Commencé le	mercredi 4 octobre 2023, 18:00
État	Terminé
Terminé le	mercredi 4 octobre 2023, 21:00
Temps mis	2 heures 59 min
Points	14,00/24,00
Note	**14,58** sur 25,00 (**58,33**%)

Description

LISEZ CES INSTRUCTIONS AVANT DE COMMENCER L'EXAMEN

- L'examen est noté 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

Terminé

Note de 1,50 sur 6,00

a) Voici un extrait du code du gabarit HTML de votre **QuestionBuilderComponent** qui est utilisé dans la création d'un jeu-questionnaire :

```
<app-question *ngFor="let question of questions; let x = index" *ngIf="questions.length !== 0"
      [question]="question" [index]="x + 1" (click)="handleSelect($event)"></app-question>
```

Voici un extrait du constructeur de ce même component :

```
communicationService: CommunicationService;
questions : Question[];
constructor(){
  this.communicationService = new CommunicationService();
  ...
}
```

Identifiez au moins 2 problèmes présents dans ces extraits. Expliquez chaque problème et proposez une solution. Considérez que tout service ou component externé référé est fonctionnel. (4 points)

b) Voici la configuration des tests et un test unitaire de votre **AnswerComponent** qui soumet une réponse choisie au serveur.

Est-ce que cette approche est correcte ? Si oui, pourquoi ? Si non, que devriez-vous changer ? (2 points)

```
export class AnswerComponent {
  constructor(public socketService: SocketClientService) { }

  sendForValidation(answer) {
    this.socketService.send('validate', answer);
  }

}

describe('AnswerComponent', () => {
  let component: AnswerComponent;
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [AnswerComponent],
    }).compileComponents();
    const fixture = TestBed.createComponent(AnswerComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });


  it('should send an answer to the server with validate event', () => {
    const spy = spyOn(component.socketService, "send");
    const eventName = "validate";
    const testString = 'test';
    component.sendForValidation(testString);
    expect(spy).toHaveBeenCalledWith(eventName, testString);
  })
}
```

a) Le premier est problème est que l'on place deux directives structurelles sur le même élément <app-question>. Pour remédier à ce problème, il faudrait créer un autre élément HTML englobant et y appliquer la deuxième condition `*ngIf="questions.length !== 0`. L'autre problème se situe au niveau de cette portion de code *ngFor="let question of questions; let x = index". Si on veut itérer selon un index, on devrait plutôt faire *ngFor="let question of questions; index as x". Cette syntaxe n'a pas été présentée dans le cours. De plus, je ne comprends pas cette portion de code

```
[index]="x + 1", normalement on utilise ca pour faire du data binding, à moins que l'intention du developpeur soit qu'il fasse
réellement ceci,
je ne vois pas son importance parce que s'il pense que ceci va incrementer son index, il a mal compris et il devrait l'enlever parce
qu on fait deja
ca.

b) Oui, l'utilisation d'un spy est un bon choix. En effet, on utilise un mock, ce qui permet d'éliminer les dépendance au service.
On veut tester
juste le comportement de AnswerComponent et ses appels aux méthodes de socketService, mais on ne veut pas tester les méthodes du
service lui même
, donc on utilise un spy. Par conséquent, on peut dire effectivement que l'approche utilisé est une bonne approche.
```

Commentaire :

a) Il faut retirer *ngIf et pas le mettre sur un élément parent : *ngFor gère déjà les tableaux vides. let x = index est une syntaxe valide utilisée dans l'exemple d'observable présenté en classe -3

b) -1.5 : Non le test est mal configuré et avec cette configuration on utilise la vrai instance du SocketClientService. Faudrait isoler le component en creant un mock du service avec SpyObj et l'ajouter aux providers.

