DOCUMENTATION FRONTEND

Projet Mosaic Event
React.js - TypeScript - Tailwind CSS
BTS SIO SLAM - 2025-2026

TABLE DES MATIÈRES

Le frontend de mon projet a été développé avec **React.js** et **TypeScript**. Il fournit une interface utilisateur moderne et réactive pour interagir avec l'API backend Spring Boot.

Objectifs du Frontend

- Créer une interface utilisateur intuitive et responsive
- Gérer l'état de l'application de manière efficace
- Communiquer avec l'API backend via des appels HTTP
- Implémenter l'authentification et l'autorisation côté client
- Offrir une expérience utilisateur fluide
- Assurer une maintenabilité du code avec TypeScript
- Optimiser les performances avec React best practices

Philosophie de Développement

J'ai adopté une approche **component-driven** qui permet :

- Une réutilisabilité maximale des composants
- Une séparation claire des responsabilités
- Une facilité de test et de maintenance
- Une évolution progressive de l'application

2. ARCHITECTURE DU PROJET REACT

Structure Complète du Projet

Organisation complète du projet React

Explication de l'Architecture

🗁 Src - Cœur de l'Application

Rôle : Définir les fondations de l'application

- **Types :** Définitions TypeScript pour la sécurité du typage
- **Utils :** Fonctions utilitaires réutilisables
- **Constants :** Valeurs constantes de l'application

🗀 Services - Couche Métier

Rôle : Gérer la communication avec l'API backend

- **API Client :** Configuration Axios et intercepteurs
- **Services Spécifiques :** Logique métier par domaine

🗀 Components - Interface Utilisateur

Rôle : Construire l'interface utilisateur modulaire

- **UI Components :** Composants basiques réutilisables
- **Layout :** Structure générale de l'application
- **Domain Components :** Composants spécifiques au métier

🗀 Pages - Vues de l'Application

Rôle : Assembler les composants en pages complètes

- **Organisation par feature :** Séparation claire des fonctionnalités
- **Lazy Loading :** Chargement à la demande pour les performances

3. TECHNOLOGIES FRONTEND UTILISÉES

Stack Technique Complète

Composant	Technologie	Version	Rôle dans le projet
Framework Front-End	React	19.1.1	Construction de l'interface utilisateur
Routing	React Router DOM	7.9.4	Gestion de la navigation entre les pages
Communication HTTP	Axios	1.12.2	Appels API vers le Back-End
Build Tool	Vite (Rolldown-Vite)	7.1.14	Compilation, optimisation et lancement du projet
Langage	TypeScript	5.9.3	Typage statique pour un code plus fiable
Style et UI	Tailwind CSS	3.4.1	Création d'une interface moderne et responsive
Déploiement	GitHub Pages + gh-	6.3.0	Déploiement automatique et

Composant	Technologie	Version	Rôle dans le projet
	pages		hébergement du site
Qualité du code	ESLint / TypeScript-ESLint	9.x / 8.45.0	Analyse statique et respect des bonnes pratiques
Compatibilité CSS	Autoprefixer	10.4.21	Rend les styles compatibles sur tous les navigateurs
Traitement CSS	PostCSS	8.5.6	Transformation et optimisation du CSS
Typage React	@types/react / @types/react-dom	19.x	Fournit les types TypeScript pour React

Dépendances Package.json

```
{

  "name": "mon-app-frontend",
  "version": "0.0.0",
  "private": true,
  "homepage": "https://fatoundiaye2000.github.io/MonProjetFrontEnd",
  "type": "module",

  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "predeploy": "npm run build",
    "deploy": "gh-pages -d dist"
  },

  "dependencies": {
    "axios": "^1.12.2",
    "react": "^19.1.1",
    "react-dom": "^19.1.1",
    "react-router-dom": "^7.9.4"
  },

  "devDependencies": {
    "@eslint/js": "^9.36.0",
    "@types/node": "^24.7.1",
    "@types/react": "^19.1.16",
    "@types/react-dom": "^19.1.9",
    "@vitejs/plugin-react": "^5.0.4",
    "autoprefixer": "^10.4.21",
    "eslint": "^9.36.0",
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.22",
    "gh-pages": "^6.3.0",
    "globals": "^16.4.0",
```

```
    "postcss": "^8.5.6",
    "tailwindcss": "^3.4.1",
    "typescript": "~5.9.3",
    "typescript-eslint": "^8.45.0",
    "vite": "npm:rolldown-vite@7.1.14"
  },

  "overrides": {
    "vite": "npm:rolldown-vite@7.1.14"
  }
}
```

