Vannes
iut
Université Bretagne Sud

Cahier des Tests du projet MathsALaMaison (Application et Serveur)

TABLE DES MATIERES

INTRODUCTION

Ce présent document contient les différents tests réalisés dans le cadre du projet MathsALaMaison.

APPLICATION

UNITAIRE

PLAYERCONTEXT

Objectif :

Le contexte PlayerContext gère les joueurs dans la partie jeu de l'application, leurs points, le joueur actif, ainsi que diverses méthodes pour manipuler cet état partagé. Les tests unitaires permettent de garantir que chaque fonction du contexte se comporte correctement, évitant ainsi des bugs difficiles à détecter en production.

Outils utilisés

Jest : Framework de test JavaScript, utilisé pour exécuter les tests et faire les assertions.

React Testing Library : Pour rendre les composants React dans un environnement de test et interagir avec eux de manière proche du comportement utilisateur.

Tests unitaires réalisés

Fonction testée	Description du test	Comportement attendu vérifié
État initial	Teste que le contexte démarre avec un état vide	*players = [], points = {}, currentPlayer = null*
setPlayers	Initialise la liste des joueurs avec des points à zéro	*players* mis à jour, *points* initialisé à zéro, *currentPlayerIndex* à 0
setPoints	Modifie directement les points	L'état *points* correspond exactement aux nouvelles valeurs
addPoint ToCurrentPlayer	Ajoute des points au joueur courant	Le score du joueur courant est incrémenté correctement
getPointsOfPlayer	Récupère le score d'un joueur spécifique (via affichage indirect)	Affiche le score correct pour chaque joueur

nextPlayer	Change le joueur courant à celui suivant dans la liste	currentPlayerIndex et currentPlayer changent cycliquement
resetPlayers	Vide la liste des joueurs et des points, remet à zéro	*players* vide, points vide, *currentPlayerIndex* remis à 0

INTEGRATION

Les tests d'intégrations sont l'ensemble des tests qu'on a effectués pour vérifier le bon fonctionnement de toutes les parties du projet. Dans le projet MathsALaMaison nous avons 3 parties distinctes :

- L'application Web React
- Le serveur NodeJS
- La base de données MySQL

Chaque partie doit correctement faire son travail et chaque partie doit également pouvoir communiquer entre eux. L'application Web React ne communique pas directement avec la base de données, il va le faire avec l'intermédiaire du serveur.

A chaque fois qu'une fonctionnalité avait été développé on a testé la connexion entre : l'application ←→ le serveur ←→ la base de données. Si une fonctionnalité ne concernait pas la connexion entre ces différents éléments alors ont testé uniquement l'impact que cela avait.

Pour développer une nouvelle fonctionnalité (exemple : reset-password), on regardait d'abord comment cette fonctionnalité était développé pour les sites fiables (Google, Apple, Microsoft).

Après avoir une idée approximative de l'implémentation que cela avait nous développions toujours les routes API en premier. Une fois que l'API était finie, la documentation était disponible via le lien : « https://mathsalamaison-backend.omergs.com/api-docs ». Cette documentation est une documentation OpenAPI qui permet de voir le fonctionnement de chaque route présente sur le serveur.

IMPLEMENTATION

Les tests d'implémentation nous ont aidé à vérifier que le code qu'on a écrit faisait bien ce qu'on voulait.

A chaque version notable, nous envoyons une version testable à différentes personnes pour avoir leur retour et les différents problèmes rencontrées.

Donc lorsqu'une personne de l'équipe avait fini de développé une fonctionnalité, il testait tous les cas qu'il pouvait. Lorsqu'on avait fait plusieurs fonctionnalités notables, on sortait une version testable pour que les utilisateurs puissent les testés. Les utilisateurs pouvaient testés les nouveautés via les versions de productions qui se généré automatiquement lorsqu'un changement avait lieu dans la branche main via le lien « https://mathsalamaison.omergs.com ».

Nous avons choisi les testeurs en fonction de l'appréhension technologiques qu'ils avaient, nous avons choisi un étudiant en informatique (Mael COIGNARD en BUT1), et des personnes de métiers et âges différentes. Les testeurs était des personnes également avec un appareil différent des personnes de ceux du groupe, pour voir si l'affichage sur tout type de format était correct.