Question 3

Terminé

Note de 3,00 sur 6,00

a) Voici un extrait de la configuration de l'API REST de votre serveur. Vous avec la route **GET /game/:name** qui permet de récupérer un questionnaire à partir de son nom et **GET /game/history** qui permet de récupérer l'historique des parties jouées. Considérez que les méthodes de **gameService** sont correctement implémentées.

```
this.router.get('game/history', (req: Request, res: Response) => {
 res.json(this.gameService.getFullHistory());
});

this.router.get('/game/:name', (req: Request, res: Response) => {
  const game = this.gameService.getGame(req.params.name);
  if (game) {
    res.status(200).json(game);
  }
  else {
    res.status(404).send();
  }
});
```

Quel est le problème avec cette configuration ? Expliquez brièvement la cause du problème et une solution possible.

Donnez au moins 1 exemple où il est possible d'envoyer une requête, obtenir le code 200, mais obtenir le mauvais objet dans le corps de la réponse. (3 points)

b) Votre collègue travaille sur la validation du mot de passe pour accéder à la page d'administration. Il a décidé de faire l'évaluation du côté du serveur dynamique en envoyant le mot de passe à travers un événement WebSocket nommé "login". Si le mot de passe est bon, le serveur vous répond à travers l'événement "loginSuccess", sinon, à travers l'événement "loginFail" et dans les deux cas il n'y pas de données qui accompagnent l'événement. Est-ce que l'approche proposée est bonne ? Justifiez (3 points)

a)
Pour la première partie de la question, je vois que les URL dans les requetes ne sont pas ce qu'on devrait avoir, rappelons que c est un API REST donc il y a des règles à respecter, l'URL devrait être /:name et l'autre devrait être /:history. De plus, moi j'aurai fait une remarque concernant les codes de retours, on sous entend que les deux seuls types de status qu'on peut avoir sont 200 et 404. Il n y a pas de try catch pour catch les erreurs liées au serveur avec un statut 500.

On pourrait obtenir le code 200 qui indique un SUCCESS , mais avoir le mauvais objet dans le corps de réponse s'il existe deux ou plusieurs objets qui possède exactement le même nom lorsqu'on effectue la requête, ce qui pourrait causé bien évidemment qu'on reçoive un autre objet possédant le même nom mais qui est différent des autres.

b) Non, je pense qu'en termes d'expérience utilisateur cette manière de procéder n'est pas correct. Supposons que l'administrateur de la page écrit son mot de passe et que celui-ci n'est pas correct. Normalement, la réponse sera à travers l'évènement loginFail, mais vu qu'aucune donnée n'est accompagné avec l'évenement, on ne pourra pas savoir la réelle cause du refus d'accès, on pourrait se demander est-ce réellement un problème lié au mot de passe saisi ou est-ce une erreur qui s'est produite au niveau du serveur interne, ce qui créerai une confusion donc j'estime que des données doivent accompagné l'évènement car éventuellement l'objectif d'un développeur est d'offrir une expérience utilisateur exempte de confusion.

Commentaire :
a) Vous faites l'hypothèse que le nom de partie n'est pas un identifiant unique, ce qui n'est pas précisé dans la question. Le point central dans cette question est le conflit de noms de route dans l'API, particulièrement lorsqu'un jeu est nommé "history". Dans ce cas, la requête GET /game/history retournera l'historique des jeux au lieu de la partie spécifique nommée "history". C'est un problème majeur qui doit être résolu pour assurer le bon fonctionnement de l'API. La réponse attendue mettait en évidence ce problème d'ambiguïté dans la conception des routes. Une solution envisageable serait par exemple de modifier le préfixe de l'un des deux gestionnaires de route. (-1.5)

b) Vous semblez être en accord avec le fait que la vérification est réalisée du côté serveur, ce qui est juste. Vous avez identifié un des problèmes à améliorer dans la réponse, mais ces lacunes découlent d'une erreur plus fondamentale que vous n'avez pas souligné. Le problème principal est l'utilisation des WebSockets pour une tâche qui serait mieux gérée avec une simple requête HTTP (offre une meilleure gestion des erreurs à travers des codes de retour différents). (-1.5)

Question 4

Correct

Note de 1,00 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. Il y a une erreur dans la configuration du pipeline : **npm install** devrait être utilisée plutôt que **npm ci**.

