

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 :

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

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 à compléter :

```
export class TileRackComponent {  
  
    tileRack: Tile[] = [];  
    constructor(private rackService: RackService) {  
        this.tileRack = this.rackService.getRack();  
    }  
  
    moveLeft() { this.rackService.moveLeft() }  
    moveRight() { this.rackService.moveRight() }  
}
```

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

```
<div id="container">  
    <app-tile *ngFor="let tile of tileRack"  
    [tile]="tile"></app-tile>  
</div>
```

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" ?

Non, l'affichage de votre chevalet ne sera pas mis à jour si l'objet tileRack de RackService est modifié par la suite à travers les fonctions "moveLeft" et "moveRight". Cela est dû au fait que la directive *ngFor ne détecte pas automatiquement les modifications apportées à l'objet tileRack. Pour résoudre ce problème, vous pouvez utiliser un Subject de RxJS pour détecter les modifications apportées à l'objet tileRack.

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. @Input/@Output permettent le passage de valeurs primitives, contrairement aux Services qui permettent la communication d'objets complexes.

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. Les Services sont des singletons, contrairement aux Components.

e. L'utilisation d'un Service élimine le besoin d'avoir un lien parent/enfant entre les Components. ✓

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.

1. Serveur client (front-end) : Lorsque vous exécutez npm start dans le dossier client, cela lance généralement un serveur de développement pour le front-end de votre application. Ce serveur sert les fichiers statiques (HTML, CSS, JavaScript) qui composent l'interface utilisateur de votre application. Le serveur statique fournit des fichiers aux clients (navigateur), mais ne fait aucune gestion de navigation de page ou logique d'application. Le client ne peut pas "offrir" des fichiers, mais seulement les récupérer. 2. Serveur d'application (back-end) : D'autre part, lorsque vous exécutez npm start dans le dossier serveur, cela démarre le serveur d'application back-end. Ce serveur gère la logique métier de votre application, comme le traitement des requêtes HTTP, l'interaction avec des bases de données, l'authentification des utilisateurs, etc. Il renvoie des données (généralement sous forme de JSON) en réponse aux requêtes du client. Ces deux serveurs travaillent ensemble pour fournir une application web complète. Ils sont souvent séparés pour respecter le principe de séparation des préoccupations, ce qui rend l'application plus facile à maintenir et à développer.

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

a. 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. ✓

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

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

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

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 adresse 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.

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

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.

c. Pour s'assurer que le code est toujours fonctionnel après les changements. ✓

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.

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

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. Oui, Socket.IO utilise le protocole WebSocket pour communiquer, donc les 2 systèmes sont compatibles.
- b. Oui, mais vous ne pouvez pas utiliser les fonctionnalités supplémentaires que Socket.IO offre par-dessus WebSocket.
- c. Non, votre serveur ne peut pas communiquer avec un autre serveur, seulement avec des clients.
- d. Non, le client et le serveur doivent utiliser Socket.IO pour une communication valide. ✓
- e. Oui, mais vous devez contacter le service à travers votre client puisque WebSocket est un protocole client-serveur.

Voici l'implémentation de la gestion du clavardage 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 cette configuration, qui recevra un message du serveur et pourquoi ?
- b) 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) Cette implémentation est fonctionnelle, mais n'est pas optimale. Donnez le problème avec l'implémentation et proposez une solution possible.

Le problème avec cette implémentation est qu'elle n'est pas efficace en termes de bande passante et de performance. Actuellement, chaque message est envoyé à tous les clients, et chaque client doit ensuite vérifier si le message est destiné à sa salle. Cela signifie que beaucoup de messages sont envoyés inutilement, ce qui peut ralentir l'application et consommer beaucoup de bande passante. Une solution possible serait d'utiliser les fonctionnalités de salle de Socket.IO pour envoyer des messages uniquement aux clients qui sont dans la même salle. Voici comment vous pouvez le faire :

```
// Côté serveur  
socket.on('chatMessage', (message) => {  
    const room = this.getRoomFromSocketId(socket.id);  
    const chatMessage = { ...message, room: room }  
    socket.to(room).emit("chatMessage", chatMessage);  
});  
  
this.socketService.on('chatMessage', (chatMessage) => {  
    this.chatMessages.push(chatMessage);  
});
```

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 ?