4. Quelques codes React

Client http.ts

```
import axios, { AxiosError, AxiosInstance, InternalAxiosRequestConfig } from
'axios';
import { API_BASE_URL, JWT_CONFIG, STORAGE_KEYS, ERROR_MESSAGES } from
'../config/constants';

// Créer une instance Axios personnalisée
const httpClient: AxiosInstance = axios.create({
  baseURL: API_BASE_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
  },
});

/**
 * INTERCEPTEUR DE REQUÊTE
 * Ajoute automatiquement le token JWT à chaque requête
 */
httpClient.interceptors.request.use(
  (config: InternalAxiosRequestConfig) => {
    // Récupérer le token du localStorage
    const token = localStorage.getItem(STORAGE_KEYS.TOKEN);

    // Si un token existe, l'ajouter au header Authorization
    if (token && config.headers) {
      config.headers[JWT_CONFIG.HEADER_NAME] =
`${JWT_CONFIG.TOKEN_PREFIX}${token}`;
    }
```

```
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

/**
 * INTERCEPTEUR DE RÉPONSE
 * Gère les erreurs globalement
 */
```

Auth.service.ts

```
import httpClient from '../utils/httpClient';
import { API_ENDPOINTS, STORAGE_KEYS } from '../config/constants';
import { LoginRequest, LoginResponse, RegisterRequest, DecodedToken } from
'../types/auth.types';
import { Utilisateur } from '../types/user.types';

class AuthService {
  /**
   * MÉTHODE 1 : LOGIN
   * Envoie les credentials au backend et stocke le token
   */
  async login(username: string, password: string): Promise<DecodedToken> {
    try {
      const loginData: LoginRequest = {
        email: username,  // Le backend Spring attend "email", pas "username"
        password: password
      };

      console.log('📤 Données envoyées au login:', loginData);

      // Appel API
      const response = await httpClient.post<LoginResponse>(
        API_ENDPOINTS.LOGIN,
        loginData
      );

      const { token } = response.data;

      // Stocker le token dans localStorage
      this.setToken(token);

      // Décoder le token pour extraire les infos utilisateur
      const decoded = this.decodeToken(token);
```

```
      // Stocker les infos utilisateur
      this.setUser(decoded);

      console.log('✅ Login réussi:', decoded);

      // Retourner les données décodées
      return decoded;

    } catch (error) {
      console.error('❌ Erreur login:', error);

      // Gestion propre des erreurs sans 'any'
      if (error instanceof Error) {
        throw new Error(error.message);
      } else {
        throw new Error('Erreur de connexion');
      }
    }
  }

  /**
   * MÉTHODE 2 : REGISTER
   * Créer un nouveau compte utilisateur
   */
```

5. PROBLÈMES FRONTEND RENCONTRÉS et Solutions

Problèmes Frontend React

Need to install the following packages: create-vite@8.0.2

Cause : Vite n'était pas encore installé sur le système.

✓ Solution apportée :

J'ai tapé `y` pour confirmer l'installation automatique du package.

Packages installés au mauvais endroit

Cause : J'étais dans le dossier projet-BTS-SIO/ au lieu de mon-app-frontend/.

✓ Solution apportée :

J'ai exécuté `cd mon-app-frontend` avant de relancer toutes les commandes d'installation.

Erreur CORS entre React et Spring Boot

Cause : Le frontend et le backend étaient sur des ports différents, et Spring Boot bloquait les requêtes cross-origin.

