

Tests et sécurités

Groupe : BARRIER Thomas et BERTHIAU Olivier

Encadrant : SENIS François, DOULCIER Valentin, ARCICAULT Quentin

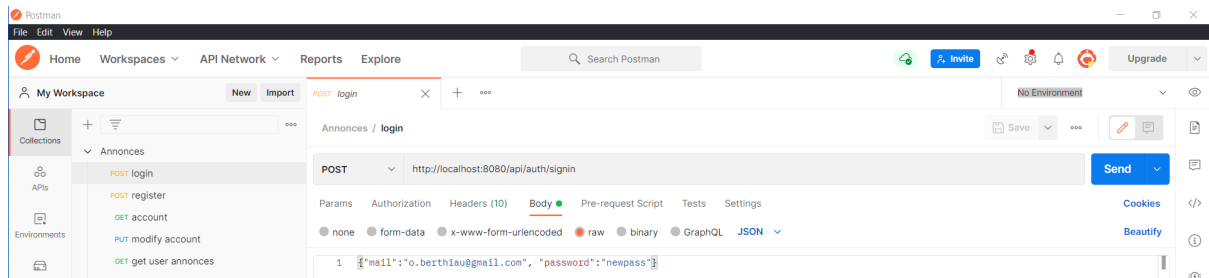
Sujet : Tests sécurités

Tests unitaires avec Postman	2
Sécurité	6
Hashage des mots de passe	6
Utilisation des tokens	7

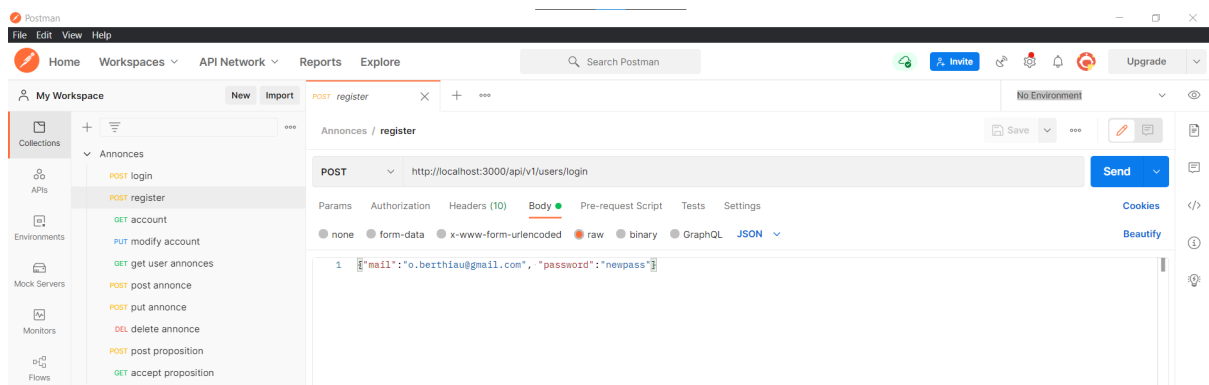
1. Tests unitaires avec Postman

La première partie de nos tests a consisté à effectuer des requêtes unitaires avec Postman afin de valider le bon fonctionnement de notre API backend. Ci-dessous se trouve la liste des requêtes que nous avons effectués :

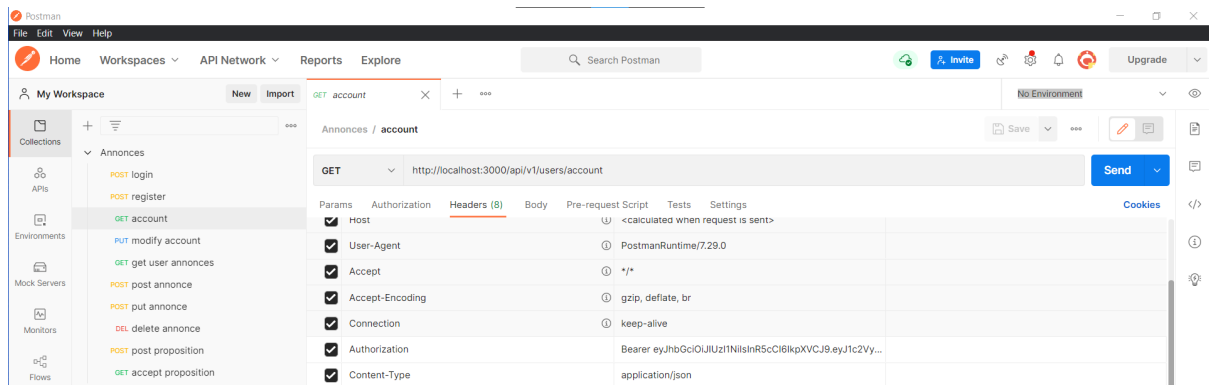
- POST login → Se connecter à son compte utilisateur