SECURITE

Les tests de sécurités sont l'ensemble des tests qu'on a effectué pour protéger les données et l'application MathsALaMaison.

MOT DE PASSE

Pour protéger les mots de passe des comptes utilisateurs, nous ne stockons pas les mots de passe directement en clair dans la base de données. Nous utilisons ce qu'on appelle une méthode de hachage avec sel côté serveur.

FONCTIONNEMENT

Lorsqu'un utilisateur veut créer ou modifier son mot de passe :

1. Le client saisie son mot de passe en clair
2. Il est transmis au serveur via une connexion sécurisés (HTTPS)
3. Le serveur va ensuite :
    a. Récupérer le mot de passe envoyé par le client
    b. Générer un sel qui est une suite de caractère aléatoire
    c. Création d'un hash du mot de passe en appliquant le sel

d. Enregistrement du hash et du sel dans la base de données

Lorsque l'utilisateur veut se connecter à son compte :

1. Le client saisie son mot de passe en clair
2. Il est transmis au serveur via une connexion sécurisés (HTTPS)
3. Le serveur récupère le sel depuis la base de données
4. Il rehache le mot de passe envoyé par le client avec le sel
5. Il vérifie que le hash obtenu est le même que celui stocké dans la base de données

POURQUOI HACHE LES MOTS DE PASSE AU LIEU DE LES STOCKER EN CLAIR ?

Nous hachons les mots de passe car dans le cas d'une fuite de données les mots de passe serais compromis et toute personne ayant accès à la base pourrais se connecter au compte des clients.

En utilisant un hash avec un sel, même si un pirate à accès à la base de données, il ne pourrait pas faire grand-chose des mots de passe, car il ne peut pas les lires et s'il essaye de rentrer le hash dans le formulaire de login, le serveur va juste dire que le hash est un mot de passe long et va refuser la connexion.

GESTIONS DES DROITS UTILISATEURS : JWTS

Pour la gestion de l'authentification et des droits utilisateurs (pour éviter qu'un utilisateur puissent aller dans le panel administrateur) nous avons choisi d'utiliser les JSON Web Token (JWT).

Un JWT est un jeton sécurisé qui contient les informations encodées, comme l'identifiant, l'adresse email, le rôle de l'utilisateur (comme par exemple : « admin » ou « user »). Ce jeton est signé par le serveur afin de garantir que personne ne puisse changé ce qu'elle contient.

Le fonctionnement est plutôt simple :

1. Lorsqu'un utilisateur se connecte avec succès, le serveur génère un JWT contenant les informations suivantes : « userId, email et randValue ». (RandValue permet que chaque JWT soit différent les uns des autres)

2. Ce jeton est ensuite envoyé au navigateur et est stocké dans les cookies du navigateur.
3. A chaque requêtes où il faut une vérification de l'identité le navigateur va automatiquement envoyer les tokens.
4. Le serveur peut alors vérifier l'identité de la personne qui envoie les requêtes et autorisé ou refuser l'accès.

CORS

CORS (Cross Origin Ressource Sharing) est un mécanisme de sécurité qui permet à un serveur de définir explicitement quels domaines peuvent l'appeler. Par exemple un serveur ayant un domaine bonjour.fr ne pourras pas appeler croissant.fr sans avoir autorisé explicitement dans les CORS. Cela permet de sécuriser l'accès et les appels au serveur

