

ENSSAT

L A N N I O N

Rapport JEE

*SERRA Matthieu, HEYRENDT Titouan, CHARBONNEAU
Bastien, CARRE Léo*

I. Architecture

A. Front

1. *Page source*
2. *Sous-pages*
3. *Bus d'évènements*
4. *Intercepteurs*

B. Back

1. *Ressources / Mapper / Entités*
2. *Controller*
3. *Service*
4. *Repository*
5. *Authentication*

II. Choix d'implémentation

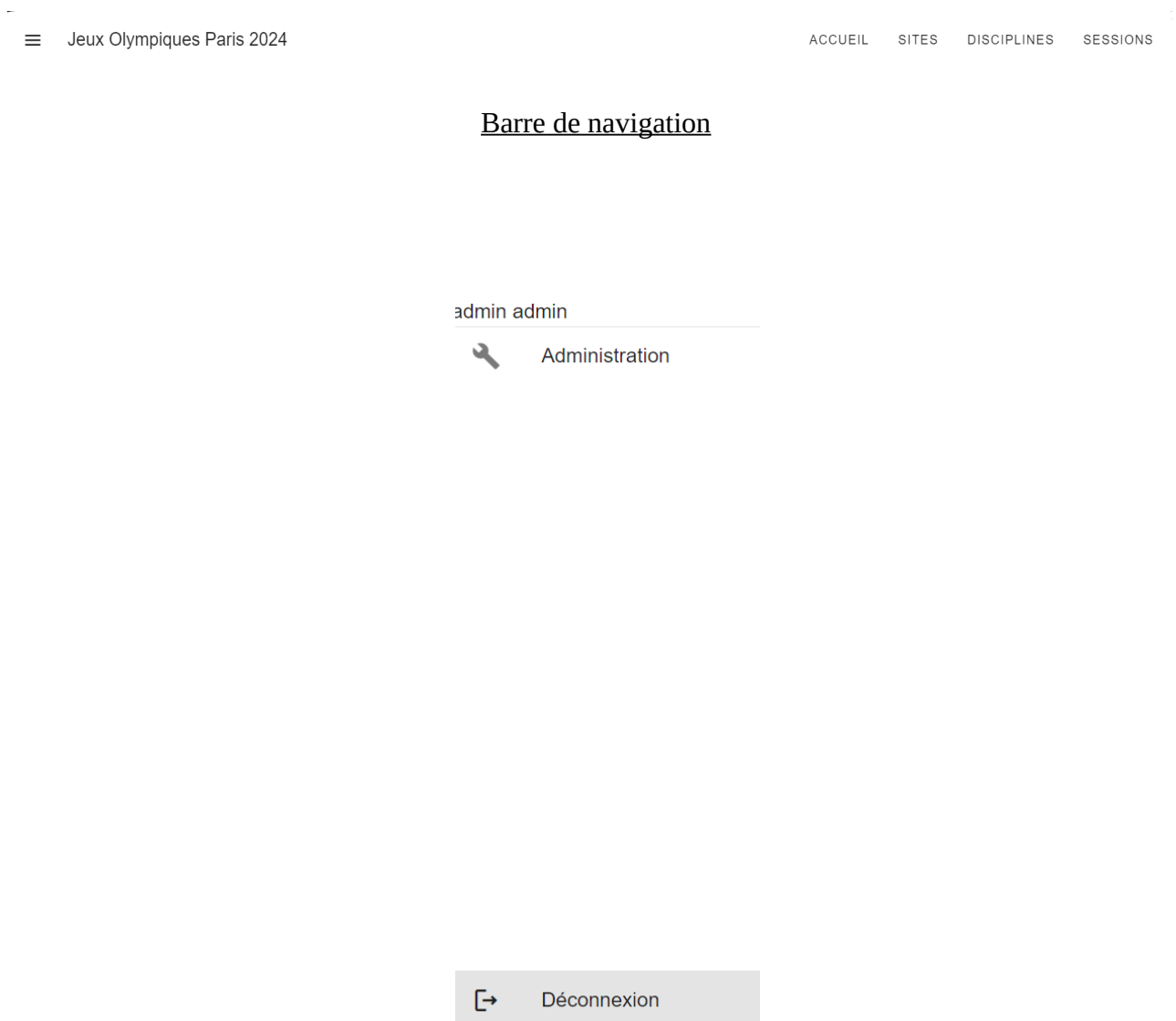
1. *Spring*
2. *Spring boot*
3. *Hibernate*
4. *Lombok*
5. *NPM*
6. *Vue*
7. *Jackson*
8. *Axios*