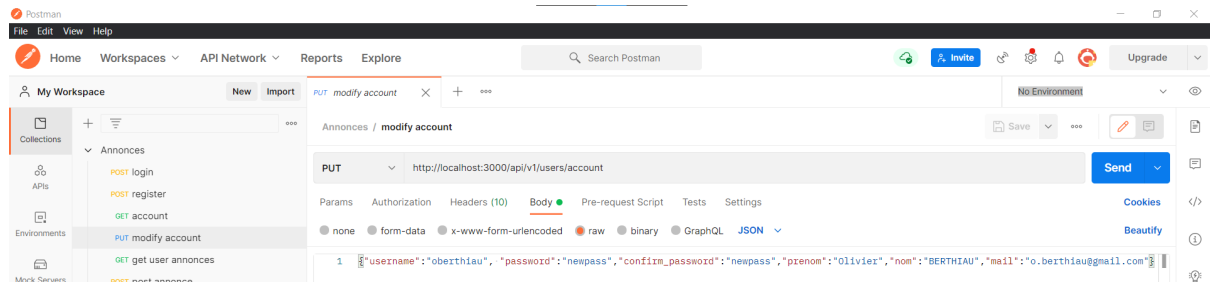
- POST register → Création d'un compte utilisateur



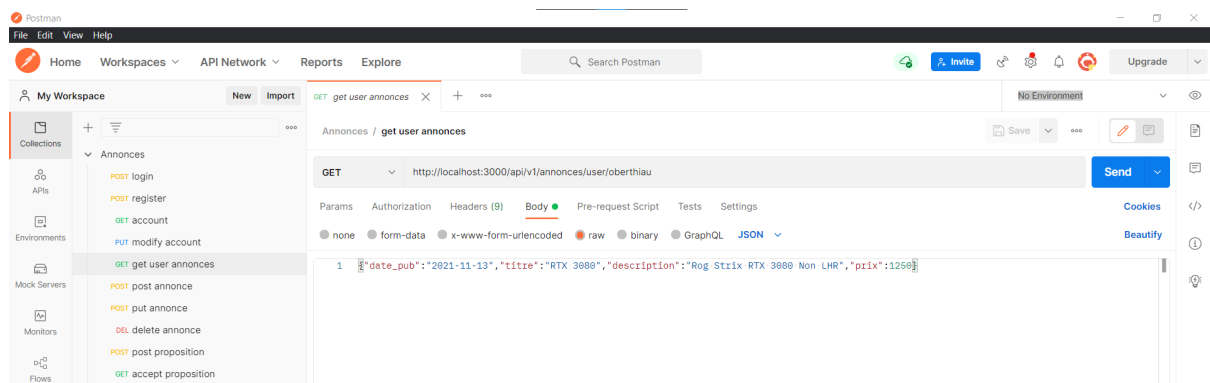
- GET account → Accéder aux informations de son compte (par utilisation du token)



