OKAMI CONNECT

OLIVIA NANQUETTE

2025-2026

Pour le titre de développeur web et web mobile

Sommaire

INTRODUCTION

Le festival OKAMI est né de la volonté d'un groupe d'amis, passionnés de festivals électroniques, de créer une bulle hors du temps. Un espace de déconnexion, de partage, et d'amour, où les participants peuvent se retrouver autour de l'expression artistique sous toutes ses formes.

Porté par une petite association composée principalement de bénévoles, le festival a déjà connu quatre éditions saluées par le public. Malgré une année 2024 marquée par des difficultés financières, une nouvelle page s'ouvre en 2025, avec un nouveau lieu et de nouveaux espoirs pour l'avenir d'OKAMI.

M'investissant de plus en plus dans l'organisation, j'ai souhaité mettre à profit les compétences acquises au cours de ma formation de développeur web et web mobile pour créer des outils numériques facilitant la gestion du festival.

Ce projet comprendra :

- une partie front-end pour l'interface des festivaliers
- une partie back-end avec gestion de base de données pour les besoins interne

Ce dossier présente, étape par étape, la conception et la réalisation de cette plateforme.

Je suis heureuse de vous partager ce projet qui me tient profondément à cœur.

NOTE D'INTENTION

1. Contexte

Le festival OKAMI est organisé par une équipe de passionnés non professionnels, œuvrant majoritairement bénévolement. Cette configuration engendre chaque année des défis organisationnels, notamment dans la coordination des différents acteurs du festival.
Ce projet vise à fluidifier la gestion interne, renforcer la collaboration avec les intervenants, et améliorer l'expérience utilisateur globale, tant pour les organisateurs que pour le public.

2. Problèmes observés / rapportés

- Trop de tableaux pour l'organisation des différents intervenants
- Dépendance technique vis-à-vis du développeur pour toute mise à jour du site web.

3. objectifs

Afin de répondre à ces besoins, les objectifs du projet sont les suivants :

- Concevoir une plateforme web dynamique pour toutes personnes s'intéressant au festival
- Sauvegarder les inscriptions (bénévoles, exposants, artistes, prestataires, personnes en situation de handicap) via des formulaires adaptés et les intégrer dans les bases de données.
- Intégration d'une PWA (Progressive Web App) pour offrir une application mobile aux festivaliers pendant le festival

4. Public cible et contexte d'utilisation prévue

Tout public participant au festival de quelque façon que ce soit (festivaliers, bénévoles, exposants, artistes, prestataires) mais aussi les membres administrateurs et référents de l'organisation.
La plateforme sera utilisée :

- Régulièrement par le public pour s'informer sur le festival (via desktop ou mobile), parfois avant d'avoir un ticket..
- Quotidiennement par l'équipe organisatrice pour gérer les inscriptions et suivre les données.
- Intensivement pendant le festival via une application mobile, permettant aux festivaliers de suivre les dernières infos en temps réel.

5. Intentions creatives

Le style visuel est inspiré des visuels existants créés par le graphiste de l'association. Le style de cette année a une couleur de fond foncée et un esprit nature et magique.
Sur le site destiné au public le fond sera bleu canard avec les textes blanc/beige, et la couleur d'accentuation sera jaune/oranger.
Sur le site destiné aux admin, mais aussi les formulaires, ce sera l'inverse: fond blanc/beige et texte bleu.
Ce contraste possède un bon ratio pour l'accessibilité: 8.25:1
https://webaim.org/resources/contrastchecker/?fcolor=EFE5D2&bcolor=01484D

Le thème choisi pour 2026 est l'équilibre, le site se rapprochera au maximum de ce thème.

La police du logo sera celle utilisée habituellement:
Büşşün

Les titres seront en KG second chances et les autres polices seront celle utilisée sur le flyer et celle utilisée sur le site actuel: Inter sans serif.

6. Intention de communication

Le ton employé sera amical et direct, avec un tutoiement assumé, pour renforcer l'esprit convivial et inclusif du festival.