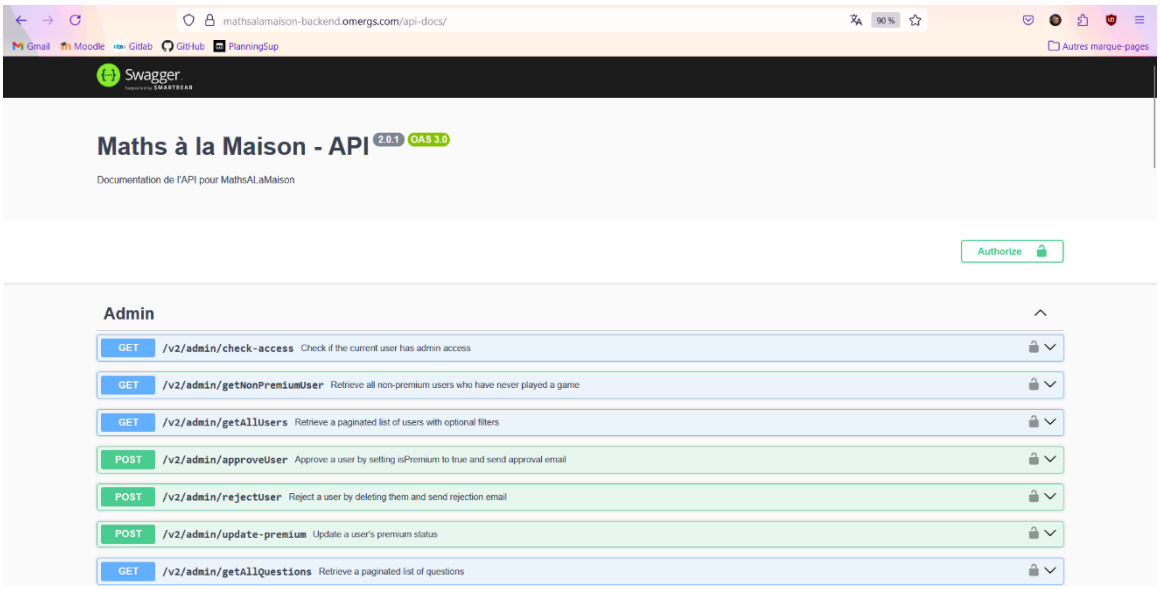
CLE API

En complément du CORS nous avons ajouté un token spécifique à l'application front-end, l'application doit pour cela envoyé un token api dans le headers dans le « x-api-key ». Le serveur vérifie ensuite que le token est bien le même que celui attendu et autorise ou pas l'accès au serveur.

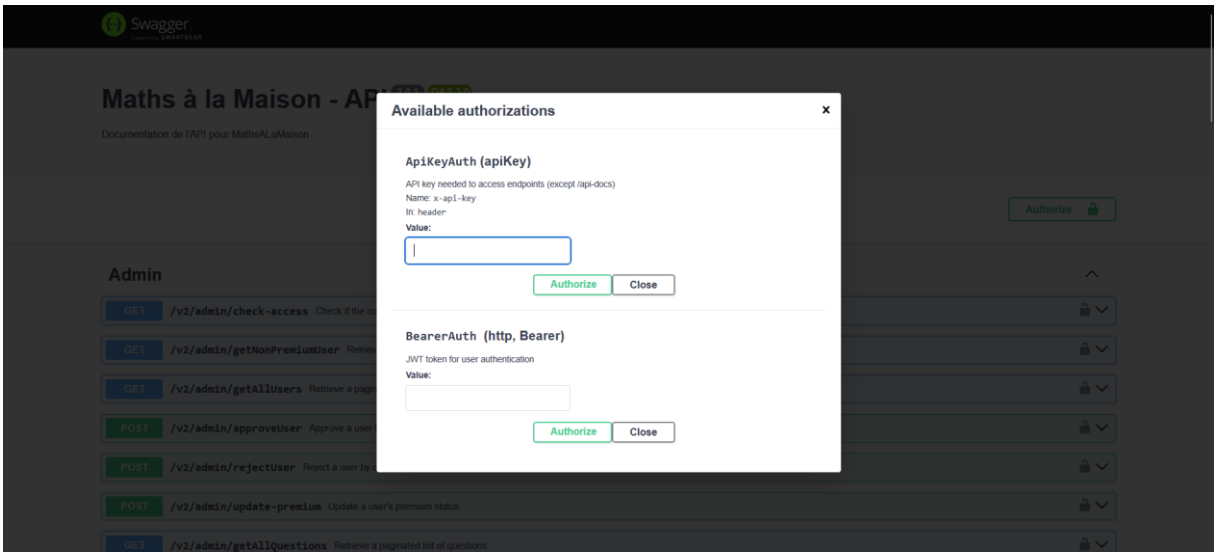
L'ajout d'un token API, des JWTs et des CORS permettent de sécuriser l'accès aux serveurs NodeJS de MathsALaMaison.