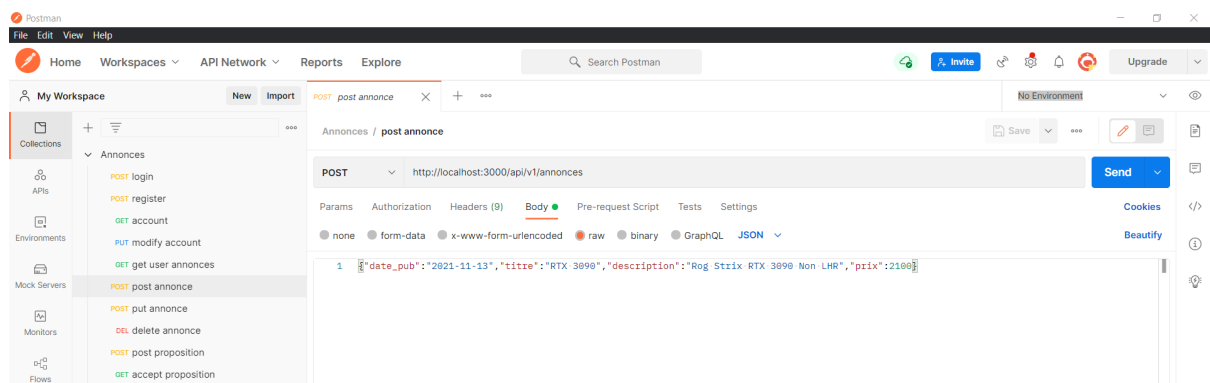
- PUT account → Modifier les informations de son compte (par utilisation du token)



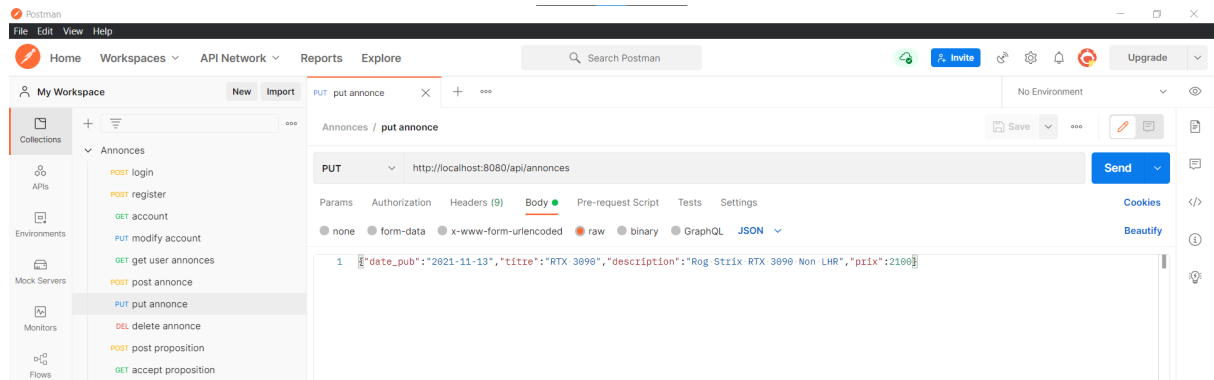
- GET user annonces → Accéder aux annonces d'un utilisateur



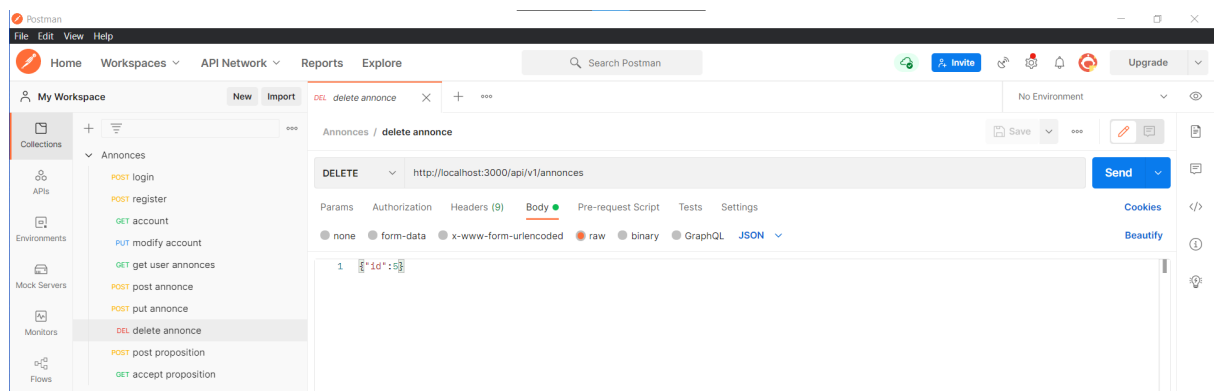
- POST annonce → Créer une publication (authentification obligatoire)