7. Contraintes

Techniques:

- Interface responsive (mobile/tablette/desktop)
- Stockage sécurisé des données (base de données en ligne)

Temporelle:
Cette plateforme étant pour un réel projet, elle devra être prête et totalement fiable lors de sa présentation.

8. Motivation personnelle

Ce projet est avant tout né d'un besoin personnel : en tant que responsable des stands, artistes et prestataires, disposer d'un outil centralisé m'est essentiel.
Le rendre fonctionnel pour la promotion de l'édition 2026 serait pour moi une immense fierté : ce serait mon premier site web en production, né d'un vrai besoin, et au service de ma communauté.

9. Liste des compétences mis en oeuvre:

- Installer et configurer son environnement de travail en fonction du projet web ou web mobile
- Maquetter des interfaces utilisateur web ou web mobile
- Réaliser des interfaces utilisateur statiques web ou web mobile
- Développer la partie dynamique des interfaces utilisateur web ou web mobile
- Développer des composants d'accès aux données NoSQL
- Développer des composants métier côté serveur
- Documenter le déploiement d'une application dynamique web ou web mobile

CAHIER DES CHARGES

1. Informations générales

• Nom du projet : OKAMI CONNECT

• Date : De Juin 2025 à Janvier 2026

• Membres de l'équipe projet : Olivia Nanquette

2. Contexte du projet

Ce projet est né de mon engagement au sein d'une association qui organise un festival rassemblant plus de 2000 participants. En participant à la gestion de plusieurs aspects de l'organisation, j'ai identifié le besoin d'un outil numérique global permettant de faciliter et centraliser les tâches liées à l'événement. Par ailleurs, je souhaite que l'association puisse

devenir entièrement autonome dans la gestion de son site internet, afin de publier et mettre à jour les informations en toute simplicité, sans dépendre de prestataires extérieurs.

3. Objectifs du projet (formulation SMART)

Objectif 1 :
Mettre en place un site internet que je pourrai administrer intégralement: Clair.

Objectif 2 :
Développer un outil de gestion interne (artistes, bénévoles, partenaires, etc.) afin de faciliter l'organisation et la coordination des équipes en amont du festival: Atteignable

Objectif 3 :
Finaliser l'ensemble du projet (site public + outils de gestion) au plus tard en Janvier 2026, pour permettre une phase de tests et d'ajustements avant la prochaine édition du festival prévue en juin 2026: Temporel.

4. Utilisateurs cibles (personae)

Les festivaliers : grand public souhaitant accéder facilement aux informations essentielles du festival (programmation, billetterie, accès, actualités), via un site clair, à jour et responsive.
Les intervenants : artistes, prestataires, bénévoles et partenaires ayant besoin d'un espace pour transmettre leurs informations (fiche technique, disponibilité, besoins logistiques, etc.) de manière fluide et centralisée.
Les organisateurs : membres de l'association en charge de la coordination, ayant besoin d'un outil de gestion pour centraliser les données des intervenants, planifier les actions, et mettre à jour facilement le site.

5. Parties prenantes

Partie prenante	Rôle dans le projet
L'association	Acteur principal du projet : elle exprime les besoins fonctionnels, valide les grandes orientations, et teste les fonctionnalités tout au long du développement.
Moi, la développeuse	Cheffe de projet et développeuse : je conçois, développe et mets en œuvre l'ensemble du projet (site web + outils internes), en respectant les besoins exprimés. Je gère également toutes les phases du projet : cadrage, conception, développement, fonctionnelle et mise en production.

Le graphiste	Chargé de l'identité visuelle : il définit la charte graphique en lien avec la thématique du festival, et fournit les éléments visuels à intégrer au site.

6. Périmètre fonctionnel du projet

Livrables inclus et exclus de la première version du projet (MVP/V1).

Inclus :

- Création d'un **site internet responsive**, consultable sur ordinateurs, tablettes et smartphones
- Intégration de **formulaires interactifs** pour alimenter la base de données (artistes, bénévoles, partenaires, etc.)

