



POLYTECHNIQUE
MONTRÉAL

Questionnaire examen final

LOG2990

Sigle du cours

Identification de l'étudiant(e)		
Nom : Kim	Prénom : Victor	
Signature :	Matricule : 1954607	Groupe :

Sigle et titre du cours			
LOG2990 - Projet de logiciel d'application Web			
Professeur		Groupe	Trimestre
Lévis Thériault		Tous	Automne 2020
Jour	Date	Durée	Heures
Samedi	19 décembre	2h30	9h30
Documentation		Calculatrice	Outils électroniques
<input type="checkbox"/> Aucune <input checked="" type="checkbox"/> Toute <input type="checkbox"/> Voir directives particulières		<input type="checkbox"/> Aucune <input checked="" type="checkbox"/> Toutes	Les appareils électroniques personnels sont interdits.
Directives particulières			
<ul style="list-style-type: none">Le professeur ne répondra à aucune question durant cet examen. Si vous estimez que vous ne pouvez pas répondre à une question pour diverses raisons, veuillez le justifier puis passer à la question suivante.Il est strictement interdit de débrocher l'examen.IMPORTANT : inscrire votre matricule sur toutes les pages numérotées.			
Cet examen contient <input type="text" value="8"/> questions sur un total de <input type="text" value="21"/> pages (incluant cette page).			
L'étudiant doit honorer l'engagement pris lors de la signature du code de conduite.			

Q1. Application client (26 points)

Ayant eu énormément de plaisir à développer PolyDessin, votre équipe décide de le pousser un peu plus loin en improvisant un sprint 4. Parmi vos nombreuses idées, vous décidez d'ajouter une fonctionnalité permettant des commentaires et des notes sur vos dessins.

Vous êtes responsable de l'affichage des commentaires et les notes laissées par les utilisateurs. Vous êtes encore dans le début de sprint et vous avez décidé de faire un prototype rapide pour la fonctionnalité en reprenant le code fait par un collègue qui a décidé de faire une autre fonctionnalité.

Chaque commentaire est représenté par l'interface suivante où "text" est le commentaire et "rating" est la note donnée par l'utilisateur. Un commentaire peut ne pas contenir de texte, mais seulement une note.

```
interface Comment {  
  text?: string;  
  rating: number;  
}
```

Vous avez le service CommentsService qui s'occupera éventuellement de la gestion des commentaires et de la communication avec un serveur à distance. Pour votre prototype, vous devez garder les commentaires locaux. Vous n'avez pas à modifier cette classe.

```
import { Injectable } from '@angular/core';  
import { Comment } from './classes/comment';  
  
const comments: Comment[] = [  
  { text: "Meilleur dessin au monde", rating: 10 },  
  { text: "", rating: 7.5 },  
  { text: "Vraiment pas impressionnant", rating: 3 },  
  { text: "WOW", rating: 8 },  
  { text: "", rating: 4 }];  
  
@Injectable({ providedIn: 'root' })  
export class CommentsService {  
  
  async getComments(): Promise<Comment[]> {  
    return comments;  
  }  
}
```

Vous avez le début d'implémentation de la fonctionnalité d'affichage. Vous devez compléter le code des 3 composantes suivantes : *CommentComponent*, *CommentsListComponent* et *AverageDisplayComponent*

Cacher les commentaires vides ☐

Meilleur dessin au monde	10/10
	7.5/10
Vraiment pas impressionnant	3/10
WOW	8/10
	4/10

Moyenne : 6.5	5 Commentaires
---------------	----------------

Figure 1 : Affichage de tous les commentaires

Cacher les commentaires vides ☒

Meilleur dessin au monde	10/10
Vraiment pas impressionnant	3/10
WOW	8/10

Moyenne : 7	3 Commentaires
-------------	----------------

Figure 2 : Affichage en cachant les commentaires vides

En voyant le peu de code dans *CommentsService*, un membre de votre équipe propose de tout simplement déplacer le code dans *CommentsListComponent* qui est le seul utilisateur de cette classe pour le reste du projet. Êtes-vous d'accord avec cette proposition? Si oui, pourquoi? Sinon, pourquoi? **(3 points)**

Non, car c'est mieux qu'on isole la partie vue et logique, pour que le service a juste comme rôle de communiquer avec le serveur, de valider l'entrée de l'utilisateur etc. Ça va être très lourd si il n'y a tout dans le component, de plus, angular se base sur l'injection de la dépendance.

