Commencé le	mardi 8 mars 2022, 18:30
État	Terminé
Terminé le	mardi 8 mars 2022, 20:29
Temps mis	1 heure 59 min
Note	**19,47** sur 25,00 (**78**%)

Description

LISEZ CES INSTRUCTIONS AVANT DE COMMENCER L'EXAMEN

- L'examen possède 11 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**

Terminer

Note de 2,00 sur 4,00

Vous êtes en charge du réusinage du chevalet ainsi que la fonctionnalité de manipulation de lettres au début du Sprint 2. Chaque tuile est représentée par l'interface suivante :

```
interface Tile {
    letter: string,
    value: number
}
```

Vous avez décidé de représenter chaque tuile sur votre chevalet par un Component : **TileComponent**. Voici l'implémentation du Component et son gabarait :

```
@Component({
  selector: 'app-tile',
  templateUrl: './tile.component.html',
  styleUrls: ['./tile.component.css']
})
export class TileComponent{
  @Input() tile : Tile;
}
```

Gabarit HTML :

```
<div id="container">
  <span id="letter">{{ tile.letter }}</span>
  <span id="value" >{{ tile.value }} </span>
</div>
```

Afin de prototyper rapidement le déplacement d'une lettre, vous avez implémenté les fonctions "*moveLeft*" et "*moveRight*" dans votre **RackService.** Assumez que le code est fonctionnel. Vous avez également la fonction "*getRack*" qui retourne votre chevalet. Voici une partie du service :

```
export class RackService {

  private tileRack: Tile[];
  getRack(): Tile[] { return this.tileRack; }

  moveRight() { ... }
  moveLeft() { ... }

}
```

Votre chevalet est représenté par **TileRackComponent**. Voici le Component et le gabarit que vous devez compléter :

```
export class TileRackComponent {

  tileRack: Tile[] = [];
  constructor(private rackService: RackService) {
    this.tileRack = this.rackService.getRack();
  }

  moveLeft() { this.rackService.moveLeft() }
  moveRight() { this.rackService.moveRight() }
}
```

Gabarit à compléter :

```
<!-- À COMPLÉTER -->
<div id="container">




</div>

<button (click)="moveLeft()">Gauche</button>
<button (click)="moveRight()">Droite</button>
```

a) Complétez le gabarit de **TileRackComponent** pour pouvoir afficher votre chevalet et pouvoir prototyper le déplacement. (2 points)

b) La valeur de tileRack est assignée seulement une fois dans le *constructeur* de **TileRackComponent** et nulle part ailleurs. Est-ce que l'affichage de votre chevalet sera mis à jour si l'objet tileRack de **RackService** est modifié par la suite à travers les fonctions "*moveLeft*" et "*moveRight*" ? Justifiez votre réponse. (2 points)

a)

```
<div id="container">
```

```
   <div [tileRack]="tileRack" *ngFor="let tile of tileRack">
```

```
     <app-tile [tile]="tile">
```

```
     </app-tile>
```

```
   </div>
```

```
</div>

<button (click)="moveLeft()">Gauche</button>
<button (click)="moveRight()">Droite</button>
```

b) On utilise la syntaxe [property] = "valeur" pour afficher les tile de tileRack. Ceci signifie que si on a une modification de tileRack dans le Component (fichier .ts), elle sera communiquer au gabarit HTML.

Ainis, quand on a un appel a moveLeft() ou moveRight() qui modifie tileRack, ce sera répercuté dans le fichier HTML.

Commentaire :
a) <div [tileRack]="tileRack" *ngFor="let tile of tileRack"> n'est pas nécessaire et n'a pas d'attribut tileRack à modifier -0.5

b) On modifie tileRack dans le Service. La question est si tileRack du Component sera modifié : ceci ne répond pas à la question -1.5

Question **3**

Partiellement correct

Note de 0,67 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 ?

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

Votre réponse est partiellement correcte.

Vous avez sélectionné trop d'options.
Les réponses correctes sont :
L'utilisation d'un Service élimine le besoin d'avoir un lien parent/enfant entre les Components.,

L'utilisation d'un Service diminue le couplage entre les Components.