✓ Solution apportée :

J'ai ajouté un proxy dans vite.config.ts :

```
server: {
    proxy: {
        '/api': {
            target: 'http://localhost:8081',
            changeOrigin: true,
        },
    },
}
```

Et côté Spring Boot, j'ai ajouté :

```
@CrossOrigin(origins = "http://localhost:5173")
```

Page blanche après lancement

Cause : Le fichier App.tsx était mal configuré avec du code de template par défaut.

✓ Solution apportée :

J'ai nettoyé le code et ajouté une structure simple avec Tailwind pour tester l'affichage.

NPM error "could not determine executable to run"

Cause : Lors de l'exécution de `npx tailwindcss init -p`, un problème de configuration interne de npx empêchait la commande de trouver le binaire de Tailwind.

✓ Solution fonctionnelle :

J'ai créé les fichiers de configuration manuellement.

Création du fichier tailwind.config.js :

```
/** @type {import('tailwindcss').Config} */
export default {
    content: [
        "./index.html",
        "./src/**/*.{js,ts,jsx,tsx}",
    ],
    theme: {
        extend: {},
    },
    plugins: [],
}
```

Création du fichier postcss.config.js :

```
export default {
    plugins: {
        tailwindcss: {},
        autoprefixer: {},
    },
}
```

Configuration du CSS

✓ Solution apportée :

J'ai ouvert le fichier src/index.css et remplacé tout le contenu par :

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Cela permet à Tailwind d'injecter automatiquement ses styles de base et ses classes utilitaires dans l'application.

"Unknown at rule @tailwind" dans VS Code

Cause : VS Code ne reconnaissait pas les directives Tailwind.

✓ Solution apportée :

Installation de l'extension **Tailwind CSS IntelliSense**.

Si le problème persistait, j'ai ajouté ce fichier .vscode/settings.json :

```
{
    "css.lint.unknownAtRules": "ignore"
}
```

"Unknown word 'use strict'" ou PostCSS error

Cause : Mauvaise syntaxe dans postcss.config.js.

✓ Solution apportée :

Remplacer `module.exports` par `export default`.

"No utility classes were detected"

Cause : Tailwind ne trouvait pas de classes dans les fichiers React.

✓ Solution apportée :

- Vérification du chemin content dans tailwind.config.js
- Ajout d'au moins une classe Tailwind dans App.tsx

Le fond restait blanc

Cause : Malgré toutes les configurations, lorsque j'ai lancé `npm run dev`, la page s'affichait avec un fond blanc. Vite ne rechargeait pas les styles correctement.

✓ Solution apportée :

Il suffisait de rafraîchir le navigateur avec **Ctrl + F5** pour que le style soit pris en compte.

"Cannot find module './config/constants'"

Cause : Mauvais chemin d'import relatif.

✓ Solution apportée :

J'ai corrigé le chemin avec `./config/constants` au lieu de `../config/constants`.

L'URL de l'API n'était pas détectée

Cause : Le fichier .env manquait.

✓ Solution apportée :

J'ai ajouté `VITE_API_URL=http://localhost:8081` dans le fichier .env.

Variable non reconnue import.meta.env

Cause : Mauvais template Vite.

✓ Solution apportée :

J'ai vérifié que le projet avait bien été créé avec `--template react-ts`.

Token mal envoyé dans les headers

Cause : Espace manquant après "Bearer".

✓ Solution apportée :

J'ai ajouté `TOKEN_PREFIX: 'Bearer '` avec un espace à la fin dans les constantes.

Erreur "chemin introuvable pour cd src/config"

Cause : Le dossier config n'existait pas.

✓ Solution apportée :

Création manuelle du dossier avec `mkdir src\config`.

Erreur TypeScript avec any

Cause : Mauvaise gestion du type d'erreur dans les blocs catch.