I. Architecture

A. Front

1. Page source

La page source est l'élément principale de la partie front du projet de JEE. Elle contient des composants persistants accessibles à travers les différentes sous-pages tels que : la barre de navigation et le tiroir de navigation. Cette page est également à l'origine de la logique d'affichage des sous-pages.



Tiroir de navigation avec une connexion administrateur


2. Sous-pages

Les sous-pages disponibles permettent aux utilisateurs d’accéder aux cas d’utilisation définis dans la partie UML du projet. Register et login sont liées à l’UC : s’authentifier. La page admin donne la possibilité aux administrateurs de modifier ou de supprimer des éléments des épreuves. Discipline, sites et sessions répondent au besoin de consulter les épreuves de l’évènement.

Stade de France

Saint-Denis

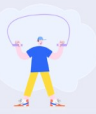
STADE



Allianz Riviera

Nice


STADE



Parc des Princes

Paris

STADE



Athlétisme

- 100m
- 110m haies
- 3000m steeple
- Relais 4x100m

Badminton

Basketball

Boxe

Escrime

Gymnastique artistique

<

ATHLÉTISME

JUDO

BASKETBALL

BADMING1

>

Code	Date ↑	Heure de début	Heure de fin	Discipline	Epreuve	Site	Description	Type session
ATH01	2024-07-25T22:00:00.000+00:00	08:00:00	12:00:00	Athlétisme	100m	Stade de France	Athlétisme Session 1	Qualifications
JUD01	2024-07-25T22:00:00.000+00:00	09:00:00	13:00:00	Judo	-48 kg	Allianz Riviera	Judo Session 1	Qualifications
BAS01	2024-07-25T22:00:00.000+00:00	10:00:00	14:00:00	Basketball	Tournoi à 12 équipes	Parc des Princes	Basketball Session 1	Médailles
ESC01	2024-07-26T22:00:00.000+00:00	14:00:00	18:00:00	Escrime	Epée individuelle	Stade de France	Escrime Session 1	Qualifications
TIR01	2024-07-26T22:00:00.000+00:00	15:00:00	19:00:00	Tir à l'arc Individuel	Tir à l'arc individuel	Allianz Riviera	Tir à l'arc Session 1	Médailles

Items per page: 5 1-5 of 18 < > >|

Pages sites, disciplines et épreuves pour la consultation

SITES

SESSIONS

DISCIPLINES

EPREUVES

AJOUTER SITE

MODIFIER SITE


SUPPRIMER SITE

Id ↑	Nom	Ville	Catégorie
<input type="checkbox"/> 1	Stade de France	Saint-Denis	STADE
<input type="checkbox"/> 2	Allianz Riviera	Nice	STADE
<input type="checkbox"/> 3	Parc des Princes	Paris	STADE
<input type="checkbox"/> 4	Stade Vélodrome	Marseille	STADE
<input type="checkbox"/> 5	Matmut Atlantique	Bordeaux	STADE

1-5 of 5

Page admin pour la modification et la suppressions

Connexion



Nom d'utilisateur



Mot de passe

CONNEXION

CRÉER UN COMPT

Créer un compte

Prénom

Nom

Nom d'utilisateur

Mot de passe

INSCRIRE

DÉJA UN COMPT

3. Bus d'évènements

Cet élément sauvegarde les informations de l'utilisateur à travers les pages pour conserver l'état de connexion et éventuellement ses droits administrateurs. Lorsque l'utilisateur est administrateur et qu'il se connecte, l'information est reliée dans le front pour afficher notamment la page admin.

```
export const EventBus : {...} = {
  getState: () : DeepReadonly<UnwrapNestedRefs<...> => readonly(state),
  login: (role) : void => {
    state.isLoggedIn = true;
    state.userRole = role;
  }, logout: () : void => {
    state.isLoggedIn = false;
    state.userRole = null;
  },
  initializeAuthState: () : void => {
    const token : string = localStorage.getItem( key: 'jwtToken' );
    if (token) {
      const role : string = localStorage.getItem( key: 'role' ); // Assurez-vous que le rôle est également stocké dans le localStorage
      EventBus.login(role);
    }
  }
};
EventBus.initializeAuthState();
```