Question **4**

Terminer

Note de 2,00 sur 3,00

Quelle est la différence entre le serveur lancé lorsqu'on fait **npm start** dans le dossier **client** et le serveur lancé par la même commande dans le dossier **serveur** de votre projet ? Quel est le rôle de chacun de ses 2 serveurs? **Justifiez** votre réponse.

Lorsque l'on lance le client, c'est le navigateur qui a pour rôle d'exécuter le code du client. Le code de la vue se trouve du côté client par exemple. Aussi, il peut y avoir plusieurs clients. Un client seul (non connecté au serveur) ne peut pas communiquer (avec le serveur donc avec les autres non plus).

Le serveur du dossier serveur est unique et peut être à distance comme lors du déploiement sur un serveur dynamique AWS. Le serveur communique avec les différents clients connectés et gère également la communication entre les clients via les webSockets qui passent par lui. Il peut aussi communiquer avec une base de donnée distante. Généralement, la validation de certaines fonctionnalités se font sur le serveur pour des raison de sécurité (comme pour la validation des mots dans le cas du projet2).

Commentaire : Vous mélangez client et serveur : dans le dossier **client**, on lance un **serveur statique** qui nous fournit les fichiers exécutés par le navigateur (le client)

Question **5**

Correct

Note de 1,00 sur 1,00

Vous êtes responsables de l'implémentation de la logique du joueur virtuel pour le Sprint 2 et vous avez trouvé un service en ligne qui permet de générer des anagrammes. Vous voulez l'utiliser pour générer les placements possibles pour votre joueur virtuel. Cependant, le serveur du service n'utilise pas Socket.IO, mais plutôt l'implémentation native de WebSocket pour sa communication.

Pouvez-vous quand même utiliser ce service dans votre projet dans son état actuel ?

a. Non, votre serveur ne peut pas communiquer avec un autre serveur, seulement avec des clients.

b. Oui, mais vous devez contacter le service à travers votre client puisque WebSocket est un protocole client-serveur.

c. Non, le client et le serveur doivent utiliser Socket.IO pour une communication valide. ✔

d. Oui, Socket.IO utilise le protocole WebSocket pour communiquer, donc les 2 systèmes sont compatibles.

e. Oui, mais vous ne pouvez pas utiliser les fonctionnalités supplémentaires que Socket.IO offre par-dessus WebSocket.

Votre réponse est correcte.

La réponse correcte est :
Non, le client et le serveur doivent utiliser Socket.IO pour une communication valide.

Question **6**

Correct

Note de 1,00 sur 1,00

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

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

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

c. 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. ✔

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

Votre réponse est correcte.

La réponse correcte est :
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.

Question **7**

Terminer

Note de 3,00 sur 4,00

Voici l'implémentation de la gestion du clavaradge dans plusieurs salles de votre projet.
Vous avez présentement 3 clients qui communiquent avec votre serveur : Client1, Client2 et Client3.
Client1 et Client3 sont présentement dans une partie de jeu et dans la même *Room* ayant le nom "123".

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

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

Client1 vient d'envoyer l'événement "chatMessage" avec l'objet **{ author: "Client1", message: "Allo"}** au serveur.
a) En fonction de la configuration donnée, qui recevera un message du serveur et **pourquoi** ?

b) Cette implémentation est fonctionnelle, mais n'est pas optimale. Donnez le problème avec l'implémentation et proposez une solution possible.

a) Le Client2 et Client3 recevront ce message, mais seul le Client3 le conservera.

En effet, le Client1 est émetteur du message, donc quand le serveur reçoit son événement de type 'chatMessage', le serveur lance un broadcast, c'est-à-dire qu'il envoie le chatMessage à tous les clients connectés sauf le client émetteur.

Lors de la réception (côté client) de cet événement de type 'chatMessage', les Client2 et Client3 vont vérifier si le message leur est destiné (`this.roomId === chatMessage.room`) et seul le Client3 qui se trouve dans la bonne room va push le chatMessage.

b) Le problème est que les clients en dehors de la partie reçoivent aussi le message. Il y a donc un envoi inutile d'information et une faiblesse au niveau de la sécurité.

