Rapport de l'Application de Gestion de Todos :

Réalisé par :

Ennerhaima Bothayna.

Encadré par :

Pr.Khamlichi.

Introduction :

Cette application est une liste de tâches développée avec Angular pour l'interface et Spring Boot pour le serveur.

1) todo-list.ts - Le Cerveau de l'Interface :

Le système de subscription est comme un "abonnement" qui nous permet de recevoir des données du serveur de manière automatique.

1.1-Charger la liste :

```
ngOnInit(): void {
  this.loadTodos();
}

loadTodos() {
  this.todoService.getTodos().subscribe(data => this.todos = data);
}
```

Fonctionnement**:**

- subscribe() active l'Observable et lance la requête HTTP
- Lorsque les données arrivent, le callback data => this.todos = data est exécuté
- La liste todos est mise à jour avec les données reçues

1.2-Ajouter un todo :

```
addTodo() {
  if (this.newTitle.trim()) {
    this.todoService.addTodo({title: this.newTitle, completed: false})
      .subscribe(todo => {
        this.todos.push(todo);
        this.newTitle = '';
      });
  }
}
```

- **Rôle**: Attend la confirmation du serveur avant d'ajouter le todo localement.

1.3-Modifier un todo :

```
  toggle(todo: Todo) {
    todo.completed = !todo.completed;
    this.todoService.updateTodo(todo).subscribe();
  }
```

- **Rôle**: Synchronise l'état avec le backend sans attendre de réponse.

1.4-Supprimer un todo :

```
  delete(todo: Todo) {
    this.todoService.deleteTodo(todo.id!).subscribe(() => {
      this.todos = this.todos.filter(t => t.id !== todo.id);
    });
  }
}
```

- **Rôle**: Attend la confirmation de suppression avant de retirer l'élément de la liste locale.

1.5-Les Avantages subscription :

Sans subscription :

- L'application se bloquerait en attendant les données
- On ne saurait pas si les opérations ont réussi
- Il faudrait recharger la page pour voir les changements

Avec subscription :

- L'application reste fluide
- On sait quand les données arrivent
- Tout se met à jour automatiquement
- Meilleure expérience utilisateur

2) todo.service.ts - Le Messager :

```typescript
export class TodoService {
  private http = inject(HttpClient);
  private apiUrl = 'http://localhost:8080/api/todos';

  // constructor(private http: HttpClient) { }

  getTodos(): Observable<Todo[]> {
    return this.http.get<Todo[]>(this.apiUrl);
  }

  addTodo(todo: Todo): Observable<Todo> {
    return this.http.post<Todo>(this.apiUrl, todo);
  }
```

Rôle **:**

- ❖ Envoie les requêtes HTTP au serveur
- ❖ Transforme les réponses en objets TypeScript
- ❖ Gère les URLs de l'API

3) Todo.java - Le Modèle de Données :

```java
public class Todo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Le titre est obligatoire")
    private String title;

    public Todo(Long id, @NotBlank(message = "Le titre est obligatoire") Stri
        super();
        this.id = id;
        this.title = title;
        this.completed = completed;
    }
```

Rôle :

- ❖ Définit la structure d'un todo en base de données
- ❖ Contient les règles de validation (@NotBlank)
- ❖ Gère la génération automatique des IDs

4) TodoController.java - Le Réceptionniste :

```java
@RequestMapping("/api/todos")
@CrossOrigin(origins = "http://localhost:4200", allowCredentials = "true")
@RequiredArgsConstructor
public class TodoController {

    @Autowired
    private TodoService service;
    @GetMapping
    public List<Todo> findAll() {
        return service.findAll();
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Todo create(@Valid @RequestBody Todo todo) {
        return service.create(todo);
    }
```

Rôle :

- Reçoit les requêtes HTTP du frontend
- Appelle le service approprié
- Retourne les réponses au frontend

5) Cycle de vie complet d'un Todo:

5.1-Création :

- Frontend: Formulaire → addTodo() → Service HTTP POST
- Backend: Validation → Persistance → Retour de l'objet créé

5.2-Lecture :

- Frontend: ngOnInit() → loadTodos() → Service HTTP GET
- Backend: Repository → Liste des todos → JSON response

5.3-Mise à jour :

- Frontend: Toggle → updateTodo() → Service HTTP PUT
- Backend: Recherche par ID → Modification → Retour objet modifié

5.4-Suppression :

- Frontend: Delete → deleteTodo() → Service HTTP DELETE
- Backend: Suppression → Retour statut 204