CommentComponent (6 points)

Ce Component affiche le texte et la note de chaque commentaire. Il prend en paramètre le commentaire et un boolean "hidden". Si "hidden" a la valeur TRUE, le contenu du Component ne doit pas être inclus dans le DOM si le commentaire ne comporte pas de texte. Le dénominateur de la note doit être configuré à la création du Component et ne doit pas être modifiable pendant la durée de vie de l'application.

Vous devez compléter le gabarit et le code du Component pour répondre aux exigences.

comment.component.html

```
<!--À COMPLETER-->
<div *ngIf="shouldDisplay" _____ class="parent">
  <span id="comment">_____ {{ this.comment.text }} _____</span>
  <span id="rating">_____ {{ this.comment.rating }}/{{ MAX_VALUE }}</span>
</div>
```

comment.component.ts

```
@Component({
  selector: 'app-comment',
  templateUrl: './comment.component.html',
  styleUrls: ['./comment.component.css']
})
export class CommentComponent {

  @Input() public comment: Comment;
  @Input() public hidden: boolean = false;
  // À COMPLETER
  constructor(com:Comment, private display:boolean )
  { this.comment=com; this.hidden=display;}
  _____
```

```
get shouldDisplay() { // À COMPLETER
  if(this.hidden==true && this.comment.text!=" ") {
    return true;
  }
  else {
    return false;
  }
}
```

```

}
}
```

AverageDisplayComponent (5 points)

Ce Component affiche la moyenne des notes de chaque commentaire ainsi que le nombre total de commentaires. Il prend comme input un tableau de notes (`number[]`).

Vous devez compléter les fonctions *average()* et *total()* du Component qui calculent respectivement la moyenne et le nombre total de commentaires.

average-display.component.html

```
<div class="parent">
  <span id="average">Moyenne : {{ average }} </span>
  <span id="totalComments">{{ total }} Commentaires</span>
</div>
```

average-display.component.ts

```
@Component({
  selector: 'app-average-display',
  templateUrl: './average-display.component.html',
  styleUrls: ['./average-display.component.css']
})
export class AverageDisplayComponent {

  @Input() comments: number[] = [];

  get average(): number { // À COMPLÉTER

    const average=0;
    for(let i=0;i<comments.length;i++) {
      average+=comments[i].rating;
    }
    return average;

  }

  get total(): number { // À COMPLÉTER
    const nbComments=0;

    for(const comment:comments) {
      nbComments+=1;
    }
    return nbComments;
  }
}
```

CommentsListComponent (12 points)

Ce Component fait l'affichage complet des commentaires dans une liste. Chaque commentaire et sa note doivent être affichés dans une liste avec la note moyenne et le nombre total de commentaires laissés affichés à la fin de cette liste. Une case à cocher (checkbox) permet de cacher ou afficher les commentaires vides. Si la case est cochée, tous les commentaires sans texte sont cachés. La moyenne et le nombre de commentaires tiennent compte seulement des valeurs des commentaires visibles. Voir les figures 1 et 2 à la page 3 pour des exemples.

Vous devez compléter le gabarit et le code du Component. Notez que la liste de commentaires à afficher peut être d'une longueur arbitraire. Vous devez minimiser la duplication de code et utiliser les méthodes fournies / à compléter. Vous devez compléter la fonction *ngOnInit()* pour obtenir la liste des commentaires à partir de votre service. Vous devez compléter la fonction *commentGrades()* qui retourne les notes de tous les commentaires visibles.

