont fournies, permettant l'identification du problème :

- message : juste une string explicitant en anglais le problème
- data : un objet Json contenant des informations par rapport au problème

```
{
    "error" : {
        "code" : 42,
        "message" : "Message d'erreur"
        "data": {
            "field" : "Structured data"
        }
    }
}
```

/users/register

Cette requête permet de s'inscrire en tant qu'utilisateur de l'application.

Type : POST

Payload

```
{
    {
        "phone":"0102030405",
        "mail":"email@provider.dom"
        "first_name":"Pinkamena",
        "last_name":"Pie",
        "pin":"1234"
    }
}
```

Réponse

Succès

```
{
    "success" : true
}
```

Erreur

Si les données fournies sont de mauvaise qualité et donc ne passent pas la validation (code: 3), (et

ce pour unicité ou autre), le champs data contient un dictinonaire des champs ayant planté, avec le même nom que durant le payload.

```
{
    "phone": [
        "Enter a valid value"
    ]
}
```

Pour savoir quel champs à merdé, il suffit de vérifier l'existence d'une valeur pour ce champ.

/users/login

Connexion et établissement d'une session avec l'application. L'UUID de session doit être soigneusement conservé car il est nécessaire à l'utilisation des requêtes nécessitant une authentification.

Payload

```
{
    "login" : "LOGIN",
    "pass" : "PASSWORD"
}
```

Réponse

```
{
    "session" : "SESSION_UUID",
    "user" : "USER_ID",
    "expire" : TIMESTAMP
}
```

/users/check/mail

Méthode permettant de vérifier la disponibilité d'un email avant de créer un utilisateur

Payload

```
{
    "mail" : "email@to.test"
}
```

Réponse

Si l'argument a bien été fourni, la requête devrait retourner un simple json contenant un booléen indiquant le succès du check. True si le mail est libre, false if not

```
{
    "success": true
}
```

/users/check/phone

Méthode permettant de vérifier la disponibilité d'un téléphone avant de créer un utilisateur

Payload

```
{
    "phone" : "0000000000"
}
```

Réponse

Si l'argument a bien été fourni, la requête devrait retourner un simple json contenant un booléen indiquant le succès du check. True si le numéro est libre, false if not

```
{
    "success": true
}
```

/alerts/getlist

Méthode principale permettant de récupérer des alerte ouverts par rapport à une position. Il retourne une liste d'identifiants correspondant à des évènemlents dont les informations pourrons être récupérés plus tard via d'autres requêtes. cela permet notamment d'utiliser un sytème de cache pour ne pas récupérer les information de tous les alerte à chaque fois.

□□

Payload

```
{
    "session" : "SESSION_UUID",
    "lat" : LATITUDE,
    "long" : LONGITUDE
    "radius": RADIUS_IN_METERS
}
```

Réponse

```
{
    [
        {
            "id" : "alert_ID"
```

```
        },
        {
            "id" : "alert_ID"
        },
        ...
    ]
}
```

/alerts/get

Cette commande permet de récupérer les informations de certains events à partir de leur id.

Payload

```
{
    "session" : "SESSION_UUID",
    "alerts" : [
        {
            "id" : "alert_ID1"
        },
        {
            "id" : "alert_ID2"
        },
        ...
    ]

}
```

Réponse

```
{
    "id1": {
        JSON_REPR_1
    }
    ...
}
```

Représentation json d'une alerte

```
{
    "id": ALERT_ID,
    "name": "name",
    "author": "FirstName LastName",
    "long": LONGITUDE,
    "lat": LATITUDE,
    "score": VOTE_SCORE,
    "distance": DISTANCE,
    "hasVoted": TRUE/FALSE
}
```

/alerts/add

Cette méthode permet de publier un nouvel évènement et d'en récupérer l'id pour pouvoir le modifier rapidement par la suite

Payload

```
{
    "session" : "SESSION_UUID",
    "lat" : LATITUDE,
    "long" : LONGITUDE,
    "name": NAME
}
```

Réponse

```
{
    "id" : "alert_ID"
}
```

/alerts/close

Fermer un évènement

Payload

```
{
    "session" : "SESSION_ID",
    "alert" : "alert_ID"
}
```

Réponse

```
{
    "success" : true
}
```

/alerts/validate

Valider (ou invalider) une alerte sur le serveur.

Validate

```
{
    "session" : "SESSION_ID",
    "alert" : "alert_ID"
    "validate" : true/false
}
```

Réponse

```
{
    [
        {
            "success" : true
        }
    ]
}
```