TEST DE L'API

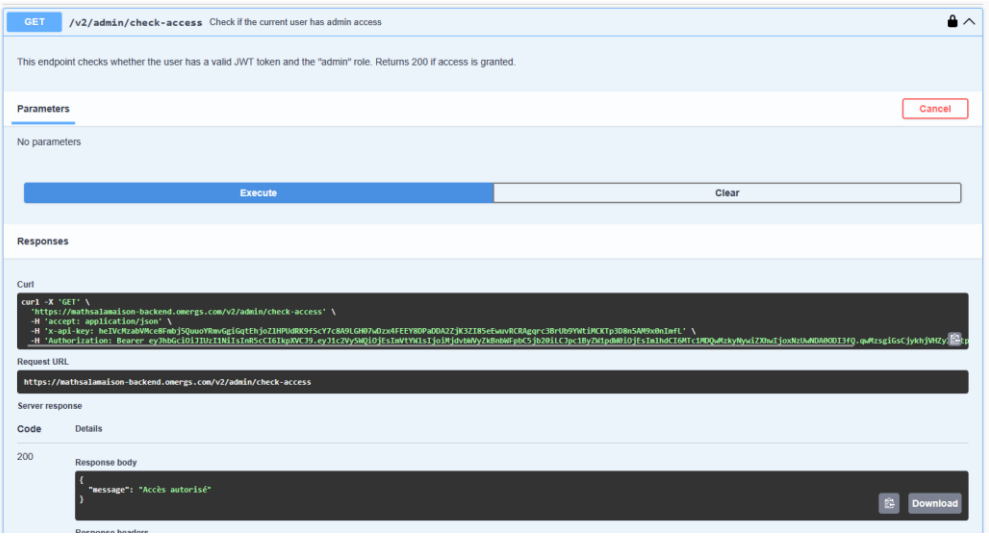
Pour tester l'API nous avons utiliser CURL et l'interface Swagger pour vérifier que les réponses que nous donner l'API était bien ce qu'on voulait. Pour tester nous allons dans la route « /api-docs » du serveur de jeu de MathsALaMaison, durant le développement nous avons utilisé l'URL : « mathsalamaison-backend.omergs.com ».



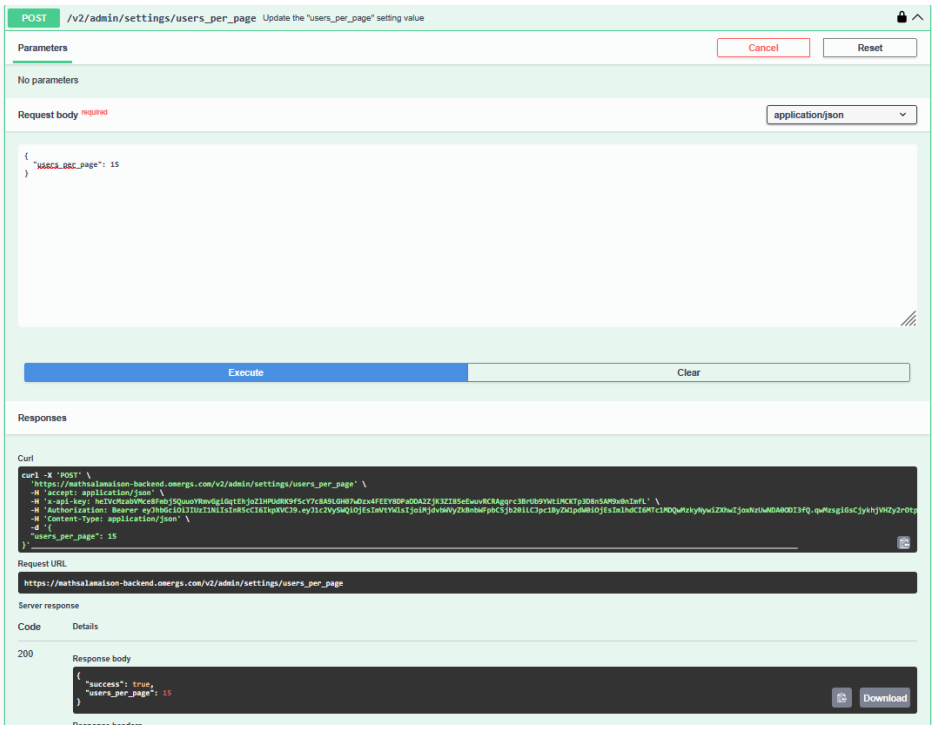
Grâce à cette page nous avons pu tester nos routes API en fournissons les jetons d'accès et la clé API nous pouvons tester toutes les routes que nous avons accès via notre jeton d'accès.