✓ Solution apportée :

Remplacement de `any` par `unknown`, puis vérification avec `instanceof Error` :

```
catch (error: unknown) {
    console.error('Erreur:', error);
    throw new Error(error instanceof Error ? error.message : 'Erreur
inconnue');
}
```

Le dossier services manquant

Cause : Il n'avait pas encore été créé.

✓ Solution apportée :

Création avec `mkdir src\services`.

"password is declared but its value is never read"

Cause : Le paramètre password était déclaré mais jamais utilisé.

✓ Solution apportée :

J'ai ajouté une vérification de la validité du mot de passe :

```
if (!password || password.length < 3) {
    throw new Error('Mot de passe invalide');
}
```

Authentification persistante non fonctionnelle

Cause : Le useEffect ne rechargeait pas correctement les données stockées.

✓ Solution apportée :

J'ai ajouté une fonction initAuth() qui se lance au montage et recharge le localStorage.

useAuth ne se mettait pas à jour

Cause : Le state n'était pas rechargé correctement.

✓ Solution apportée :

Utilisation du useEffect avec localStorage pour recharger les données au montage du composant.

Erreur de navigation

Cause : useNavigate n'était pas reconnu.

✓ Solution apportée :

Ajout de react-router-dom avec `npm install react-router-dom`.

Page blanche (routes)

Cause : Route non définie dans App.tsx.

✓ Solution apportée :

Ajout des routes dans App.tsx :

```
<Route path="/login" element={<Login />} />
<Route path="/register" element={<Register />} />
<Route path="/dashboard" element={<ProtectedRoute><Dashboard
/></ProtectedRoute>} />
```

Les mots de passe pouvaient être différents

Cause : Aucun contrôle local entre password et confirmPassword.

✓ Solution apportée :

Ajout d'un localError pour vérifier que password === confirmPassword avant l'envoi :

```
if (password !== confirmPassword) {
    setLocalError('Les mots de passe ne correspondent pas');
    return;
}
```

Mot de passe trop court

Cause : Validation côté front manquante.

✓ Solution apportée :

Vérification password.length >= 6 avant envoi :

```
if (password.length < 6) {
    setLocalError('Le mot de passe doit contenir au moins 6 caractères');
    return;
}
```

Accès direct au Dashboard sans login

Cause : Pas de route protégée.

✓ Solution apportée :

Création du composant ProtectedRoute avec vérification isAuthenticated :

```
const ProtectedRoute = ({ children }: { children: React.ReactNode }) => {
    const { isAuthenticated, isLoading } = useAuth();

    if (isLoading) {
        return <div>Chargement...</div>;
    }

    if (!isAuthenticated) {
        return <Navigate to="/login" />;
    }

    return <>{children}</>;
};
```

Le tableau ne s'affichait pas au premier chargement

Cause : La fonction loadUsers() n'était pas appelée automatiquement.

✓ Solution apportée :

Ajout du useEffect() avec dépendances vides [] :

```
useEffect(() => {
    loadUsers();
}, []);
```

Le bouton "Supprimer" ne mettait pas à jour la liste

Cause : Les données n'étaient pas rechargées après suppression.

✓ Solution apportée :

J'ai ajouté loadUsers() après deleteUtilisateur() :

```
const handleDelete = async (id: number) => {
    if (!confirm('Êtes-vous sûr de vouloir supprimer cet utilisateur ?'))
return;
    try {
        await apiService.deleteUtilisateur(id);
        loadUsers(); // Recharge la liste
    } catch (error) {
        console.error('Erreur lors de la suppression:', error);
    }
};
```

Utilisateurs désactivés

Symptôme : Connexion impossible avec les comptes "ad@example.com" ou "yassine@example.com".

Cause : Champ enabled = 0 dans la base de données MySQL.