Exclus :

- Développement d'une **application web mobile (PWA)** permettant un accès rapide aux informations du festival, même hors connexion
- Pas de système de création de compte utilisateurs via une page

GESTION DE PROJET

1. Méthodologie

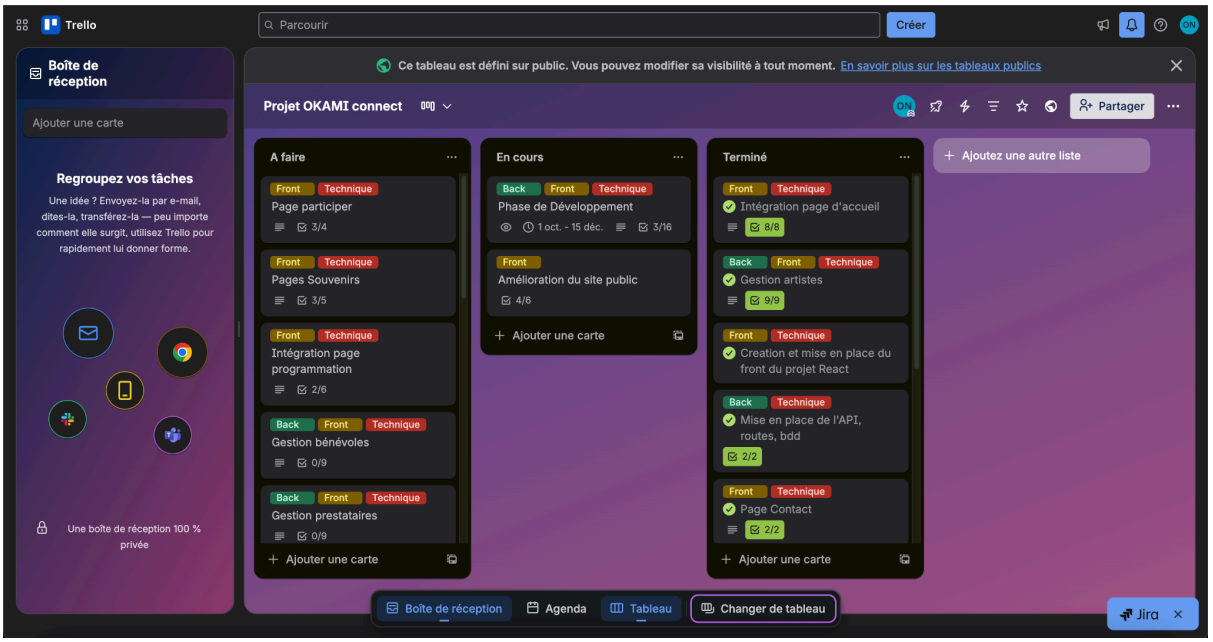
Pour ce projet, j'ai choisi d'adopter la méthodologie **AGILE**.
Même si elle met habituellement l'accent sur la collaboration en équipe, je trouve qu'elle reste parfaitement adaptée à ma situation. En effet, elle offre une grande souplesse et permet de s'ajuster facilement aux imprévus et aux évolutions. C'est exactement ce dont j'ai besoin, car mon projet avance en parallèle de mon apprentissage, et cette méthode m'accompagne dans cette progression.

2. Planning

Ce projet a été découpé en plusieurs phases classiques dans un projet digital qui a aidé à définir un planning afin de respecter les délais.

3. Outils utilisé

Pour m'aider à respecter ce planning, j'ai utilisé l'outil TRELLO qui me permet de structurer les différentes phases du projet de manière logique et ordonnée.
Cela inclut la définition des objectifs, la création d'un plan de travail détaillé et la mise en place de repères pour suivre les progrès.



