Commencé le	mercredi 19 octobre 2022, 18:00
État	Terminé
Terminé le	mercredi 19 octobre 2022, 19:35
Temps mis	1 heure 35 min
Note	**18,00** sur 25,00 (**72**%)

Description

LISEZ CES INSTRUCTIONS AVANT DE COMMENCER L'EXAMEN

- L'examen possède 12 questions notées sur un total de 25 points.
- La note partielle associée à chaque question est marquée à gauche de la question.
- Certaines questions demandent d'ajouter une justification à votre réponse. Une bonne réponse sans justification valide ne sera pas acceptée.
- **Vous avez une seule tentative pour l'examen! Ne soumettez pas votre tentative à moins d'être 100% sûr(e) d'avoir terminé l'examen ! La soumission sera automatique à la fin de la période.**
- **En cas de doute sur le sens d'une question, faites une supposition raisonnable, énoncez-là clairement dans votre réponse et poursuivez. Une "question" vide est disponible sur la première page d'examen si vous avez besoin de plus de place ou pour des questions spécifiques.**
- **Vous avez accès à une copie des notes de cours PDF sur l'instance Moodle et droit à 1 feuille recto-verso (imprimée ou manuscrite) comme documentation.**

Bon travail!

Question **1**

Non répondue

Non noté

Cette question est un espace dédié à vos commentaires ou questions sur l'examen. Nous ne répondons à aucune question pendant l'examen.

Priorisez de mettre vos commentaires et/ou hypothèses directement dans la question spécifique.

Question **2**

Correct

Note de 1,00 sur 1,00

Vous décidez de faire communiquer deux de vos Components à travers un Service partagé. Parmi les choix suivants, le ou lesquels sont des avantages architecturaux d'utiliser un Service partagé au lieu de l'utilisation des décorateurs @Input/@Output pour la communication entre 2 Components ?

- a. Les Services sont des singletons, contrairement aux Components.
- b. Les décorateurs nécessitent des variables publiques, ce qui brise le concept d'encapsulation, contrairement à l'utilisation d'un Service privé.
- c. L'utilisation d'un Service diminue le couplage entre les Components. ✔
- d. @Input/@Output permettent seulement le passage de valeurs primitives, contrairement aux Services qui permettent la communication d'objets complexes.
- e. L'utilisation d'un Service élimine le besoin d'avoir un lien parent/enfant entre les Components. ✔

Votre réponse est correcte.

Question **3**

Incorrect

Note de 0,00 sur 1,00

Vous avez un Service qui possède un attribut de type Subject<string> qui est utilisé pour communiquer avec plusieurs de vos Components qui s'y abonnent à travers la méthode *subscribe()*.

Pourquoi est-il important de se désabonner avec la méthode *unsubscribe()* de l'Observable à un moment donné du cycle de vie de chaque Component ?

- a. Un abonnement non terminé pourrait causer une fuite de mémoire.
- b. Un Observable peut être observé par un seul Observer à la fois. Le désabonnement permet aux autres Observers de fonctionner. ✖
- c. La méthode *unsubscribe()* n'est pas nécessaire puisque RxJS est capable de gérer le désabonnement automatiquement pour vous.
- d. La librairie RxJS force l'utilisation des méthodes *subscribe()* et *unsubscribe()* pour les Observables.

Votre réponse est incorrecte.

Question **4**

Terminé

Note de 1,00 sur 3,00

Voici un extrait du code du gabarit HTML de votre classe **GameCarouselComponent** qui possède comme enfants des instances de **GameCardComponent**.
Le tableau **gameCards** possède les jeux à afficher (1 à 4) sous la forme d'un objet **GameCard** qui contient tous les attributs nécessaires pour l'affichage.
GameCardComponent possède un seul attribut public qui est : @Input() gameCard: **GameCard**

```
1 <div id="game-container" *ngIf="gameCards.length > 0">
2     <div id="game-container-2" *ngFor="let gameCard of gameCards">
3         <app-game-card gameCard="gameCard"> </app-game-card>
4     </div>
5 </div>
```

a) Votre collègue vous demande pourquoi est-ce qu'il y a 2 éléments qui englobent <app-game-card>. Que lui répondez-vous ? Est-ce qu'il y a une manière plus simple qui permet d'avoir le même comportement ? Justifiez votre réponse. (2 points)

b) Le compilateur d'Angular lance une erreur lors de la transpilation de la ligne 3 du gabarit. Pourquoi ? (1 point)

a) Je lui répondrait que l'on a mis un premier div pour verifier la condition de la taille puis le second pour afficher les gameCard.