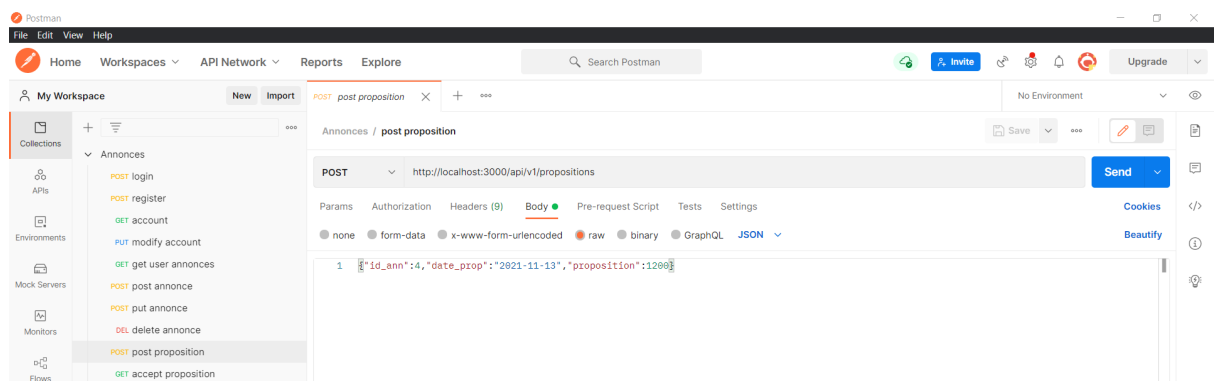
- PUT annonce → Modifier une annonce (authentification obligatoire)



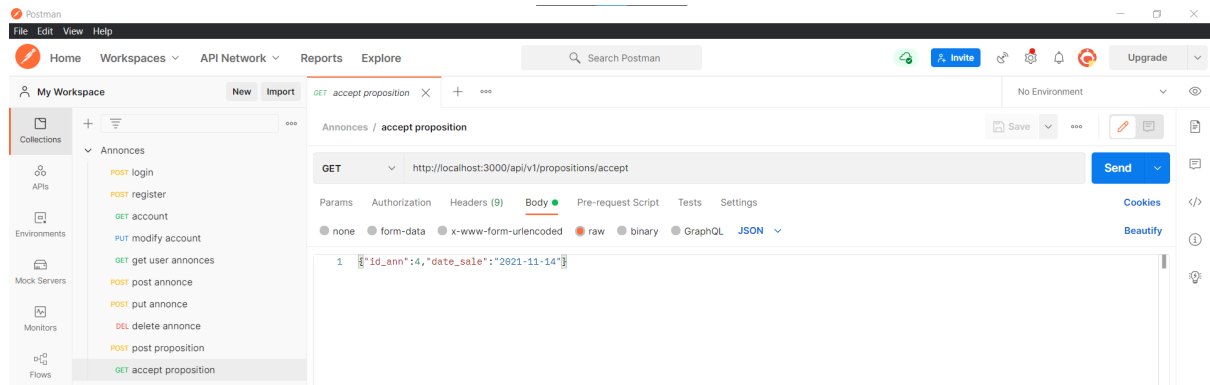
- DELETE annonce → Supprimer une annonce (authentification obligatoire)



- POST propositions → Faire une proposition de prix (authentification obligatoire)



- GET propositions → Accepter une proposition de prix
(authentification obligatoire)