Analyse et conception

1. Contexte et identification des besoins

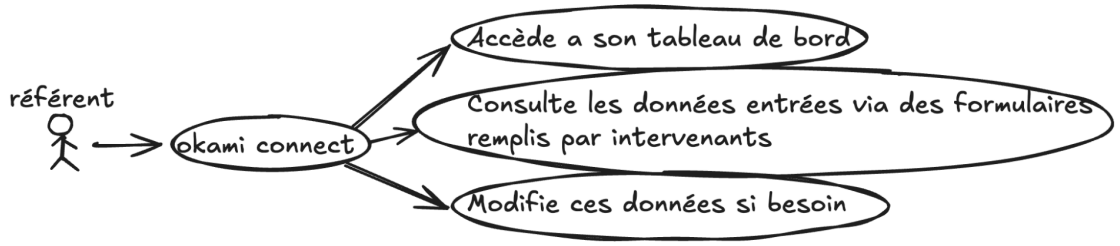
Pour analyser nos besoins, j'ai mis ma casquette de référente du pôle gestion des artistes, prestataires et stands.

J'ai constaté que nous travaillions sur des fichiers Excel distincts, souvent non partagés entre les pôles. Cela entraînait un manque de visibilité et des erreurs possibles dans les informations (doublons, oublis de mises à jour, etc.).

L'objectif principal du projet OKAMI Connect est donc de centraliser toutes les données dans une application web unique, accessible depuis un tableau de bord administrateur.

Cette plateforme permettra une gestion des utilisateurs par les admin, un stockage sécurisé des données (base de données en ligne), une mise à jour simplifiée des informations en temps réel, une meilleure communication entre les équipes et une vision globale des intervenants du festival (artistes, bénévoles, prestataires, stands, etc.)