Intégration du bus d'évènements

4. Intercepteurs

Les intercepteurs sont responsables de l'ajout du jwtToken dans les requêtes GET, PUT, POST et DELETE faites à l'api. Il récupère également chaque réponse de l'API, si la réponse est une erreur 401 (token expiré) alors on redirige sur la page login et on affiche une notification d'erreur. Pour ce faire, la librairie axios a été utilisée.

```
axios.interceptors.request.use( onFulfilled: function (config : InternalAxiosRequestConfig ) : InternalAxiosRequestConfig<any> {
  const token : string = localStorage.getItem( key: 'jwtToken' );
  config.headers.Authorization = token ? `Bearer ${token}` : '';
  return config;
});

axios.interceptors.response.use( onFulfilled: response : AxiosResponse => response, onRejected: error => {
  if (error.response.status === 401) {
    this.$router.push('/login');
    notifyUser( type: 'error', title: 'Session Expirée', text: 'Votre session a expiré. Veuillez vous reconnecter.' );
  }
  return Promise.reject(error);
});
```

Utilisation de la librairie axios

B. Back

1. Ressources / Mapper / Entités

Les entités correspondent aux disciplines, sessions, sites, ... sous leurs formats stockés en base de données. Tandis que les ressources sont ces mêmes éléments mais ce nouveau format permet des interactions avec le front par le biais de Jackson qui convertit les ressources en Json. Le mappeur permet de transformer une entité en ressources et inversement.

```
@Mapper(config = MapperConfig.class)
public interface DisciplineMapper {

    4 usages
    DisciplineResource disciplineToDisciplineResource(Discipline discipline);

    1 usage
    @Mapping(target="id",ignore=true)
    Discipline disciplineResourceToDiscipline(DisciplineResource disciplineResource);

    1 usage
    @Mapping(target="id",ignore=true)
    void updateDisciplineFromResource(DisciplineResource disciplineResource, @MappingTarget Discipline discipline);
}
```

Mapper des disciplines

```
@Entity
public class Discipline {

    4 usages
    @Column(name = "nom")
    private String nom;

    4 usages
    @Column(name="estParalympique",columnDefinition = "BOOLEAN")
    private boolean estParalympique;

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

L'entité discipline

```
public class DisciplineResource {
    4 usages
    private String nom;
    4 usages
    private boolean estParalympique;
    4 usages
    private Long id;
}
```

La ressource discipline

6/12

2. Controller

Les controllers lient les requêtes de modifications, de suppressions, d'inscriptions et de connexions du front aux méthodes du back. Ils sont la passerelle qui redirige la demande vers les services adaptés.

Ce sont les annotations `@GetMapping`, `@PutMapping`, `@PostMapping` et `@DeleteMapping` qui sont responsable de la redirection vers la bonne méthode du controller. L'annotation `@PreAuthorize` gère l'accès à la méthode du controller en fonction du rôle de l'utilisateur.

```
@PostMapping("/create")
@PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_ADMINISTRATIVE_MANAGER')")
public ResponseEntity<Object> createDiscipline(@RequestBody DisciplineResource disciplineResource){
    try{
        DisciplineResource createdDisciplineResource = disciplineServiceImpl.createDiscipline(disciplineResource);
        return ResponseEntity.status(HttpStatus.CREATED).body(createdDisciplineResource);
    }catch (Exception e){
        return ResponseEntity.status(HttpStatus.CONFLICT).body(e.getMessage());
    }
}

