TP – Séminaire technologies web

1. Notation

Le projet est noté sur 30 points (ramenés à 20).

La notation se divise en 3 parties : les routes de base de l'API REST, des routes « avancées » et une partie visualisation. Chaque partie vaut 10 points.

Pour les routes, les points de chaque route ne sont **attribués que si la route passe le test unitaire lié**.
Vous aurez les tests unitaires, à vous de bien vérifier que vos routes les valident !

a. API REST de base

REST	Route	Résultat attendu	Points
GET	/users	Renvoie tous les users + code 200	0.25
GET	/users/id	Renvoie le user avec l'id + code 200	0.25
POST	/users	Créé un user (name, email, alliance_id). Renvoie 200 + nouvel user (récupéré en base !)	1
DELETE	/users/id	Supprime le user avec l'id. Renvoie 200	0.5
PUT	/users/id	Modifie le user avec l'id (name, email, alliance_id) Renvoie 200 + user modifié (récupéré en base !) Rappel : On doit fournir une entité complète dans le corps d'une requête PUT	1
GET	/alliances	Renvoie tous les alliances + code 200	0.25
GET	/alliances/id	Renvoie l'alliance avec l'id + code 200	0.25
POST	/alliances	Créé une alliance (name). Renvoie 200 + nouvelle alliance (récupérée en base !)	1
DELETE	/alliances/id	Supprime l'alliance avec l'id. Renvoie 200	0.5
PUT	/alliances/id	Modifie l'alliance avec l'id (name) Renvoie 200 + alliance modifiée (récupérée en base !)	1
GET	/characters	Renvoie tous les characters + code 200	0.25
GET	/characters/id	Renvoie le character avec l'id + code 200	0.25
POST	/characters	Créé un character (name, class, user_id, point). Renvoie 200 + nouvel character (récupéré en base !)	1
DELETE	/characters /id	Supprime le character avec l'id. Renvoie 200	0.5
PUT	/characters /id	Modifie le character avec l'id (name, class, user_id, point) Renvoie 200 + character modifié (récupéré en base !)	1

Total		Point bonus (sur 30) si toutes les routes de base fonctionnent.	9 + 1 (bonus)

b. Routes « avancées »

REST	Route	Résultat attendu	Points
GET	/alliances/id/users	Renvoie tous les users de l'alliance avec l'id + code 200	1
GET	/alliances/id/characters	Renvoie tous les personnages appartenant aux joueurs de l'alliance + code 200	1
GET	/users/id/characters	Renvoie tous les personnages du joueur avec id + code 200	1
GET	/characters/all/class	Renvoie tous les personnages avec la classe correspondante + code 200	1
GET	/alliances/id/characters/class	Renvoie tous les personnages de l'alliance avec la classe correspondante + code 200	2
GET	/characters/id/allies/radius	Renvoie tous les autres personnages de la même alliance (personnage actuel exclu) dans un radius en mètres (on considère que character.point.x = lat et character.point.y = long) + code 200	2
GET	/characters/id/ennemies/radius	Idem avec les personnages ennemies (cad n'appartenant pas à l'alliance)	2
Total			10

Note : toutes les routes doivent renvoyer une erreur 500 si une erreur arrive côté serveur + la stacktrace de l'erreur (donc si quelque part dans vos promesses vous avez un « throw error », cela doit remonter jusqu'à votre route et elle doit renvoyer 500 + l'erreur)

```
.catch((error) =>
  res.status(500)
    .json({
      status: 'Error',
      message: error
    })
)
```

c. Visualisation

La partie visualisation est libre au niveau technologie (HTML statique, serveur node.js + Pug, React, VueJS, Android, etc.). Seul iOS est exclu car je n'ai pas de périphérique de test. La notation se compose de plusieurs parties :

- Des « vues » : 4.5 points
- De bonus sur les vues : 4 points
    - Si l'alliance_id d'un user est remplacé par le nom de l'alliance : 1 point
    - Si le user_id d'un character est remplacé par le nom du user : 1 point
    - Si sur la vue d'un character on a une map (Gmap, OpenStreetMap) centrée sur le character et avec 1 pin sur la position du character : 2 points
- Si le site est « responsive » : 1.5 points

Les vues attendues (localhost peut avoir un port) :

URL	Résultat attendu	Points
/localhost/users	Affichage de tous les utilisateurs avec leurs informations de base (name, email, alliance id). Sur le OnClick chaque « box » doit amener à /localhost/users/id.	1
/localhost/alliances	Affichage de toutes les alliances avec leurs informations de base (name). Sur le OnClick chaque « box » doit amener à /localhost/alliances/id.	1
/localhost/characters	Affichage de tous les personnages avec leurs informations de base (name, class, user_id, point). Sur le OnClick chaque « box » doit amener à /localhost/characters/id.	1
/localhost/users/id	Affichage des informations d'un utilisateur (style page de profil).	0.5
/localhost/alliances/id	Affichage des informations d'une alliance (style page de profil).	0.5
/localhost/characters/id	Affichage des informations d'un character (style page de profil).	0.5
		4.5

2. Déploiement et détails

Pour mettre en place votre DB :

- psql -U postgres < schema.sql
- psql -U postgres efrei < data-tests.sql

- Je dois pouvoir « git clone » votre serveur REST et le lancer via un « npm install » + « npm start »
- Les tests unitaires doivent être présents dans le dossier Test de votre projet d'API
- Votre projet vue s'il est statique peut être sous la forme d'une archive dont vous me fournirez le lien. Sinon le processus doit être le même que pour le serveur REST (clone, install, start)
- En cas d'application mobile il faut que l'application se connecte au serveur REST sur un réseau local. Je ne perdrais pas de temps à configurer au-delà d'un fichier txt.
- Les readme.md de vos repositories doivent lister le nom des 2 membres du groupe

3. Unit testing

Les unit test sont déjà prêts. Il suffit de unzip l'archive dans un dossier /tests à la racine du projet (le zip est sur discord ; https://discord.gg/GfKc5wR).

Pour run les tests ;

- Télécharger le fichier Database.js sur Discord et remplacer l'ancien
- Remplacer tous vos « DB. » par « DB.accessor. »
- Télécharger le fichier data-tests.sql et placez le dans dumps/
- npm install -g ava
- ava --init
- npm install -s supertest
- ava --serial --verbose tests/