comments-list.component.html

```
<div id="container" *ngIf="comments">
  <div id="input-container">
    <label for="input">Cacher les commentaires vides</label>
    <!--À COMPLETER-->
    <input id="input" type="checkbox" [checked]="this.handleClick($event)" />
  </div>
  <!--À COMPLETER-->
    <div *ngIf="this.hidden==false">
      <div *ngFor="let comment of comments">
        <li>{{comment.text}} {{ average }} {{ total }} </li>
      </div>
    </div>

</div>
```

comments-list.component.ts

```
@Component({
  selector: 'app-comments-list',
  templateUrl: './comments-list.component.html',
  styleUrls: ['./comments-list.component.css']
})
export class CommentsListComponent implements OnInit {

  comments: Comment[];
  hidden: boolean;

  constructor(private commentsService: CommentsService) {}

  ngOnInit() { // À COMPLÉTER

    async this.commentsService.getComments()
    .then((c:comments[])=>{
      this.comments=c;
    })
    .catch(()=>{
      console.log("failed to get comments");
    })

  }

  handleClick(event: MouseEvent) { this.hidden = !this.hidden; }

  get commentGrades(): number[] { // À COMPLÉTER

    grades:number[];
    for(const comment:comments) {
      if(comment.text!=" ") {
        grades.push(comment.rating);
      }
    }
    return grades;

  }

}
```

Q2. Serveur (12 points)

Ayant impeccablement terminé votre fonctionnalité sur le client, vous décidez de vous attaquer au code du serveur qui devra gérer les commentaires. Avant d'écrire le code, vous décidez de planifier l'architecture de votre API REST. Vous savez que votre serveur devra permettre les actions suivantes:

1. Obtenir tous les commentaires d'un dessin spécifique
2. Obtenir un commentaire spécifique
3. Ajouter un nouveau commentaire pour un dessin spécifique
4. Modifier un commentaire spécifique (son texte et/ou la note donnée)
5. Supprimer un commentaire spécifique
6. Obtenir tous les commentaires d'un auteur spécifique

Vous planifiez utiliser l'interface *Comment* de la Question 1 (voir ci-dessus). Devriez-vous modifier cette interface pour répondre aux exigences du serveur? Si oui, donnez la nouvelle interface, sinon justifiez pourquoi. (**3 points**)

Oui je vais modifier l'interface de *Comment*, car on va devoir identifier les commentaires sur le server pour faciliter les manipulations comme trouver un commentaire spécifique ou prendre tous les commentaires d'un auteur

```
interface Comment {  
    text?: string;  
    rating: number;  
    commentId:number;  
    author: number;  
}
```


Vous devez implémenter une route HTTP gérée par Express pour chaque action décrite ci-dessus. Pour chaque action, donnez les éléments suivants : **(9 points)**