2. Diagramme de cas d'utilisation



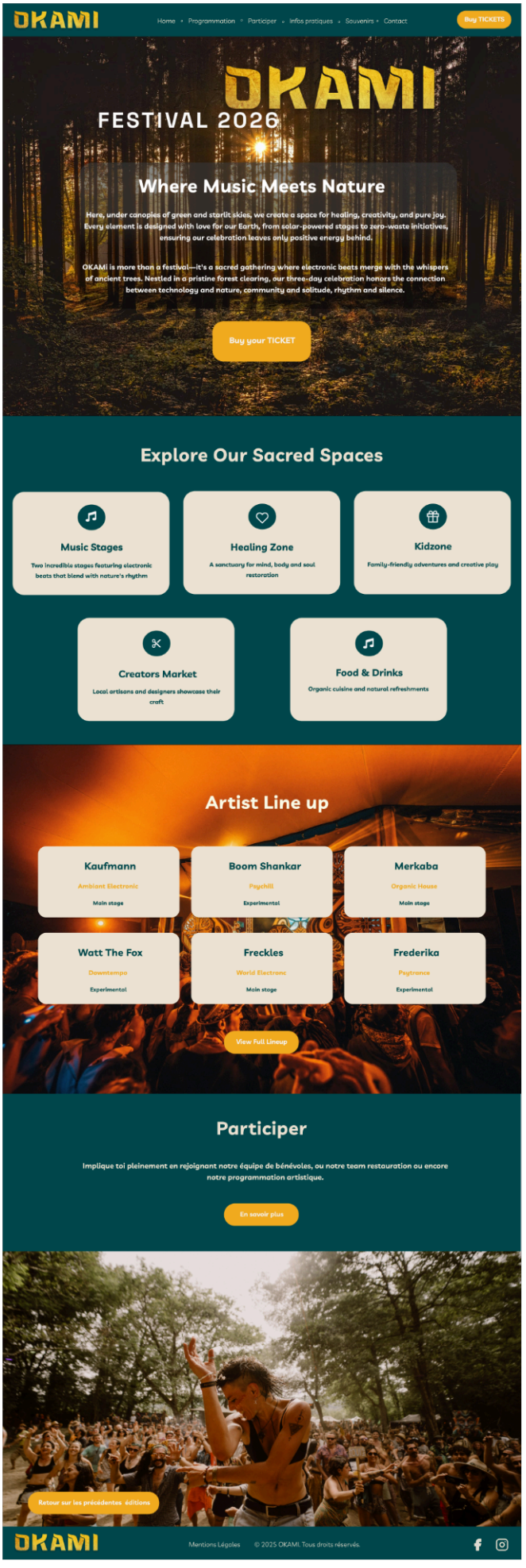
3. Choix techniques

Type de contrainte	Détail	Impact sur le projet
Langage Front-end	JavaScript / TypeScript + vite	Langage unique pour tout la stack: expérience utilisateur moderne
Framework Front-end	React	hooks, composants réutilisables, fetch API
Back-end	JavaScript (Node.js)	Création d'API REST pour communiquer avec la base de donnée, bonne performance pour applications en temps réel
Framework Back-end	Express js	Structure et organisation Écosystème riche
Base de données	MongoDB (NoSQL)	Base de données en ligne plus largement accessible

		pour gérer différentes entités (Bénévoles, artistes, exposant, prestataires..)
Gestion des tokens	JSON Web Tokens: (JWT) pour l'authentification	Sécurité
Accessibilité	Conformité RGAA : contrastes, navigation clavier, alt text	Test régulier, amélioration UX pour les personnes en situation de handicap
Sécurité	HTTPS validation des entrées	Sécurisation côté client et serveur, logique de gestion des accès
Navigateurs supportés	Chrome, Firefox, Safari, Edge	Nécessite des tests multi-navigateurs pour éviter les incompatibilités
Performance	Chargement optimisé des images, lazy loading	Réduction du temps de chargement, meilleure expérience utilisateur
Versionning	GitHub	Gestion du code et traçabilité

4. MAQUETTES ET WIREFRAMES

Maquettes haute fidélité


OKAMI
Home
Programmation
Participer
Infos pratiques
Souvenirs
Contact
Buy TICKETS
OKAMI
FESTIVAL 2026
Where Music Meets Nature
Here, under canopies of green and starlit skies, we create a space for healing, creativity, and pure joy. Every element is designed with love for our Earth, from solar-powered stages to zero-waste initiatives, ensuring our celebration leaves only positive energy behind.
OKAMI is more than a festival—it's a sacred gathering where electronic beats merge with the whispers of ancient trees. Nestled in a pristine forest clearing, our three-day celebration honors the connection between technology and nature, community and solitude, rhythm and silence.
Buy your TICKET
Explore Our Sacred Spaces
Music Stages
Two incredible stages featuring electronic beats that blend with nature's rhythm
Healing Zone
A sanctuary for mind, body and soul restoration
Kidzone
Family-friendly adventures and creative play
Creators Market
Local artisans and designers showcase their craft
Food & Drinks
Organic cuisine and natural refreshments
Artist Line up
Kaufmann
Ambiant Electronic
Main stage
Boom Shankar
Psychill
Experimental
Merkaba
Organic House
Main stage
Watt The Fox
Downtempo
Experimental
Freckles
World Electronic
Main stage
Frederika
Psytrance
Experimental
View Full Lineup
Participer
Implique toi pleinement en rejoignant notre équipe de bénévoles, ou notre team restauration ou encore notre programmation artistique.
En savoir plus
Retour sur les précédentes éditions
OKAMI
Mentions Légales
© 2025 OKAMI. Tous droits réservés.

Développement

I. Présentation des principales fonctionnalités développés

Front end

Le Front est séparé en 2 parties: Un site vitrine qui présente le festival et un site privé avec des formulaires personnalisés + une partie destinée aux administrateurs.
Il est développé en React 19 avec Vite 7 pour la compilation. Le projet utilise TypeScript, Sass pour la gestion des styles, React Router pour la navigation, et i18next pour la gestion multilingue.