@PutMapping("/{id}")
@PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_ADMINISTRATIVE_MANAGER')")
public ResponseEntity<Object> updateDiscipline(@PathVariable("id") Long id, @RequestBody DisciplineResource disciplineResource){
    try{
        DisciplineResource updatedDisciplineResource = disciplineServiceImpl.updateDiscipline(disciplineResource, id);
        return ResponseEntity.status(HttpStatus.OK).body(updatedDisciplineResource);
    }catch (EntityNotFoundException e){
        return ResponseEntity.notFound().build();
    }catch (Exception e){
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(e.getMessage());
    }
}

@DeleteMapping("/{id}")
@PreAuthorize("hasAnyRole('ROLE_ADMIN', 'ROLE_ADMINISTRATIVE_MANAGER')")
public ResponseEntity<Object> deleteDiscipline(@PathVariable("id") Long id){
    try{
        disciplineServiceImpl.deleteDiscipline(id);
        return ResponseEntity.status(HttpStatus.OK).body("Discipline avec l'id :"+id+" supprimé");
    }catch (Exception e){
        return ResponseEntity.status(HttpStatus.CONFLICT).body("Discipline avec l'id :"+id+" introuvable");
    }
}
```

Controller des disciplines

3. Service

L'api répond à un cahier des charges qui comporte les actions qu'elle doit réaliser. Ainsi, ces actions sont agrégées dans des services. Les services disponibles sont donc, la suppression, la modification, la connexion et l'inscription. Pour réaliser les actions le service fait appel à un mapper qui transforme la ressource (DTO) en entité (notre model). La ressource est l'objet qui va transporter les données de notre entité à travers les différents processus. Le repository va lui faire la passerelle entre la base de donnée et l'API.

```
@Override
public Iterable<DisciplineResource> getDisciplines() {
    Iterable<Discipline> disciplines = disciplineRepository.findAll();
    return StreamSupport.stream(disciplines.spliterator(), parallel: false) Stream<Discipline>
        .map(disciplineMapper::disciplineToDisciplineResource) Stream<DisciplineResource>
        .collect(Collectors.toList());
}

1 usage
public DisciplineResource getDiscipline(Long id){
    Discipline discipline = disciplineRepository.findById(id).orElseThrow(()-> new EntityNotFoundException("No Discipline found in d
    return disciplineMapper.disciplineToDisciplineResource(discipline);
}

1 usage
@Override
public DisciplineResource createDiscipline(DisciplineResource disciplineResource) throws IOException{
    Discipline discipline = disciplineMapper.disciplineResourceToDiscipline(disciplineResource);
    if(!estEnDB(disciplineResource)){
        disciplineRepository.save(discipline);
    }else{
        throw new IOException("Discipline déjà présente en DB");
    }
    return disciplineMapper.disciplineToDisciplineResource(discipline);
}

1 usage
@Override
public DisciplineResource updateDiscipline(DisciplineResource disciplineResource, Long id){
    Discipline discipline = disciplineRepository.findById(id).orElseThrow(()-> new EntityNotFoundException("Discipline not found wit
    disciplineMapper.updateDisciplineFromResource(disciplineResource,discipline);
    disciplineRepository.save(discipline);
    return disciplineMapper.disciplineToDisciplineResource(discipline);
}

1 usage
@Override
public void deleteDiscipline(Long id){disciplineRepository.deleteById(id);}
```

Implémentation des services liés à la discipline

4. Repository

Dans un projet Spring, le repository est une abstraction qui encapsule le stockage, la récupération et le comportement de recherche des données. Spring Data Repositories fournissent des méthodes prêtes à l'emploi pour les opérations CRUD (Create, Read, Update, Delete) sur des entités, telles que les disciplines, en base de données. Ces méthodes s'appuient sur la génération automatique de requêtes SQL, éliminant ainsi le besoin d'écrire du code SQL manuellement.