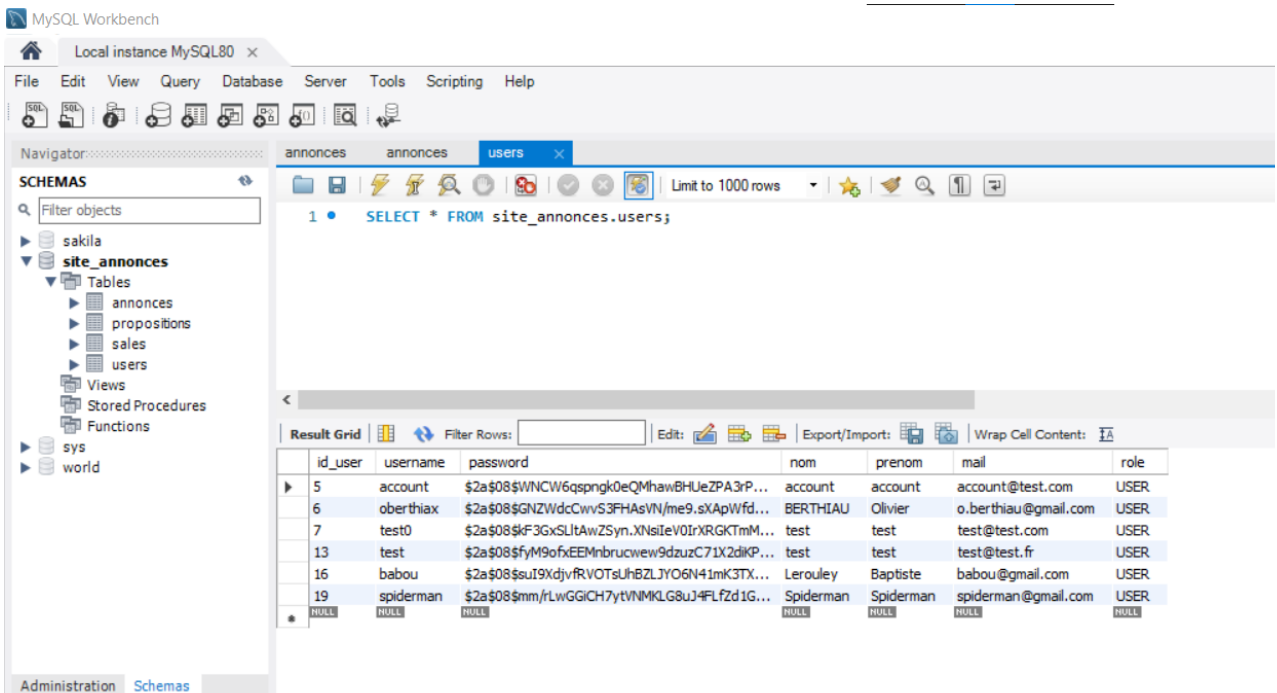


2. Sécurité

2.1. Hashage des mots de passe

Une des étapes importantes de la sécurité de notre site web a été bien évidemment la sécurisation des mots de passe. Pour se faire, nous avons utilisé une librairie JavaScript qui se nomme “bcrypt”. Cette librairie permet de faire un hashage de mots de passe. Comme on peut le voir sur les 2 captures d’écrans qui suivent, tous les mots de passe sont hashés de bout en bout.

```
// hash password if it exists
hashPassword = async (req) => {
  if (req.body.password) {
    req.body.password = await bcrypt.hash(req.body.password, 8);
  }
}
```