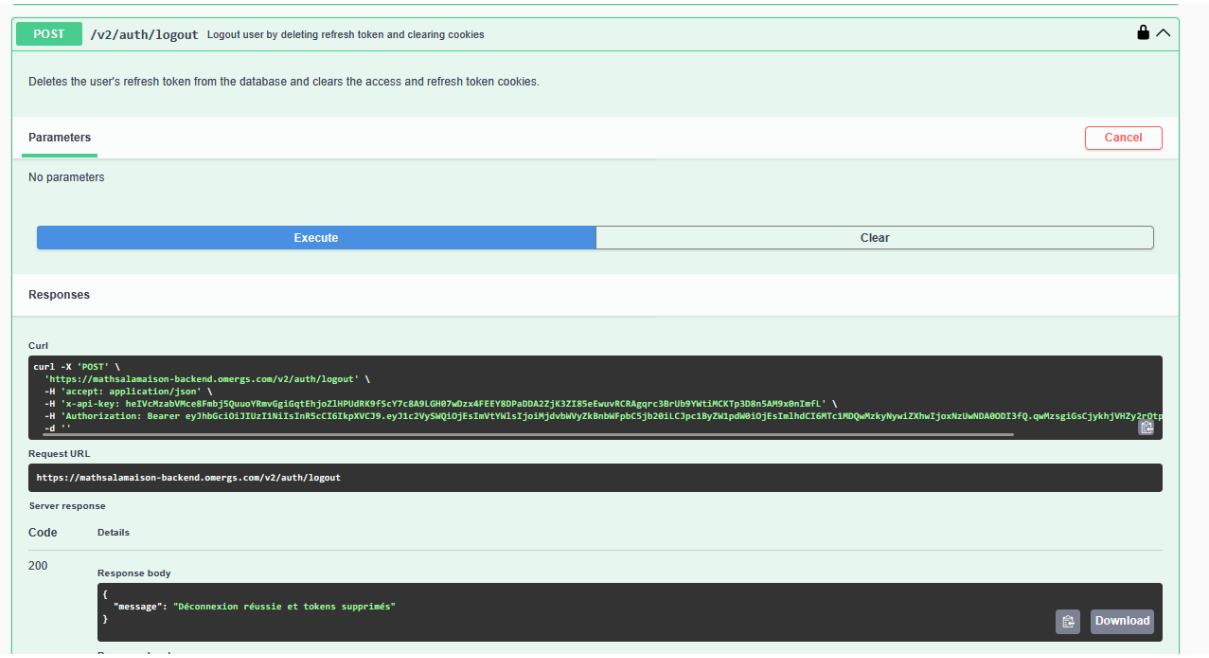
Pour tester la route « /v2/admin/check-access/ », on s'est donc connecté avec un compte qui avais un accès administrateur et des token valide. Une fois qu'on a cliqué sur « Try it out » on pouvait peut bien voir que la réponse de l'API est bien reçue : { message : « Accès autorisé » }



Pour tester la route « /v2/admin/settings/users_per_page/ », on s'est donc connecté avec un compte qui avais un accès administrateur et des token valide. Une fois qu'on a cliqué sur « Try it out » on pouvait peut bien voir que la réponse de l'API est bien reçue et que la valeur à bien changé dans la base de données.