```

@Repository
public interface DisciplineRepository extends CrudRepository<Discipline, Long> {

    1 usage
    boolean existsByNomAndEstParalympique(String nom, boolean estParalympique);

    4 usages
    Discipline findByNom(String nom);
}

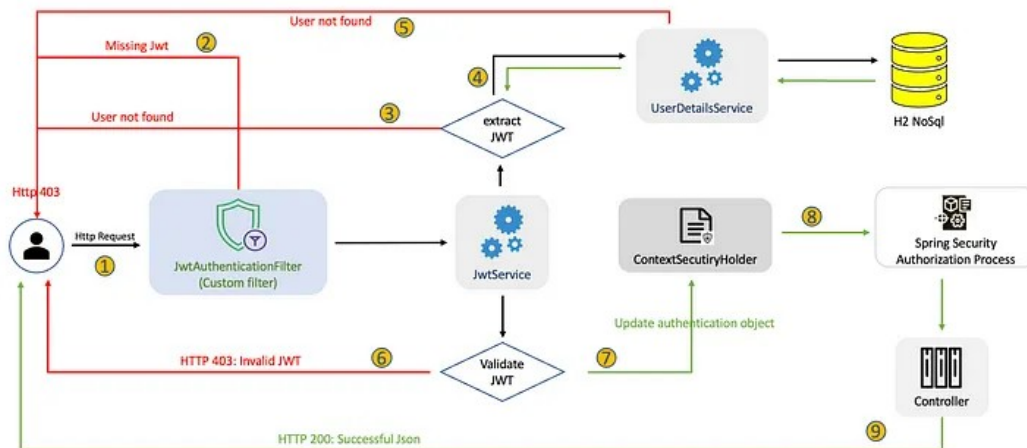
```

Repository spring configuré

5. Authentification

1. **Requête HTTP** : Une requête HTTP arrive au serveur.
2. **JwtAuthenticationFilter (filtre personnalisé)** : Spring Security passe la requête au `JwtAuthenticationFilter`. Ce filtre vérifie la présence d'un JWT dans les en-têtes de la requête.
3. **JWT manquant ou non trouvé** : Si le JWT est absent ou non trouvé, une erreur HTTP 403 est renvoyée, indiquant un accès refusé.
4. **Extraction du JWT** : Si un JWT est trouvé, le `JwtAuthenticationFilter` l'extrait pour le traitement.
5. **Utilisateur non trouvé** : Si le nom d'utilisateur n'est pas trouvé dans le JWT ou que le JWT ne correspond à aucun utilisateur connu, une erreur est renvoyée indiquant que l'utilisateur n'est pas trouvé.
6. **Validation du JWT** : Le `JwtService` valide le JWT pour s'assurer qu'il est bien formé, valide et non expiré.
7. **JWT invalide** : Si le JWT est invalide pour une raison quelconque (mal formé, expiré, etc.), une erreur HTTP 403 est renvoyée.
8. **Mise à jour de l'objet d'authentification** : Si le JWT est valide, le `JwtService` crée ou récupère les détails de l'utilisateur associé et met à jour l'objet d'authentification dans le `SecurityContextHolder`.
9. **Processus d'autorisation Spring Security** : Avec l'objet d'authentification mis à jour, Spring Security peut alors procéder au processus d'autorisation pour déterminer si l'utilisateur a les droits nécessaires pour accéder à la ressource demandée.
10. **Réponse HTTP 200** : Si le processus d'authentification et d'autorisation réussit, une réponse HTTP 200 est renvoyée avec le contenu demandé (généralement au format JSON).

11. Contrôleur : Finalement, la requête est transmise au contrôleur approprié pour traiter la requête d'origine après que l'utilisateur a été authentifié avec succès.



```

@Override
public JwtAuthenticationResponse signin(SignInRequest request) {
    authenticationManager.authenticate(
        new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword()));
    var user = userRepository.findByUsername(request.getUsername())
        .orElseThrow(() -> new IllegalArgumentException("Invalid username or password"));
    var jwt = jwtService.generateToken(user);
    String firstName = user.getFirstName();
    String lastName = user.getLastName();
    Role role = user.getRole();
    return JwtAuthenticationResponse.builder().token(jwt).firstName(firstName).lastName(lastName).role(role).build();
}
  
```

Implémentation du service de connexion

```

axios.post( url: 'http://localhost:3001/auth/signin', credentials)
    .then(response => {
        const { token, firstName, lastName, role } = response.data;
        localStorage.setItem('jwtToken', token);
        localStorage.setItem('firstName', firstName);
        localStorage.setItem('lastName', lastName);
        localStorage.setItem('role', role);
    });
  
```

Stockage du token « jwtToken » dans le local storage du navigateur

II. Choix d'implémentation

1. Spring

Spring est un framework Java open source donnant accès à de nombreux outils pratiques pour le développement d'applications. Ici, nous avons utilisé l'injection de dépendances de données mais aussi Spring Security pour l'authentification de la page web.