- sa **route**
- son **verbe HTTP**
- le(s) **code(s) de retour HTTP**. Dans le cas de plusieurs codes, donnez le cas où chaque code sera retourné
- les **paramètres de la requête** (s'il y a lieu)
- le **corps de la requête** (s'il y a lieu)

```

app: express.Application;

constructor() {
  this.app=express();
  this.app.use(bodyParser.json());
}

comments:Comment[];

this.app.get("/comments",req:Request,res:Response)=> {
  res.send(comments);
}

this.app.get("comments/:author",req:Request,res:Response)=>{
  const toReturn:Comments[];
  for(comment of comments) {
    if(comments.author==ParseInt(req.params.author)) {
      toReturn.push(comment);
    }
  }
  if(toReturn.length==0) {
    res.send(404);
  }
  res.send(toReturn);
}

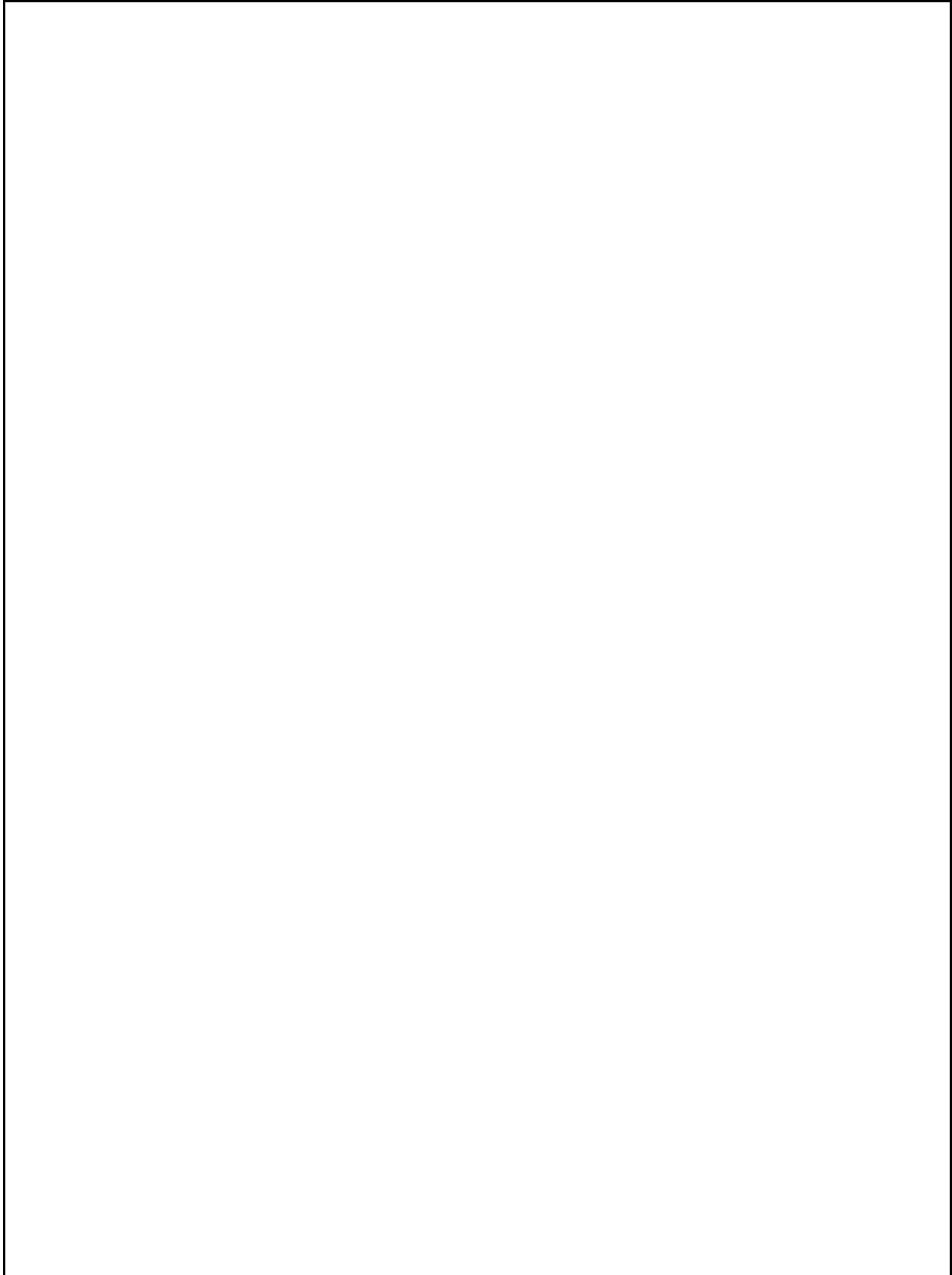
this.app.get("/comments/:id",req:Request,res:Response)=>{
  for(const comment of comments) {
    if(comment.id==parseInt(req.params.id)) {
      res.send(comment);
    }
  }
  res.send(404);
}

this.app.delete("/comments/:id",req:Request,res:Response)=>{
  for(let i=0;i<comments.length;i++) {
    if(comments[i].id==parseInt(req.params.id)) {
      comments.splice(i,1);
      res.send(201);
    }
  }
  res.send(404);
}

```

LOG2990

Matricule : _____

A large, empty rectangular box with a thin black border, occupying the central portion of the page. It is intended for a drawing or a detailed written response.