Effectivement c'est une bonne idée d'utiliser un SpyObj. Ainsi, on utilise un mock et on se débarrasse des dépendances 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) 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 ?

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.

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 newsletter = this.letterBank.obtenirAleatoire(a);  
        this.letters.forEach((v, i) => {  
            if (this.items[i] === a) { this.items[i] = newsletter; }  
        });  
        return true;  
    } else if (correct === false) {  
        return false;  
    }  
}
```

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);

A 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;  
}
```

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 ?

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

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

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

a) Votre collègue vous demande pourquoi est-ce qu'il y a 2 éléments <div> 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 ?
On ne peut pas placer 2 directives structurelles sur le même élément, *ngFor peut aller directement sur <app-game-card>. Les 2 divs ne sont pas nécessaires.

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

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 ?

- Ceci n'a aucun impact sur les tests exécutés et leurs résultats.
- Ceci permet d'exécuter les tests plus rapidement.
- Ceci est une limitation du fonctionnement de l'outil Karma.

d) Ceci permet de détecter la présence de dépendances entre les tests unitaires. ✓

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.

a) Oui, l'utilisation d'un SpyObj pourrait être nécessaire dans la configuration. Ainsi on pourrait utiliser mock afin de se débarrasser 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) 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 ?

Selon 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.

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.

```
const TRUE = true;
const FALSE = false;
const ONE = 1;
const none = '#000000';

@Injectable({ providedIn: 'root' })
export class PencilService {
  canRedo: boolean = FALSE;
  canUndo: boolean = TRUE;
  public fill: TRUE;
  public circleDiameter: number = ONE;
  Color: any = none;
  mouseDown: boolean = FALSE;

  @Injectable()
  constructor() { this.drawingService = new DrawingService("crayon") }

  // obtener la talle
  getColor() { return this.color }
  setColor(c: string) { this.color = c }
```

- Utilisation du français 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 déjà 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.

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.

L'avantage de l'option "Pipelines must succeed" est qu'elle assure que le code ne sera fusionné que si tous les tests passent, ce qui contribue à maintenir la qualité du code. L'inconvénient est que cela peut ralentir le processus de fusion, surtout si les pipelines sont longs, même pour des changements qui n'affectent pas la logique du code, comme les modifications de CSS.

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 Console Install 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 Continuous Integration 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. Voici l'implémentation de votre ChronometerService qui gère la minuteuse dans vos parties. 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). Quelle sera la valeur initiale du temps affichée lorsque vous débutez une 2e partie après avoir complété une partie avant pris 30 secondes ?

```
@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;
  }
}
```

Parmi les énoncés suivants, lesquels sont erronées ?

- l'utilisation de Services est obligatoire dans un projet Angular. ✓
- l'utilisation d'un Service élimine le besoin d'avoir un lien parent/enfant entre les Components. ✓
- Les décorateurs @Input/@Output permettent le passage de valeurs primitives seulement, contrairement aux Services qui permettent la communication d'objets complexes. ✓
- La syntaxe [[maVariable]] est la seule manière de faire une liaison bi-directionnelle entre un Gabarit et le code TS de son Component.
- Un Component est identifié seulement par son attribut 'selector' dans le gabarit HTML d'un autre Component.
- La directive *ngIf ne peut pas être placée sur le même élément HTML que la directive *ngFor. La création d'une nouvelle fiche de jeu dans votre système se fait à travers une requête POST sur la route /games/ et les informations de la fiche dans le corps de la requête. Vous avez décidé de ne pas permettre la création de 2 fiches avec le même nom. Lorsqu'un tel cas arrive, le code de retour dans la réponse HTTP est : 404 Not Found. Sémantiquement, quel est le meilleur code de réponse dans une telle situation ?
- 500 Internal Server Error
- 204 No Content
- 409 Conflict ✓
- 404 Not Found
- Aucune de ses réponses
- 400 Bad Request

Vous êtes responsables de l'implémentation de la logique du système de détection de différences entre 2 images pour le Sprint 1 et vous avez trouvé un service en ligne qui permet de générer les différences entre 2 images avec un rayon d'élargissement. Vous voulez l'utiliser pour générer les images de différences pour vos jeux. Votre équipe a déjà mis en place une communication utilisant Socket.IO pour la logique de jeu entre votre site web et votre serveur. Cependant, le serveur du service trouvé 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 ?

