Examen Final - Sommatif H24

# 420-5W5

## Consignes

* Vous avez 1h50 pour compléter l’examen.
* Toute documentation est permise pour l’examen.
* L’utilisation d’IA est **interdite.**

# Question 1 – Tests unitaires et Mock

À partir des fichiers de départs dans **Mock**, tester **SeatsController**.

Le contrôleur est utilisé dans un système de réservation de places pour différents concerts et évènements.

Pour tester le contrôleur, vous devrez utiliser un ou plusieurs mock.

Vous devrez donc utiliser le package Nuget Moq.

ReserveSeat

L’action sert à réserver une place dans une salle de concert. Il doit y avoir 4 tests pour couvrir les 4 cas suivants. Chaque cas doit inclure un assert pour vérifier le résultat de l’appel à **ReserveSeat**.

1. Lorsque l’action s’exécute correctement, la place (classe Seat) est retournée à l’utilisateur.
2. Dans le cas où l’utilisateur connecté essaie de réserver une place qui est déjà réservée par un autre utilisateur, l’action retourne **Unauthorized**.
3. Dans le cas où le numéro de la place sélectionnée est plus que le maximum (la salle contient 100 places), l’action retourne **NotFound**.
4. Dans le cas où l’utilisateur connecté essaie de réserver une place, mais qu’il a déjà une place de réservée, l’action retourne **BadRequest**.

# Question 2 – BackgroundService et SignalR

**ATTENTION** : L’exemple utilise un système d’authentification par cookie. Le système est **déjà en place**, mais il faut utiliser **ng serve –-ssl** pour qu’il fonctionne.

**ATTENTION** : À l’école, vous aurez également à rouler :

* dotnet dev-certs https –clean
* dotnet dev-certs https --trust

Ce qui existe déjà :

* Un client qui se connecte à un serveur avec une authentification par Cookie.
* Un serveur qui génère une nouvelle question de Math chaque 20 secondes.
* Un client qui affiche cette question et les 4 réponses possibles.
* La possibilité pour un joueur de choisir une réponse.

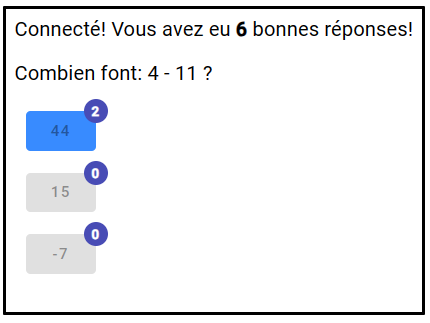
Il faut encore :

1. Mettre à jour les choix des joueurs qui sont déjà affiché avec un badge. Faites-le en utilisant le message **IncreasePlayersChoices** qui est déjà écouté et est déjà fonctionnel sur le client. Il faut envoyer ce message dans la méthode **SelectChoice** de **MathBackgroundService. [Clarification : Ici, on affiche simplement le nombre de joueurs qui ont fait ce choix, pas si c’est la bonne réponse ou pas !]**

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

(Ici on voit qu’un autre joueur a choisi la réponse 44)



(Si le joueur fait le même choix, on voit qu’il y a maintenant 2 joueurs qui ont fait ce choix)

Il existe déjà une méthode **EvaluateChoices** dans **MathBackgroundService** qui évalue les réponses, vous devez ajouter votre **code serveur** dans cette méthode pour :

1. Que les joueurs voient s’ils ont eu la bonne réponse ou non :

[Serveur] Notifier les clients avec un message **SignalR** qui indique qu’ils ont eu la bonne réponse si c’est le cas.

[Serveur] Notifier les clients avec un message **SignalR** qui indique qu’ils ont eu la mauvaise réponse si c’est le cas.

[Client] Faire un **alert** qui affiche « Bonne réponse ! » et qui met à jour **nbRightAnswers** lorsque le client reçoit un message qui indique une bonne réponse.

[Client] Faire un **alert** qui affiche « Mauvaise réponse ! La bonne réponse était X » où X est la bonne réponse à la question.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Note : Si un joueur prend **trop de temps** pour répondre, il a une **mauvaise** réponse.

1. Mettre à jour le nombre de bonnes réponses **(NbRightAnswers)** des joueurs qui ont eu la bonne réponse dans la **BD**. (Vérifier que la donnée est encore bonne après un refresh de la page)

# Question 3 – Animations

À partir des fichiers de départs dans **ngAnimations**,

Lorsque l’on clique sur **« Animer une fois »**, faites les animations **Angular** suivantes :

**Animation shake** du carré **rouge** pendant **0.6 seconde**

Aussitôt que la première animation est terminée :

**Animation bounce** du carré **vert** pendant **0.8 seconde**

**100 ms avant** que l’animation du carré vert soit terminée : (Comprendre que le délai doit être de 0.7 seconde)

**Animation tada** du carré **bleu** pendant **0.9 seconde**

Pour le bouton **« Animer en boulce »**, faites exactement les mêmes animations (possible d’utiliser la même méthode, mais avec un paramètre boucle:boolean) et recommencez à les jouer aussitôt que la 3e animation est terminée.

Pour le bouton **« Faire tourner »,** utiliser l’animation **css** avec la classe **rotate-left.** Il faut simplement faire jouer l’animation une fois en **750 ms**. **Une fois que l’animation est terminée**, on peut cliquer encore sur le bouton et l’animation va jouer.