Or je lui dirais également qu'il s'agit d'une bonne question car en effet il serait préférable qu'un seul élément englobe <app-game-card>.

En effet, il existe une façon plus simple d'avoir le même comportement. Premièrement pour avoir une meilleure lisibilité du HTML il aurait été préférable de créer une fonction displayCards() dans gameCarouselComponent.ts qui permettrait de les afficher, on aurait eu donc:

dispayCard(): GameCard[] {

game: GameCard;

```
    if (this.gameCards.length < 0) {
        return;
    }
    for (i=0; i < this.gameCards.length; i++) {
         game = this.gameCards[i];
         return game;
     }
}
```

Par la suite, dans gameCarouselComponent.html, on peut mettre:

```
<div id="game-container" 'displayCard()' >
    <app-game-card [gameCard]="gameCard"> </app-game-card>
</div>
```

b) Le compilateur lance une erreur car le code a été mal écrit, il aurait fallut que ce soit

```
<app-game-card [gameCard]="gameCard"> </app-game-card>
```

Commentaire :
a) On ne peut pas placer 2 directives structurelles sur le même élément

a) *ngFor peut aller directement sur <app-game-card>. Les 2 divs ne sont pas nécessaires. La syntaxe de la réponse donnée (le TS et gabarit Angular) n'est pas valide -2

Question **5**

Correct

Note de 1,00 sur 1,00

Quelle est la différence entre Express, NodeJS et Socket.IO ?

a. La librairie Socket.IO est nécessaire pour traiter le protocole WebSocket, mais Express est optionnelle pour un serveur utilisant NodeJS.

b. NodeJS est un environnement d'exécution à partir duquel on utilise les librairies Express et Socket.IO, pour une gestion à plus haut niveau des requêtes HTTP et de la communication WebSocket, respectivement. ✔

c. La librairie Express est nécessaire pour traiter des requêtes HTTP, mais Socket.IO est optionnel pour un serveur utilisant NodeJS.

d. NodeJS, Express et Socket.IO sont des librairies côté serveur pour la communication réseau.

e. NodeJS est un environnement d'exécution à partir duquel on utilise la librairie Express pour une gestion à plus haut niveau des requêtes HTTP et SocketIO est un protocole de communication bidirectionnelle.

Votre réponse est correcte.

Question **6**

Terminé

Note de 2,00 sur 3,00

À cette étape de la session, vous utilisez au moins 2 serveurs différents pour votre projet. Expliquez quels sont ces serveurs et expliquez brièvement le rôle de chacun dans votre système.

On utilise un server coté client et un autre dans le dossier server.
 Le serveur coté client est un serveur web qui est statique qui permet de nous donner des fichiers qui seront par la suite utilisé par le navigateur.

Alors que le serveur coté server lui est unique, permet la communication avec plusieurs client mais également la communication entre un client et un websocket qui passe par lui. Il peut communiquer avec des BD tel mangoDB qui a la particularité d'être a distance. Il a également l'avantage d'être plus sécuritaire ainsi il est possible de valider certaines fonctionnalité sur ce serveur.

Commentaire :
Qu'est-ce que serveur "unique" veut dire ? MangoDB n'est pas un vrai produit et n'est pas nécessairement déployé sur une machine différente que le serveur dynamique. Vous utilisez HTTP : il n'y a aucune sécurité

Question **7**

Terminé

Note de 3,00 sur 4,00

Voici l'implémentation de la gestion du clavardage dans plusieurs salles de votre projet.
Vous avez présentement 3 clients qui communiquent avec votre serveur : ClientA, ClientB et ClientC.
ClientB et ClientC sont présentement dans la même partie de jeu et dans la même *Room* ayant l'identifiant "XYZ".