Q3. Canvas (6 points)

- a) Les pixels dans un Canvas sont représentés par un objet de type *Uint8ClampedArray* qui est un tableau en une dimension qui contient les informations de chaque pixel dans l'ordre RGBA avec des valeurs de 0 à 255. Lors de la transformation d'une image sur Canvas en JPEG, outre la compression du format JPEG, à quel autre changement peut-on s'attendre, contrairement à une transformation en PNG? **(3 points)**

Les images jpg sont plus grosse donc plus détaillé, pour jpeg on peut s'attendre à une diminution de la qualité des images.

- b) Vous avez un Canvas de taille `width * height`. Y a-t-il une différence entre appeler `getImageData(0, 0, width, height)` pour récupérer tous les pixels d'un Canvas et itérer à travers le tableau pour les manipuler et itérer à travers tous les pixels en faisant 2 boucles `for` et appeler `getImageData(i, j, 1, 1)`? Quelle méthode est préférable? **(3 points)**

Oui `getImageData(0,0,width,height)` permet de mettre tous les pixels à partir de l'origine et va manipuler un ensemble de pixels en même temps tandis que `getImageData(i,j,1,1)` va manipuler pixel à la fois. Je pense `getImageData(0,0,width,height)` est préférable, car on a pas à utiliser 2 boucles donc la complexité va diminuer.

Q4. Expérience utilisateur (6 points)

Votre équipe hésite entre l'utilisation d'un **glisseur** (*slider*) ou d'un **champ de saisie** (*input field*) pour la paramétrisation de l'épaisseur de vos bordures de formes pour avoir la meilleure expérience utilisateur. Choisissez **UNE** des deux options et discutez des avantages (2) et des désavantages (2) de choisir cette option. Justifiez chacun des éléments de votre réponse.

Un slider permet de limiter la valeur qu'on peut donner aux modifications, de plus il permet à l'utilisateur d'avoir une vue de la modification. Les désavantages c'est qu'il y a toujours une donc on ne peut pas mettre des valeurs qui dépassent la limite et c'est difficile de décider quoi mettre comme intervalle. De plus, quand l'intervalle est très grand, c'est difficile contrôler le slider.

Q5. Tests (16 points)

- a) Voici un extrait des tests unitaires pour la fonction **translatePoints**. En fonction des tests présentés, implémentez la fonction **translatePoints**. Vous pouvez assumer que la classe **Point** existe et possède les variables **x** et **y** de type **number**. **(5 points)**

```
let service: PointsService;
let points: Point[];
beforeEach(() => {
  TestBed.configureTestingModule({ providers: [PointsService] });
  points = [Point( 2, 3 ), Point( 3, 4 ), Point( 1, 1 )];
});

it ('Should return translated points by an x and y offset', () => {
  service = TestBed.get( PointsService );
  const expectedPoints: Point[] = [Point( 3, 5 ), Point( 4, 6 ), Point( 2, 3 )];
  expect(service.translatePoints( points, 1, 2 ) ).toEqual( expectedPoints );
});

it ('Should not translate points below 0', () => {
  service = TestBed.get( PointsService );
  const expectedPoints: Point[] = [Point( 0, 0 ), Point( 1, 1 ), Point( 0, 0 )];
  expect( service.translatePoints( points,-2,-3 ) ).toEqual( expectedPoints );
});
```

Réponse :

```
translatePoints(x:number,y:number) {
  this.pointService.translate(x,y);
}
```

Matricule : _____

- b) La fonction suivante ne possède pas encore de tests unitaires. Expliquez ce que vous feriez afin de la tester adéquatement. Vous pouvez fournir du code si vous le jugez pertinent. Vous pouvez supposer que la classe **Point** existe et possède les variables **x** et **y** de type **number**. (5 points)

```
public shrinkPoints(points: Point[], factor: number): Point[] {  
  shrunkPoints: Point[] = [];  
  
  for (let point of points) {  
    newX: number = point.x / factor;  
    newY: number = point.y / factor;  
    newPoint: Point = new Point(newX, newY);  
    shrunkPoints.push(newPoint);  
  }  
  
  return shrunkPoints;  
}
```

Réponse :

tester le cas d'un tableau vide,
un facteur de 0,
des données négatives.