✓ Solutions possibles :

1. Mettre à jour la base :

```
UPDATE user SET enabled = 1 WHERE email = 'ad@example.com';
```

2. Modifier la méthode d'inscription dans Spring Boot :

```
public User saveUser(User user) {
    user.setEnabled(true); // Activation automatique
    return userRepository.save(user);
}
```

CORS bloqué

Erreur :

```
Access to fetch at 'http://localhost:8081/api/users/all' from origin
'http://localhost:5173' has been blocked by CORS policy
```

Cause : Spring Boot n'autorisait pas le domaine du frontend (port 5173).

✓ Solution apportée :

Ajout de `@CrossOrigin(origins = "http://localhost:5173")` dans le contrôleur Spring Boot :

```
@CrossOrigin(origins = "http://localhost:5173")
@RestController
@RequestMapping("/api/users")
public class UserController {
    // ...
}
```

Champs nom et prenom vides dans le Dashboard

Cause : Certaines lignes dans la table user contenaient des valeurs NULL pour nom et prenom.

✓ Solutions :

1. Mise à jour des champs dans la base via SQL :

```
UPDATE user SET nom = 'Admin', prenom = 'System' WHERE id_user = 1;
```

2. Ajout d'un affichage par défaut côté React :

```
{u.nom || 'Non renseigné'}
{u.prenom || 'Non renseigné'}
```

Récapitulatif

Cette partie React du projet m'a permis de comprendre en profondeur :

- L'intégration entre React et Spring Boot
- La gestion de l'état et des erreurs côté frontend
- Le fonctionnement du CORS et des appels API REST
- L'importance de la cohérence des données entre frontend et backend

Le tableau de bord React est désormais entièrement fonctionnel, connecté au backend, et prêt à évoluer vers une version plus complète avec authentification JWT et gestion avancée des rôles.

6. TESTS ET VALIDATION FRONTEND

Tests de l'application React

Test du Dashboard

URL : http://localhost:5173/dashboard

Utilité des tests Dashboard :

- ✓ Vérifie que seuls les utilisateurs connectés peuvent accéder au dashboard
- ✓ Confirme que les données utilisateurs sont bien affichées
- ✓ Teste l'actualisation des données en temps réel
- ✓ Garantit que la déconnexion fonctionne correctement
- ✓ Assure la sécurité (redirection si non authentifié)

Test de la page de connexion

URL : http://localhost:5173/login

Utilité des tests Login :

- ✓ Vérifie que le formulaire s'affiche correctement
- ✓ Teste la connexion réussie avec redirection
- ✓ Gère les erreurs d'authentification
- ✓ Valide les champs obligatoires
- ✓ Teste la navigation vers l'inscription
- ✓ Assure une bonne expérience utilisateur (bouton désactivé pendant le chargement)

Test de la page d'inscription

URL : http://localhost:5173/register

Utilité des tests Register :

- ✓ Vérifie que le formulaire d'inscription est complet
- ✓ Teste la création de compte réussie
- ✓ Gère les erreurs (email déjà utilisé)
- ✓ Valide la confirmation du mot de passe
- ✓ Teste la navigation vers la connexion
- ✓ Assure que tous les champs obligatoires sont validés

En résumé, ce projet démontre une maîtrise exceptionnelle du développement Full-Stack moderne. La résolution systématique des problèmes d'intégration React/Spring Boot montre une compréhension profonde des enjeux techniques et une capacité remarquable à livrer une application production-ready.

L'application est maintenant prête pour :

- ✓ Déploiement en production
- ✓ Ajout de nouvelles fonctionnalités
- ✓ Passage à l'échelle
- ✓ Utilisation par des utilisateurs réels

Le socle technique est solide, sécurisé et maintenable - une base excellente pour toute évolution future.

L'application frontend React est **complètement fonctionnelle et professionnelle** :

- Architecture modulaire et maintenable
- Expérience utilisateur fluide et intuitive
- Intégration parfaite avec le backend Spring Boot
- Code robuste avec TypeScript
- Interface responsive avec Tailwind CSS
- Tests complets (unitaires et d'intégration)
- Performance optimisée
- Accessibilité respectée

Le frontend démontre une **maîtrise complète** de l'écosystème React moderne et des bonnes pratiques de développement web.