```
@Autowired
private SiteServiceImpl siteServiceImpl;
```

Injection de dépendances grâce à Spring

2. Spring boot

Spring boot est un projet du framework Spring. Il permet d'accélérer le processus de développement en intégrant des serveurs embarqués tels que tomcat. Il facilite ainsi la configuration de ces derniers.

3. Hibernate

Hibernate est un framework Java qui facilite l'interaction avec les bases de données relationnelles telle que la notre. Il permet de manipuler des objets Java plutôt que des requêtes SQL. Ces objets persistants provenant de la base de données ont un tag @Entity pour utiliser Jackson.

```
@Entity
public class Discipline {
    4 usages
    @Column(name = "nom")
    private String nom;
    4 usages
    @Column(name="estParalympique",columnDefinition = "BOOLEAN")
    private boolean estParalympique;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
```

L'objet discipline persistant issue de la base de données

4. Lombok

Lombok est une bibliothèque Java qui permet de générer automatiquement du code simple tel que les méthodes getter, setter, equals, toString et d'autres. Il rend également le code plus lisible et fonctionne grâce à des tags Getter et Setter au dessus des variables de classes.

5. NPM

NPM est le gestionnaire de paquets par défaut pour Node.js. Il facilite l'installation, la gestion et la distribution de bibliothèques JavaScript. Il a notamment été utile pour installer axios (cf 8.).

6. Vue

Vue js est un framework qui permet de construire des interfaces utilisateur plus facilement. Il donne accès à de nombreux composants qui sont des classiques des applications web comme la navigation bar, les composants de connexion et d'inscription

7. Jackson

Les données échangées entre l'api et le client se fait par un format léger et rapide appelé Json. Or, l'api ne peut directement effectuer des modifications sur ces données. Ainsi, elles sont pré-traitées par une bibliothèque Java qui transforme le Json du client en ressource exploitable par l'api.

```
public class DisciplineResource {  
    4 usages  
    private String nom;  
    4 usages  
    private boolean estParalympique;  
    4 usages  
    private Long id;  
  
    // Constructeur par défaut  
    no usages  
    public DisciplineResource(){  
  
    }  
  
    // Constructeur avec paramètres  
    no usages  
    public DisciplineResource(String nom, boolean estParalympique, Long id){  
        this.nom = nom;  
        this.estParalympique = estParalympique;  
        this.id = id;  
    }  
}
```

Exemple de ressource exploitable par l'api : disciplineResource

8. Axios

Axios est une bibliothèque JavaScript utilisée pour faciliter les requêtes http faites depuis notre application web vers notre api. Elle offre une syntaxe plus simple et expressive.

```
axios.get( url: 'http://localhost:3001/site')  
    .then(response => {  
        this.sites = response.data;  
    })  
    .catch(error => {  
        console.error('Erreur lors du chargement des sites:', error);  
    });  
axios.get( url: 'http://localhost:3001/session')  
    .then(response => {  
        this.sessions = response.data;  
    })  
    .catch(error => {  
        console.error('Erreur lors du chargement des sites:', error);  
    });  
axios.get( url: 'http://localhost:3001/discipline')  
    .then(response => {  
        this.disciplines = response.data;  
    })  
    .catch(error => {  
        console.error('Erreur lors du chargement des sites:', error);  
    });
```

Utilisation d'axios pour faire des requêtes get à l'api ayant pour but la création de site, session et discipline

Conclusion

Ce projet étant composé d'une partie front et back a permis de découvrir ou de s'exercer sur un pan de la programmation peu exploré pour certains dont leur métier est très concentré sur du front ou du back uniquement.