Matricule : _____

- c) Expliquez brièvement quelle est la différence entre un "Mock" et un "Spy". Donnez un exemple d'utilisation pour chacun des concepts en lien avec votre projet. **(6 points)**

Un mock c'est un instance d'objet qui n'existe pas qu'on va utiliser pour faire comme si on avait un objet réel. Un Spy c'est quand on ne fait que observer un méthode qui être réellement appelé.

PS: je voulais dire spy appel réellement la méthode mais pas stub

Q6. Architecture (10 points)

Quel patron de conception avez-vous utilisé pour la fonctionnalité "**Annuler-Refaire**" du sprint 2? Si vous n'avez pas utilisé un patron de conception spécifique, lequel auriez-vous utilisé pour implémenter cette fonctionnalité?

- a) Dans les deux cas, expliquez le patron de conception et **justifiez** votre choix. (5 points)

Le design pattern pour que plusieurs hérite de la classe de base qu'on puisse accéder aux même fonctionnalité de base et des spécifications pour chaque classe dérivé.

- b) Expliquez sommairement la manière dont la fonctionnalité "**Annuler-Refaire**" a été implémentée dans votre projet. Vous pouvez donner du code en appui, si vous le jugez pertinent. (5 points)

Dans le projet chaque action est un classe dérivé de la classe de base et on a 2 pile un pour undo et l'autre pour redo qu'on va déplacer les objets selon les buttons et redo fait appel au fonction dans les services spécifiques.

Q7. Assurance Qualité (18 points)

Voici 2 extraits de code en lien avec votre projet. Pour chaque extrait, identifiez les problèmes de **qualité du code** et proposez une **amélioration**. Vous pouvez fournir du code en exemple, si vous le trouvez pertinent. Vous pouvez supposer que le code fourni est fonctionnel.

a) La fonction *handleKey* qui gère les raccourcis pour l'outil Texte. (6 points)

```
public handleKey(kc: number, i: number, j: number): boolean {  
    const letter = String.fromCharCode(kc);  
    switch (true) {  
        case this.isLetter(kc):  
            return this.handleLetter(letter, i, j);  
        case this.isBackspace(kc):  
            return this.handleOther(i, j);  
        case this.isArrow(kc):  
            return this.gererFleches(kc, i, j);  
        default:  
            return false;  
    }  
}
```

Réponse :

trop de cas devient lourd mieux de faire de l'héritage

b) La classe *SprayService* qui implémente le début de l'outil Aérosol. (12 points)

```
const TRUE = true;
const FALSE = false;
const ZERO = 0;
const FIVE = 5;
const DIX = 10;

@Inject({ providedIn: 'root'})
export class SprayService {
  canRedo: boolean = FALSE;
  cantUndo: boolean = TRUE;
  public fill = TRUE;
  public circleDiameter: number = ZERO;
  SprayFrequency: any = DIX;
  mouseDown: boolean = false;

  constructor(){ this.drawingService = new DrawingService() }

  getFrequency() { return this.SprayFrequency }
  setFrequency(f: number) { this.SprayFrequency = f }

  onMouseMove(event: MouseEvent): void {
    if(this.mouseDown){
      if(this.isOnCanvas(event)){
        this.spray();
      }
    }else{
      this.mouseDown = FALSE;
    }
  }

  private isOnCanvas(event): boolean{
    if(this.drawingService.isOnCanvas(event.offsetX, event.offsetY))
      return true;
    else
      return false;
  }
}
```

Text

LOG2990

Matricule : _____

Réponse :

Pas besoin d'un if else juste if c assez

Q8. Gestion de projet (6 points)

Tout au long de la session, vous avez utilisé la plateforme GitLab pour la gestion de votre projet en mode "intégration continue". Expliquez ce qu'est l'intégration continue et donnez deux (2) avantages de l'utilisation de GitLab dans le cadre du projet intégrateur de LOG2990.

chacun travaille sur sa branche pour éviter les conflits, et chacun développe une fonctionnalité et on fait plusieurs sprints aka étapes de développement.

PAGE SUPPLÉMENTAIRE

Si vous avez besoin de place supplémentaire pour une question, vous pouvez utiliser cet espace.
Indiquer **clairement** le numéro de la question répondue.