<u>Composition du site public vitrine:</u>

- Accueil *https://okami-sigma.vercel.app/*
  - Header
    - Logo (page d'accueil)
    - Menu de navigation
    - Bouton ticket
    - Boutons de choix de langage
  - Section Héro (présentation)
    - Compte à rebours
  - Section Aftermovie
    - Vignette YouTube
  - Section présentant les différents espaces
    - Scène 1
    - Scène 2
    - Healing zone
    - Kidzone
    - Marché artisanale
    - Bar et restauration
  - Section Line up *redirige vers la page Programmation*
    - Carousel dynamique de vignettes d'artiste
  - Section participer
    - Bouton En savoir + *redirige vers la page Participer*
  - Section galerie
    - Bouton Voir + *redirige vers la page Souvenirs*
  - Footer

- Logo
- Mentions légales
- liens réseaux sociaux

- Programmation
  - Music
    - Vignettes artistes
  - Healing Zone
    - Vignettes intervenants healing
  - Kidzone
    - Vignettes intervenants kidzone
  - Performers & spectacles
    - Vignettes intervenants performers
  - Ateliers & activités
    - Vignettes intervenants
- Participer
  - Section Bénévolat
  - Section Stands
- Infos pratiques
  - Infos général
  - Partie FAQ
- Souvenirs
  - 2025
  - 2024
  - 2023
  - 2022
- Contact
  - Formulaire de contact général

<u>Composition des pages privées</u>

- Formulaires accessibles via son lien

- Page Login
- Dashboard
  - Artistes
    - Tableaux liste (creation et suppression possible)
    - Modification
  - Bénévoles
    - Tableaux liste (creation et suppression possible)
    - Modification
  - Stands

```
        ├─ Tableaux liste (creation et suppression possible)
        └─ Modification
├─ PMR
        ├─ Tableaux liste (creation et suppression possible)
        └─ Modification
└─ Prestataires
        ├─ Tableaux liste (creation et suppression possible)
        └─ Modification
```

BACK END

Le Back de l'application est développé en Node.js avec Express.
Voici les fonctionnalités principales:

- Système d'authentification : pour un accès aux données privées par les admin avec bcrypt et JWT
- CRUD : création, modification, suppression et consultation des fiches par type d'intervenants
- Envoi d'emails via Nodemailer par le formulaire de contact du front
- Connexion à la base de données : avec MongoDB via Mongoose
- Gestion des fichiers avec Multer et Cloudinary

LIAISON FRONT-BACK

Pour assurer la communication entre le front et le back, j'ai choisi d'utiliser **Axios**, une librairie JavaScript permettant d'effectuer facilement des requêtes HTTP (GET, POST, PUT, DELETE). Elle offre une syntaxe plus simple et lisible que fetch tout en gérant automatiquement certaines fonctionnalités comme les en-têtes, les erreurs ou la conversion des données JSON.

J'ai mis en place cette liaison à travers un fichier dédié (api.ts) qui centralise toutes les requêtes vers l'API, par exemple pour récupérer la liste des artistes ou envoyer un message via le formulaire de contact.
Cela permet d'avoir un code plus propre, organisé et réutilisable, car toutes les fonctions d'appel à l'API sont regroupées au même endroit au lieu d'être dispersées dans tout le projet.

Concrètement, Axios envoie les données du front vers le back via des requêtes HTTP . Le back traite ensuite ces données (via les routes programmées) et renvoie une réponse que le front affiche dynamiquement à l'utilisateur.

Ce système assure donc une communication fluide et sécurisée entre les deux parties de l'application, tout en facilitant la maintenance et les futures évolutions du projet.

2. Exemple de code commenté (voir annexes)

Front → page artists.tsx

Back → artistControllers.js

3. Gestion de la sécurité, accessibilité, responsive design, référencement et déploiement

Sécurité

Afin de mettre en place une application sécurisée, je me suis basée sur le top 10 de l'OWASP (Open Web Application Security Protect) qui est une communauté mondiale dédiée à la sécurisation des applications web (mis à jour en Novembre 2025)

A01: Broken Access Control

Problème d'accès à des données non autorisées.