Nous avons également pu tester des actions utilisateurs directement ici, sans avoir besoin d'aller dans l'application. Ici on teste la route « /v2/auth/logout/ » pour vérifier la réponse de notre API.



Nous pouvons bien voir dans les logs du serveur que je fais des appelles via mon compte à moi comme si j'étais dans l'application MathsALaMaison. Ce qui permet de développer l'API pendant que le front-end de la fonctionnalité est entrain d'être développé ce qui nous permets de gagner du temps.

```
gunes@gunes:~/SAE/MathsALaMaisonServer/src$ ./start
[2025-06-20 09:45:16.358 +02:00] MathsALaMaison - LocalAdmin v.3.0.1
>
 __  __       _   _      __  __       _   _       _                 _
|  \/  | __ _| |_| |__  |  \/  | __ _| |_| |__   | |    __ _ _ __ | |__   ___  _ __
| |\/| |/ _` | __| '_ \ | |\/| |/ _` | __| '_ \  | |   / _` | '_ \| '_ \ / _ \| '_ \
| |  | | (_| | |_| | | || |  | | (_| | |_| | | | | |__| (_| | | | | | | | (_) | | | |
|_|  |_|\__,_|\__|_| |_||_|  |_|\__,_|\__|_| |_| |_____\__,_|_| |_|_| |_|\___/|_| |_|

[2025-06-20 09:45:16.358 +02:00]
[2025-06-20 09:45:16.358 +02:00] Licensed under The MIT License (use command "license" to get license text).
[2025-06-20 09:45:16.358 +02:00] Created by OmerGS and Romain Péron and Rayanne Mellah and Noé Parcollet, 2024 - 2025
[2025-06-20 09:45:16.358 +02:00]
[2025-06-20 09:45:16.358 +02:00] Build Date: 2025-06-20T07:45:16.358Z
[2025-06-20 09:45:16.358 +02:00] Environment: production
[2025-06-20 09:45:16.358 +02:00] Node.js Version: v23.7.0
[2025-06-20 09:45:16.358 +02:00] Platform: Linux 6.8.0-59-generic (x64)
[2025-06-20 09:45:16.359 +02:00] Repo: https://github.com/OmerGS/MathsALaMaison
[2025-06-20 09:45:16.359 +02:00]
[2025-06-20 09:45:16.359 +02:00] Type 'help' to get list of available commands.
[2025-06-20 09:45:16.359 +02:00] Have fun! 🚀
[2025-06-20 09:45:16.359 +02:00]
[2025-06-20 09:45:17.779 +02:00] Started new session on port 5000
[2025-06-20 09:45:17.779 +02:00] Connecting to database host: 127.0.0.1:3306
[2025-06-20 09:45:17.780 +02:00] Database found: mathsALaMaison
[2025-06-20 09:45:17.780 +02:00] User 'app' connected to database
[2025-06-20 09:45:17] INFO: WebSocket server started
[2025-06-20 09:45:20] INFO: [GET] request from 192.168.1.254 by 27omerf@gmail.com to /api-docs/
[2025-06-20 09:45:20] INFO: [GET] request from 192.168.1.254 by 27omerf@gmail.com to /api-docs/swagger-ui.css
[2025-06-20 09:45:20] INFO: [GET] request from 192.168.1.254 by 27omerf@gmail.com to /api-docs/swagger-ui-standalone-preset.js
[2025-06-20 09:45:20] INFO: [GET] request from 192.168.1.254 by 27omerf@gmail.com to /api-docs/swagger-ui-bundle.js
[2025-06-20 09:45:20] INFO: [GET] request from 192.168.1.254 by 27omerf@gmail.com to /api-docs/swagger-ui-init.js
[2025-06-20 09:45:52] INFO: [POST] request from 192.168.1.254 by 27omerf@gmail.com to /v2/auth/logout
[2025-06-20 09:45:52] INFO: Refresh Token supprimé de la base de données.
```

Nous avons testé toutes les routes API de notre serveur avec différents cas pour s'assurer du bon fonctionnement de celle-ci.