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

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

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

e. **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**. ✔

Votre réponse est correcte.

La réponse correcte est :
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**.

Question 5

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 utilisant 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 la librairie Express pour une gestion à plus haut niveau des requêtes HTTP et SocketIO est un protocole de communication bidirectionnelle.

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

e. 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. ✔

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 6

Correct

Note de 1,00 sur 1,00

Sachant que vous ne permettez pas la mise à jour d'information avec une ressource si son attribut *timeStamp* indiquant l'heure de modification est plus ancien que celui de la ressource déjà sur le serveur, quel est le code de réponse le plus sémantiquement approprié dans un tel cas?

- a. 403 Forbidden ✔
- b. 500 Internal Server Error
- c. 400 Bad Request
- d. 200 OK
- e. 404 Not Found
- f. 409 Conflict

Votre réponse est correcte.

Les réponses correctes sont :
403 Forbidden,

409 Conflict

Question 7

Incorrect

Note de 0,00 sur 1,00

Qui devrait faire l'intégration du code d'une demande de fusion (Merge Request) dans la branche cible ?

- a. La personne ayant fait la revue du code de la demande ✖
- b. La personne n'ayant pas participé au code de la branche source
- c. La personne désignée par le processus interne de l'équipe
- d. La personne ayant écrit la majorité du code dans la branche source

Votre réponse est incorrecte.

La réponse correcte est :
La personne désignée par le processus interne de l'équipe

Question 8

Correct

Note de 1,00 sur 1,00

Quel est l'objectif principal d'un *stand up* dans l'approche Scrum ?

- [ ] a. Revoir et mettre à jour le calendrier du projet
- [x] b. Identifier des obstacles au travail des membres de l'équipe ✔
- [ ] c. Assigner des tâches aux membres de l'équipe
- [ ] d. Faire une revue de code rapide des nouvelles fonctionnalités

Votre réponse est correcte.

La réponse correcte est :
Identifier des obstacles au travail des membres de l'équipe

Question 9

Partiellement correct

Note de 0,50 sur 1,00

Vous recevez l'erreur **'NG0304: 'app-info' is not a known element'** lorsque vous exécutez les tests unitaires de votre component app-parent. Sachant que la balise <app-info> est utilisée dans le gabarit du component testé, quelle serait la meilleure solution pour corriger l'erreur ?

- a. Le component app-info possède un défaut dans son code HTML et le défaut devrait être corrigé en premier.
- b. Le component app-info devrait être ajouté dans le tableau declarations de la configuration du TestBed. ☑
- c. Un spy sur le component app-info devrait être ajouté dans le tableau declarations de la configuration du TestBed.
- d. Un mock du component app-info devrait être ajouté dans le tableau declarations de la configuration du TestBed.

Votre réponse est partiellement correcte.

La réponse correcte est :
Un mock du component app-info devrait être ajouté dans le tableau declarations de la configuration du TestBed.

Question 10

Correct

Note de 1,00 sur 1,00

Combien de serveurs sont utilisés dans votre projet, excluant la base de données ?

a. 1 serveur statique et 1 serveur dynamique : un qui sert le site web et qui communique avec le serveur dynamique et un responsable de la communication réseau entre les clients.

b. 2 serveurs dynamiques : un qui sert le site web et un qui est responsable de la communication réseau entre les clients.

c. 2 serveurs statiques : un qui sert le site web et un qui est responsable de la communication réseau entre les clients.

d. 1 serveur statique et 1 serveur dynamique : un qui sert le site web et un responsable de la communication réseau entre les clients. ✔

e. 1 serveur statique et 2 serveurs dynamiques : un qui sert le site web, un qui est responsable de la communication HTTP et un qui est responsable de la communication avec la base de données.

Votre réponse est correcte.

La réponse correcte est :
1 serveur statique et 1 serveur dynamique : un qui sert le site web et un responsable de la communication réseau entre les clients.