Il faudrait faire la vérification côté serveur, en envoyant à tous les clients de la room sauf l'émetteur via un

this.sio.to(this.room).emit("chatMessage", `${socket.id} : ${chatMessage}`)

plutôt qu'un

socket.broadcast.emit("chatMessage", chatMessage);

et modifier en conséquent le code côté client.

Commentaire :
a) OK

b) Ceci modifie le format envoyé au client, ce qui peut être problématique. chatMessage est un objet et non une string : on obtiendra alors '[Object object]' -1

Question **8**

Terminer

Note de 3,00 sur 3,00

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

```
constructor(private readonly gridService: GridService) {}

// play-area.component.spec.ts
beforeEach(async () => {
   gridSpy = jasmine.createSpyObj('GridService', ['placeLetter','drawGrid','changeSize']);
   TestBed.configureTestingModule({
       declarations: [PlayAreaComponent],
       providers: [{ provide: GridService, useValue: gridSpy }],
   }).compileComponents();
});
```

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

b) Sachant qu'aucune erreur n'est lancée dans la console pour ce Component et en vous basant sur l'objet **declarations** du module de tests, que pouvez-vous déduire du contenu du gabarit de PlayAreaComponent ?

a) Effectivement c'est une bonne idée d'utiliser un SpyObj. Ainsi, on utilise un mock et on se débarrasse des dépendance pour ces tests. On ne souhaite tester que le comportement du component et sans doute ses appels aux méthodes de GridService, mais on ne veut pas tester les méthodes du service GridService. Sans ce spy, on risque d'avoir des problèmes.

b) On peut en conclure que le component PlayAreaComponent ne contient pas d'autres components (pas comme TileRackComponent qui comportait TileComponent dans la question2). Il n'y a pas d'autres particularités.

Commentaire :
a) Excellent

b) Parfait!

Question **9**

Correct

Note de 1,00 sur 1,00

Vous venez de terminer l'implémentation de l'intégration de la communication par sockets sur votre site web et vous êtes en mesure d'échanger des messages avec votre serveur.

Cependant, lors de l'exécution de vos tests, vous avez parfois un problème avec les tests des sockets qui n'arrivent pas à faire une connexion valide avec un serveur. Cette erreur se présente des fois lorsque vous lancez les tests sur votre machine, mais arrive toujours lorsque les tests sont exécutés par le pipeline automatisé de GitLab.

Quelle est la source d'erreur la plus probable dans cette situation ?

a. Le protocole WebSocket a besoin d'une connexion réseau pour fonctionner, ce qui n'est pas le cas sur la machine de pipeline.

b. Les tests font des appels vers un vrai serveur Socket.IO. Ils réussissent lorsque vous laissez votre serveur local ouvert sur votre machine et échouent sur GitLab qui n'a pas de serveur de disponible. ✔

c. Les tests font des appels vers un vrai serveur Socket.IO. La machine qui exécute les tests sur GitLab n'a pas la même addresse que votre machine locale et la connexion échoue toujours.

d. WebSocket est un protocole instable qui fonctionne mal sur Linux et GitLab utilise toujours des serveurs Linux pour exécuter les pipelines.

Votre réponse est correcte.

La réponse correcte est :
Les tests font des appels vers un vrai serveur Socket.IO. Ils réussissent lorsque vous laissez votre serveur local ouvert sur votre machine et échouent sur GitLab qui n'a pas de serveur de disponible.

Question **10**

Terminer

Note de 3,00 sur 4,00

Le code qui suit contient plusieurs défauts. Modifiez-le afin de corriger tous les défauts. Assurez-vous que votre code final respecte les standards de qualité et ne contienne pas de mauvaises odeurs.

Vous pouvez créer autant de nouvelles fonctions que nécessaire. Si vous désirez ajouter de nouveaux noms, ou en modifier, choisissez des noms le plus adéquats possible selon votre compréhension du code.