- Oui, mais vous ne pouvez pas utiliser les fonctionnalités supplémentaires que Socket.IO offre par-dessus WebSocket.
- Non, votre serveur ne peut pas communiquer avec un autre serveur, seulement avec des clients.
- Oui, mais vous devez établir une connexion supplémentaire en utilisant le protocole WebSocket seulement.
- Non, le client et le serveur doivent utiliser Socket.IO pour une communication valide.
- Oui, Socket.IO utilise le protocole WebSocket pour communiquer, donc les 2 systèmes sont compatibles.
- Oui, mais vous devez contacter le service à travers votre client puisque WebSocket est un protocole client-serveur.

Considérez le code suivant qui gère le téléversement et l'affichage d'une image dans la Vue de création.

```
async imageUpload(e: Event, index: number): Promise<void> {
  let f = (e.target as HTMLInputElement).files.item(0) as File;
  const check = !(this.imagesService.isNotValidImage(f));
  if (!check) {
    this.displayImageError();
    return;
  }
  switch (index) {
    case 0: {
      this.drawImage(this.originalCanvas.nativeElement, f);
      break;
    }
    case 1: {
      this.drawImage(this.modCnvs.nativeElement, f);
      break;
    }
    case 2: {
      this.drawImage(this.modCnvs.nativeElement, f);
      break;
    }
  }
}
```

a) Identifiez les erreurs de qualité de code présentes. Considérez que les autres fonctions appelées sont bien implémentées et ne causent pas de bogues.

Voici une liste d'erreurs de qualité de code :

- La conversion forcée de 'e.target' et du résultat de l'appel 'item(0)' sans vérification.
- La double négation de check lignes 4 et 5.
- La fonction négative isNotValidImage devrait s'appeler isValidImage et retourner si une image est valide. On peut inverser le résultat de l'appel si besoin.

- Le nom de la variable 'f' n'est pas descriptif de ce que la variable contient.
- Le nom de la variable 'check' n'est pas descriptif non plus.
- L'indentation des cas du switch n'est pas faite.
- Les accolades autour de chaque cas du switch sont inutiles.

b) Votre collègue considère que le switch/case présente un problème de duplication de code et propose le code suivant. Est-ce que vous considérez que sa solution est acceptable et améliore la qualité de votre code ?

```
if(index%2 === 0)
  this.drawImage(this.originalCanvas.nativeElement, f);
if(index == 2)
  this.drawImage(this.modCnvs.nativeElement, f);
```

Ce nouveau code n'est pas une meilleure solution car :

- Il ne fonctionne pas pour le cas où index est égal à 1.
- Le fait d'utiliser l'opérateur modulo ici diminue la lisibilité du code.
- Il faudrait remplacer le double égal par un triple égal pour valider que le type de index est vraiment number. Une fois qu'une demande de fusion (Merge Request) a été revue et approuvée par un membre de l'équipe, qui devrait être responsable de l'intégration du code dans la branche de destination ?

a. Seulement le membre d'équipe ayant implémenté la majorité du code.

b. Le ou les membre(s) désigné(s) par une convention quelconque dans l'équipe.

c. Aucune de ses réponses.

d. N'importe quel membre de l'équipe.

e. Seulement un membre de l'équipe qui n'est pas l'auteur ou l'évaluateur de la demande fusion.

f. Seulement le membre d'équipe ayant révisé la majorité du code

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 externe référé est fonctionnel.

- Il faut retirer *ngIf et pas le mettre sur un élément parent :

2. *ngFor gère déjà les tableaux vides donc même pas besoin du *ngIf

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 ? Non le test est mal configuré et avec cette configuration on utilise la vrai instance de SocketClientService. Faudrait isoler le component en créant un mock du service avec SpyObj et l'ajouter aux providers.

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

- Revoir et mettre à jour le calendrier du projet

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