Question 11

Terminé

Note de 2,00 sur 3,00

Voici le code de la fonction **submitChoices** qui permet de soumettre les choix de réponse pour validation dans une partie :

```
submitChoices(questionType) {
  if (questionType === QTYPE_QCM) {
    let answers = [];
    if (this.answers[0].isChecked == true) answers.push(1);
    if (this.answers[1].isChecked == true) answers.push(2);
    if (this.answers[2].isChecked == true) answers.push(3);
    if (this.answers[3].isChecked == true) answers.push(4);
    this.answerService.submitForValidation(answers);
  }
}
```

Un de vos coéquipiers considère que cette fonction a des problèmes de qualité de code et propose la solution suivante :

```
submitChoices(questionType) {
  if (questionType === QTYPE_QCM) {
    const answers = this.answers.map((answer, i) => {
      if (answer.isChecked)
        return i;
    }
    );
    this.answerService.submitForValidation(answers);
  }
}
```

a) Quels sont les problèmes de qualité de code à corriger dans l'implémentation initiale ? (1.5 points)

b) Est-ce que la solution de votre collègue est acceptable comme remplacement ? Justifiez votre réponse (1.5 points)

a) Premièrement, on un problème au niveau de la duplication de code. Les conditions if sont répétés pour chaque index de answers, au lieu qu'il y est un mécanisme d'itération comme une boucle for ou un mapping par exemple. Deuxièmement, if(`this.answers[0].isChecked == true`) peut être remplacé par if(this.answers[0].isChecked). La présence d'un == true n'est pas nécessaire. En plus, answers comme nom de variable n est pas très intuitif à première vue on aurait pu utilisé selectedChoices. On utilise QTYPE_QCM on aurait pu juste ecrire QCM parce que y a de la redondance on sait deja qu on parle de questionType dans la condition if. J'aurai egalement dit le nom submitForValidation pourait etre remplacer par validate(selectedChoices).

b) Je ne vois pas que c'est suffisant. Par exemple, on utilise i alors qu'on devrait donner un nom plus représentatif comme choiceIndex. On aurait pu mettre à la place de answer comme nom de clé : choice. Au lieu de answers on aurait mis selectedChoices. Encore une fois le QTYPE_ n'est pas necessaire dans QTYPE_QCM, QCM est suffisant.

Commentaire :
b) -1 n'a pas parlé du bug

Question 12

Correct

Note de 2,00 sur 2,00

Voici l'implémentation de votre **ChronometerService** qui gère une minuterie globale qui affiche le temps total passé dans la partie. Ses 2 méthodes sont appelées au début et à la fin de chaque partie et l'attribut *nbElapsedSeconds* est affiché à l'écran dans votre **GamePageComponent**. La fonction *interval* est un Observable qui émet à chaque seconde (1000ms).

```
@Injectable({ providedIn: 'root'})
export class ChronometerService {
   public nbElapsedSeconds: number = 0;
   private chronometer: Subscription = new Subscription();
   public startChronometer(): void {
       this.chronometer = interval(N_MS_IN_SECOND).subscribe(() => {
           this.nbElapsedSeconds++;
       });
   }
   public stopChronometer(): number {
       if (this.chronometer) this.chronometer.unsubscribe();
       return this.nbElapsedSeconds;
   }
}
```

Quelle sera la valeur initiale du temps affichée lorsque vous débutez une 2e partie après avoir complété une partie ayant pris 300 secondes (5 minutes) ?

a. 300 secondes puisque l'instance de ChronometerService est toujours la même ✔

b. 0 secondes puisque nbElapsedSeconds est initialisé à 0 à chaque fois

c. undefined puisqu'il y un défaut dans stopChronometer()

d. 300 secondes + le temps écoulé entre 2 parties puisque if(this.chronometer) retourne toujours false et on ne se désabonne jamais de l'observable.

e. Aucune de ses réponses

Votre réponse est correcte.

La réponse correcte est :
300 secondes puisque l'instance de ChronometerService est toujours la même