Voici la gestion de cet événement du côté serveur. La fonction *getRoomFromSocketId(socketId)* retourne l'identifiant de la salle d'un socket dans une partie de jeu :

```
socket.on('chat', (message) => {
  const room = this.getRoomFromSocketId(socket.id);
  const chatMessage = { room: room , ...message}
  socket.broadcast.emit("chatMessage", chatMessage);
});
```

Voici également la gestion du message du serveur du côté client. L'attribut *roomId* représente le nom de la Room dans laquelle le client est présentement :

```
this.socketService.on('chatMessage', (chatMessage) => {
  if (this.roomId === chatMessage.room) {
    this.chatMessages.push(chatMessage);
  }
});
```

ClientB vient d'envoyer l'événement "chat" avec l'objet **{ message: "Salut", player: "ClientB" }** au serveur.

a) En fonction de la configuration donnée, qui recevera un message du serveur et **pourquoi** ? (1.5 points)

b) Cette implémentation possède un problème avec la gestion des messages dans les salles. Donnez le problème avec l'implémentation et proposez une solution possible. (2.5 points)

a) Selon cette configuration, le ClientA et le ClientC recevront le message. En effet, ici, le clientB émet le message et donc le conservera, puis une fois que le server reçoit l'évènement, il lancera le broadcast ainsi le message sera envoyé au ClientA et ClientC. Le message n'est pas envoyé au ClientB car il s'agit de l'émetteur. Le ClientA et ClientC vérifient si le message leur est bien destiné, si cela est le cas il push le message.

b) Cette implémentation possède un problème avec la gestion des messages dans les salles car même si certain client ne sont pas dans la même salle que l'émetteur, il recevront le message. Ainsi il aurait été préférable de faire une vérification coté server que l'on envoie le message uniquement aux clients de la même room que l'émetteur, pour cela il aurait été possible de (coté server):

io.to("room").emit("chatMessage", chatMessage);

Commentaire :
a) Rien dans le code indique que le ClientB conserve le message.

b) chatMessage est un objet inutile puisqu'il ne faut plus envoyer l'objet "room" : on peut seulement envoyer l'objet message. La syntaxe to("room") est invalide : ceci envoie seulement à la salle dont l'identifiant est la châine "room".

Question **8**

Partiellement correct

Note de 0,50 sur 1,00

Pourquoi faut-il toujours exécuter les tests après avoir amélioré un segment de code?

- [x] a. Pour s'assurer que le code est toujours fonctionnel après les changements. ✔
- [ ] b. Pour s'assurer que les modifications n'ont pas créé de nouveaux défauts dans le code.
- [ ] c. Pour vérifier que les changements apportés sont de bonne qualité.
- [ ] d. Pour vérifier que les tests sont toujours de bonne qualité.
- [ ] e. Pour s'assurer du succès des tests qui seront exécutés lors du *Merge Request*.

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 1.

Question **9**

Correct

Note de 1,00 sur 1,00

Les tests du côté client utilisant les librairies Jasmine et Karma sont toujours exécutés dans un ordre aléatoire d'un lancement des tests à un autre. Pourquoi ?

- ( ) a. Ceci n'a aucun impact sur les tests exécutés et leurs résultats.
- ( ) b. Ceci permet d'exécuter les tests plus rapidement.
- ( ) c. Ceci est une limitation du fonctionnement de l'outil Karma.
- (•) d. Ceci permet de détecter la présence de dépendances entre les tests unitaires.✔

Votre réponse est correcte.

Question **10**

Terminé

Note de 3,00 sur 3,00

Voici le constructeur du component **PlayAreaComponent** qui utilise la classe **GameService** ainsi que la configuration de ses tests unitaires.

```
constructor(private readonly gameService: GameService) {}

// play-area.component.spec.ts
beforeEach(async () => {
   gameSpy = jasmine.createSpyObj('GameService', ['handleClick','blinkDifference','playSound']);
   TestBed.configureTestingModule({
       declarations: [PlayAreaComponent],
       providers: [{ provide: GameService, useValue: gameSpy }],
   }).compileComponents();
});
```

a) Est-ce que l'utilisation d'un SpyObj pourrait être nécessaire dans la configuration? **Justifiez** votre choix. (2 points)

b) Suite à l'exécution des tests de PlayAreaComponent, vous obtenez à plusieurs reprises le message d'erreur suivant dans la console : ERROR: 'NG0304: 'app-differences-area is not a known element:' .
Quelle est **la raison** de ce message ? (1 point)

a) Oui, l'utilisation d'un SpyObj pourrait être nécessaire dans la configuration. Ainsi on pourrait utiliser mock afin de se débarasser des dépendances entre ses tests. On veut tester uniquement le comportement de PlayAreaComponent ainsi que ses appels aux méthodes de GameService sans pour autant toutes les tester ce qui est possible avec l'utilisation de SpyObj.

b) Cela moi il s'agit d'un problème d'import, il se peut que app-differences-area soit utilisé par PlayAreaComponent, et si c'est le cas alors il faudrait l'importer et le déclarer par la suite dans declarations.