Il se peut aussi que votre code final contienne encore des mauvaises odeurs que vous ne pouvez corriger par manque d'information sur le contexte du code. Dans ce cas, décrivez-les par un court texte.

```
replaceLetters(a: Letter): boolean {
   let correct: boolean = isValidItem(a) ? true : false;
   if (correct === true) {
       let newLetter = this.letterBank.obtenirAleatoire(a);
       this.letters.foreach((v, i) => {
           if (this.items[i] === a) { this.items[i] = newLetter; }
       });
       return true;
   }
   else if (correct === false) {
       return false;
   }
}
```

```
replaceLetter(letter: Letter): void {
   if (!isValidItem(letter)) return;

   this.letters.foreach((v, i) => {
      if (this.items[i] === letter) { this.items[i] = this.letterBank.getRandom(); }     //getRandom() n'a pas de raison d'avoir
un parametre, mais je ne connais pas le code...
   });
}
```

```
//replaceLetter() retourne bizarrement un boolean, je pense qu'il serait mieux d'avoir un void, mais je ne connais pas le code...
```

```
//Je pense qu'un .find() serait mieux qu'un foreach, mais je ne suis pas sûre car je ne me souviens plus de l'implémentation.
```

Voici les défauts qu'il fallait pointer dans le code (soit les corriger, soit les discuter dans le texte):

Expression booléenne (1 pt)
L'expression ternaire est complètement inutile et devrait être éliminée (0,5 pt). De plus, on n'a pas besoin de la variable correct (0,5). On doit appeler directement la méthode isValidItem().

Renommage des variables (1 pt)
Le paramètre "a" n'est assurément pas un bon nom (0,5 pt). Idem pour les paramètres "v" et "i" (0,5 pt)

Fonction obtenirAleatoire() (1 pt)
Elle devrait être en anglais comme les autres (0,5 pt). De plus, comme elle prend un paramètre, le nom de la fonction ne semble pas adéquat (0,5 pt). Soit on considère qu'il faut enlever le paramètre, soit on change son nom pour quelque chose comme exchangeLetter().

Attribut this.items (0,5 pt)
On voit pas très pourquoi il existe en parallèle avec this.letters, et en plus le nom n'est pas adéquat. Il semble que un des deux ne devrait pas exister. De plus, this.letters est un Map qu'on parcourt de manière non habituelle (par les valeurs plutôt que par les clés)

Logique du code (0,25)
On se demande pourquoi on change toutes les occurrences de la lettre par une même lettre. Ici, pas nécessaire de corriger l'erreur, mais on doit au moins pointer ce défaut potentiel. Pour corriger, soit on enlève la variable temporaire et on fait l'appel à chaque occurrence de la lettre trouvée, soit on modifie le code pour ne changer que la première occurrence de la lettre (mais on ne fait pas ça en mettant un break dans le forEach())

Pas de nouveau défaut introduit dans votre nouvelle version du code (0,25)

À noter qu'il y a d'autres défauts qui ont été exclus du barème de correction, mais qui pourraient amener à être moins sévère sur le reste si vous les avez identifiés, notamment:

- La méthode ne devrait pas retourner un booléen
- le nom isValidItem() n'est pas adéquat
- La validation de la lettre devrait être faite avant d'appeler la méthode

La méthode améliorée pourrait ressembler à ceci:

```
replaceLetter(letterToReplace: Letter): void {
   this.playerLetters[this.playerLetters.indexOf(letterToReplace)] = letterToReplace;
}
```

Commentaire :
renommage v et i -0,5

this.items -0,5

Question **11**

Partiellement correct

Note de 0,80 sur 1,00

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

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

Votre réponse est partiellement correcte.

Vous avez sélectionné trop d'options.
La réponse correcte est :
Pour s'assurer que le code est toujours fonctionnel après les changements.

Question **12**

Terminer

Note de 2,00 sur 2,00

Présentez deux (2) bonnes pratiques à suivre lorsque vous êtes responsables de faire la revue de code d'une demande fusion (*Merge Request*) sur GitLab.

- Relire le code et s'assurer d'une bonne qualité de code par la personne assignée (qui ne doit pas être l'auteur.trice du code)

- Mettre des commentaires (questions ou points d'amélioration) sur GitLab voire ajouter des modifications pour corriger les problèmes directement dans la branche source du Merge Request.

Commentaire :
Parfait!

◄ Annonces

Aller à...

Introduction ►