Pour cela j'ai mis en place un middleware d'authentification et l'utilisation de json Web Token (JWT) sur le back

authMiddleware.js

```
const jwt=require('jsonwebtoken')

const authMiddleware=(req,res,next)=>{
   const authHeader=req.headers.authorization

   if(!authHeader || !authHeader.startsWith('Bearer')){
       return res.status(401).json({message: "Accès refusé. Token manquant"})
   }

   const token = authHeader.split(' ')[1]

   try{
       const decoded = jwt.verify(token, process.env.JWT_SECRET)
       req.user=decoded
       next()
   }catch(err){
       res.status(401).json({message: "Token invalide ou expiré"})
   }
}
```

```
module.exports = authMiddleware
```

A02: Security Misconfiguration

Sécuriser les en têtes HTTP avec Helmet pour protéger contre les attaques courantes vers les API

app.js

```
app.use(helmet({
 crossOriginResourcePolicy: { policy: 'cross-origin' }, // Permet Cloudinary
 contentSecurityPolicy: {
   directives: {
     defaultSrc: ["'self'"],
     imgSrc: ["'self'", 'data:', 'https://res.cloudinary.com'],
     scriptSrc: ["'self'"],
     styleSrc: ["'self'", "'unsafe-inline'"],
   },
 },
}))
```

A03: Software Supply Chain Failures

Cela correspond aux vulnérabilités qui proviennent des bibliothèques, packages ou composants tiers utilisés par l'application. Même si le code interne est propre, un package malveillant ou obsolète peut créer une faille.

Bonnes pratiques à appliquer régulièrement sur le back mais aussi le front:

Mettre à jour les packages (npm update), scanner les vulnérabilités (npm audit), vérifier la fiabilité des packages utilisés, limiter les dépendances inutiles.

A04: Cryptographic Failures

Problème de mot de passe stocké en clair

Pour ce projet j'ai installé *bcrypt* sur le back qui permet de hacher les mots de passe, puis de les comparer.

Code ajouté dans le schéma de modèle du système de connexion des admin pour le hachage:

```
adminSchema.pre('save',async function(next){
   if(!this.isModified('password')) return next()

   try{
       const salt=await bcrypt.genSalt(12)
       this.password=await bcrypt.hash(this.password,salt)
   }catch(err){
       next(err)
   }})
```

Code pour récupérer et comparer les mots de passe, et générer un token si c'est OK, dans le controller de connection des admin:

```
const  admin = await Admin.findOne({email})
       if(!admin) return res.status(401).json({message: "Cet email n'existe pas"})
             const isMatch = await bcrypt.compare(password, admin.password)
```

```
    if(!isMatch) return res.status(403).json({message:"Mot de passe invalide"})

    const token = jwt.sign(
        {
            id: admin.id, email: admin.email
        },
        process.env.JWT_SECRET,
        {expiresIn: '1d'}
    )
    res.json({token})
```

A05 Injections:
Problèmes d'injections via des entrées utilisateurs (URL)
Sur ce projet j'ai utilisé des requêtes paramétrées avec *mongoose* et ses controllers.
Mon système de connexion étant uniquement pour les admin qui sont des personnes que je connais, je ne trouve pas utile d'installer d'autres plugins comme express validator pour solidifier la sécurité.

A06: Insecure Design
Correspond aux failles qui apparaissent parce que l'application n'a pas été conçue avec la sécurité dès le départ. Sur mon API par exemple, les routes admin sont protégées par le middleware d'authentification, et le site utilise https en production.

A07: Authentication failures
Correspond aux failles ou le système ne vérifie pas correctement l'identité des utilisateurs, ce qui peut permettre à des attaquants de se connecter à la place d'autres personnes ou d'abuser de leurs comptes.
Ici l'admin se connecte avec son email + mot de passe. Ce mot de passe est stocké haché donc protégé.

A08: Software/Data integrity failures
Correspond aux failles ou l'application ou ses données peuvent être modifiées de manière malveillante, faute de vérification de leur intégrité.
Ici, toutes les données sont validées côté serveur avant d'être enregistrées en base, ce qui évite toute modification malveillante via un formulaire par exemple.

A09: Security Logging and Alerting Failures
Correspond aux failles ou l'application ne conserve pas ou ne surveille pas correctement les événements de sécurité, rendant difficile la détection et la réaction face aux attaques.
Les bonnes pratiques sont:
Loguer toutes les actions importantes côté serveur et y conclure des infos utiles, déclencher des alertes pour événements critiques, protéger les logs et surveiller régulièrement les logs.