Commentaire :

Question **11**

Terminé

Note de 4,00 sur 4,00

Vous êtes responsable de faire la revue de code pour la fonctionnalité de dessin avec crayon pour le Sprint 2 d'un autre membre de votre équipe.

Voici un extrait du code à revoir. Identifiez les problèmes de qualité avec ce code.

Vous pouvez assumer que le code est fonctionnel et que les fonctions et classes référencées existent.

```
1  const TRUE = true;
2  const FALSE = false;
3  const ONE = 1;
4  const noir = '#000000';
5
6  @Injectable({ providedIn: 'root'})
7  export class PencilService {
8      canRedo: boolean = FALSE;
9      cantUndo: boolean = TRUE;
10     public fill = TRUE;
11     public circleDiameter: number = ONE;
12     Color: any = noir;
13     mouseDown: boolean = false;
14
15     constructor(){ this.drawingService = new DrawingService("crayon") }
16     // obtenir la taille
17     getColor() { return this.Color }
18     setColor(c: string) { this.Color = c }
19
20     onMouseMove(event: MouseEvent): void {
21         if(this.mouseDown){
22             if(this.isOnCanvas(event)){
23                 this.draw();
24             }
25         }else{
26             return;
27         }
28     }
29
30     private isOnCanvas(event): boolean{
31         if(this.drawingService.isOnCanvas(event.offsetX, event.offsetY))
32             return true;
33         else
34             return false;
35     }
36 ...
37 }
```

- Utilisation du francais et de l'anglais, le code est majoritairement écrit en anglais or on a une constante qui s'appelle noir, il aurait fallut écrire black

- on assigne une couleur noir qui a deja été déclaré avant a Color, il aurait été préférable de faire Color: any = '#000000'

- setColor prend pour parametre une variable c or on ne sait pas ce que cela veut dire, ainsi il faudrait changer le nom de la variable pour que cela soit plus instructif

- pour la méthode onMouseMove(), il n'est pas nécessaire de laisser les lignes 25, 26 et 27. Ainsi a supprimer

- Déclaration des constantes TRUE et FALSE qui ne sont jamais utilisé donc supprimer ces constant

- Déclaration d'une constante ONE qui équivaut au chiffre 1 et 1 n'est pas un nombre magique donc faire directement

```
public circleDiameter: number = 1
- fill et circleDiameter sont ici mis en attribut public mais préférable de les mettre en privé.
```

Commentaire :

Question **12**

Terminé

Note de 1,50 sur 2,00

Lorsqu'on crée une demande fusion (*Merge Request*) sur GitLab, il est possible de bloquer la fusion tant qu'un pipeline qui lui est associé n'est pas exécuté au complet sans erreurs avec l'option "*Pipelines must succeed*".

Donnez un avantage et un désavantage de l'utilisation de cette option sur votre processus de travail. Justifiez brièvement votre réponse.

L'avantage a cela est de savoir si dans le code que l'on veut merge il y a des erreur lint mais également s'il y a des tests qui échouent.

L'inconvénient de cette méthode est que attendre la fin d'une pipeline prend beaucoup de temps alors que quelque fois le code que l'on veut merge est uniquement du CSS et donc ne nécessite pas spécialement autant de temps.

Commentaire :
Ce n'est pas un avantage de l'option Pipelines must succeed. Votre pipeline va rouler dès qu'il y a un MR. Même sans l'option, vous allez savoir s'il y a des erreurs de lint etc.

Question **13**

Incorrect

Note de 0,00 sur 1,00

L'étape "install" de votre pipeline automatisée sur GitLab exécute la commande **npm ci**. Pourquoi utiliser cette commande plutôt que **npm install** ?

a. **npm ci** s'assure d'installer les versions exactes des dépendances du projet et toujours reproduire le même environnement, ce qui n'est pas garanti avec **npm install**.

b. **npm ci** veut dire **C**onsole **I**nstall et est simplement un alias de **npm install**.

c. Il y a une erreur dans la configuration du pipeline : **npm install** devrait être utilisée plutôt que **npm ci**. ✖

d. **npm ci** veut dire **C**ontinuous **I**ntegration et doit donc être utilisé dans un processus automatisé.

e. Il n'y a pas de différence entre les commandes et les 2 peuvent être utilisées de manière interchangeable.

Votre réponse est incorrecte.

◄ Annonces

Aller à...

Introduction ►