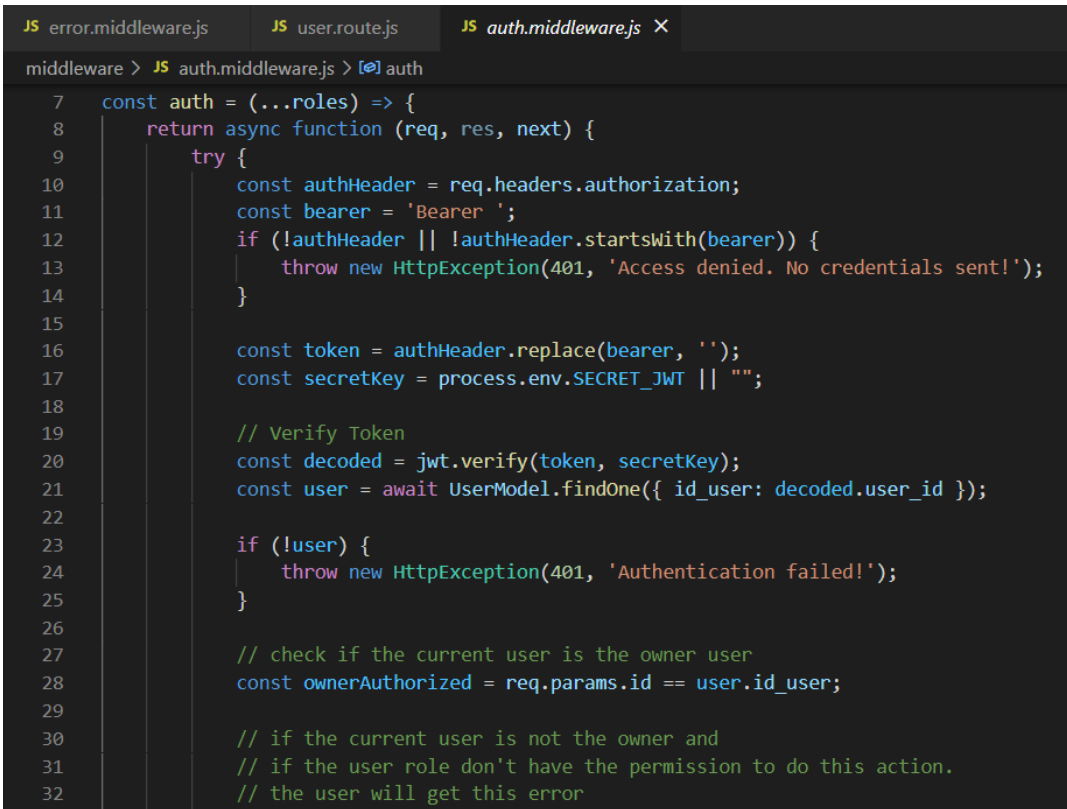


MySQL Workbench interface showing the 'users' table in the 'site_annonces' database. The table contains columns: id_user, username, password, nom, prenom, mail, and role. The 'password' column shows hashed values using bcrypt.

id_user	username	password	nom	prenom	mail	role
5	account	\$2a\$08\$WNCW6qspngk0eQMhawBHueZPA3rP...	account	account	account@test.com	USER
6	oberthiax	\$2a\$08\$GNZWdcCwvS3FHAsVN/me9.sXApWfd...	BERTHIAU	Olivier	o.berthiau@gmail.com	USER
7	test0	\$2a\$08\$Kf3GxSLtAwZSyn.XNsiIeV0IrXRGTmM...	test	test	test@test.com	USER
13	test	\$2a\$08\$fym9ofxEEMnbrucwew9dzuzC71X2dkP...	test	test	test@test.fr	USER
16	babou	\$2a\$08\$suI9XdjvFRVOTsUhbZLJYO6N41mk3TX...	Lerouley	Baptiste	babou@gmail.com	USER
19	spiderman	\$2a\$08\$mm/rLwGGICH7ytVNMKLg8uJ4FLfzd1G...	Spiderman	Spiderman	spiderman@gmail.com	USER

2.2. Utilisation de tokens

Un autre point pour la gestion des utilisateurs est l'utilisation des tokens. Pour mettre en place cette fonctionnalité, nous avons utilisé un standard du JavaScript qui se nomme "JSONWebToken". L'utilisation de token nous permet de gérer les actions qui sont possibles pour chaque utilisateur. La capture d'écran ci-dessous montre un extrait de la fonction qui gère l'authentification.



```
JS error.middleware.js JS user.route.js JS auth.middleware.js X
middleware > JS auth.middleware.js > [⌕] auth
7  const auth = (...roles) => {
8      return async function (req, res, next) {
9          try {
10             const authHeader = req.headers.authorization;
11             const bearer = 'Bearer ';
12             if (!authHeader || !authHeader.startsWith(bearer)) {
13                 throw new HttpException(401, 'Access denied. No credentials sent!');
14             }
15
16             const token = authHeader.replace(bearer, '');
17             const secretKey = process.env.SECRET_JWT || "";
18
19             // Verify Token
20             const decoded = jwt.verify(token, secretKey);
21             const user = await UserModel.findOne({ id_user: decoded.user_id });
22
23             if (!user) {
24                 throw new HttpException(401, 'Authentication failed!');
25             }
26
27             // check if the current user is the owner user
28             const ownerAuthorized = req.params.id == user.id_user;
29
30             // if the current user is not the owner and
31             // if the user role don't have the permission to do this action.
32             // the user will get this error
```