A10: Mishandling of Exceptional Conditions

Correspond aux failles créées lorsque l'application ne gère pas correctement les erreurs ou situations inattendues, pouvant exposer des infos sensibles ou provoquer des plantages.
Tout au long de la construction de l'application, j'ai utilisé des try/catch pour parer à cela.

ACCESSIBILITÉ

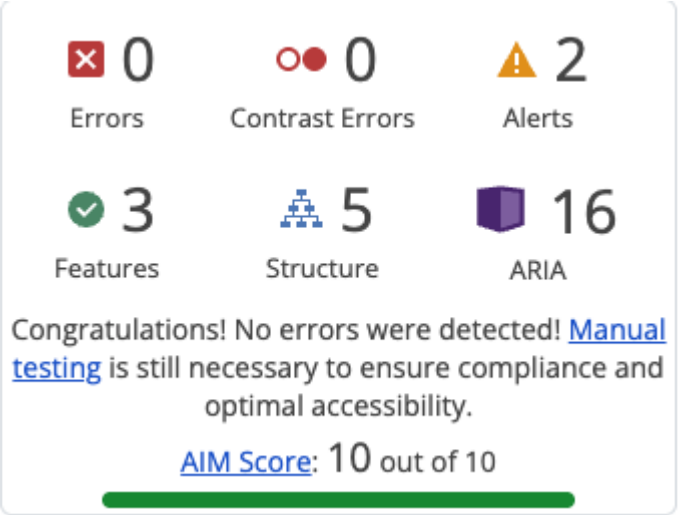
Tout le monde, y compris les personnes en situation de handicap visuels, auditifs, moteurs ou cognitifs doivent être en mesure d'utiliser l'application. Pour cela j'ai mis en place:

- Une structure de balises sémantiques HTML appropriées (header, nav, main, footer, button)
- Une hiérarchie logique des titres (h1, h2, h3…)
- Contraste texte/fond suffisant
- Mise en place d'aria-label pour les actions possibles
- Textes alternatifs pour toutes les images

Voici ce que j'aurais pu aussi mettre en place:

- Taille ajustable par l'utilisateur et utilisation de REM pour les espacements
- Utiliser tabindex

Score du site vérifier par https://wave.webaim.org/



RESPONSIVE DESIGN

Le responsive a été pensé tout au long du développement de l'application afin de faciliter certaines intégration.
J'ai intégré plusieurs breakpoints:

- Un a 1200px pour les différentes vues possibles sur un écran d'ordinateur
- Un a 1024px pour les tablettes
- Un a 768px pour les téléphones
- Un a 360px pour les toutes petites tailles d'écran de téléphone

Les grilles et images utilisent des flexbox pour s'adapter à toutes les tailles d'écran.
Le menu de navigation se transforme en hamburger sur tablette pour optimiser l'espace.
Boutons et zones cliquables suffisamment grands sur mobile.

L'application a été testée sur différents navigateurs et appareils pour garantir une expérience cohérente.

Référencement

Pour mon projet je ne souhaite pas de référencement, car ce n'est pas le site officiel du festival et que la partie admin n'a pas besoin d'apparaître dans les moteurs de recherche.
Pour cela, j'ai mis en place:

- un fichier robots.txt dans le dossier public:

  User-agent: *

  Disallow: /
- La balise suivante dans le <head>

```
<meta name="robots" content="noindex, nofollow">
```

- La balise <title> laissée vide.

Bonnes pratiques SEO pour les pages publiques:

- Recherche de mots clés pour optimiser le contenu et la visibilité
- Structure correctes des balises HTML
- Optimisation de l'expérience utilisateur
- Fichiers sitemap.xml et robots.txt pour guider les moteurs de recherche
- Balise <head> avec la balise <meta description> pour chaque page et JSON-LD pour l'ajout de données structurées.
- Décrire précisément les images dans leur texte alternatif

Déploiement

Mon application a été déployée sur deux plateformes distinctes :
Render pour le back (API) et **Vercel** pour le front.

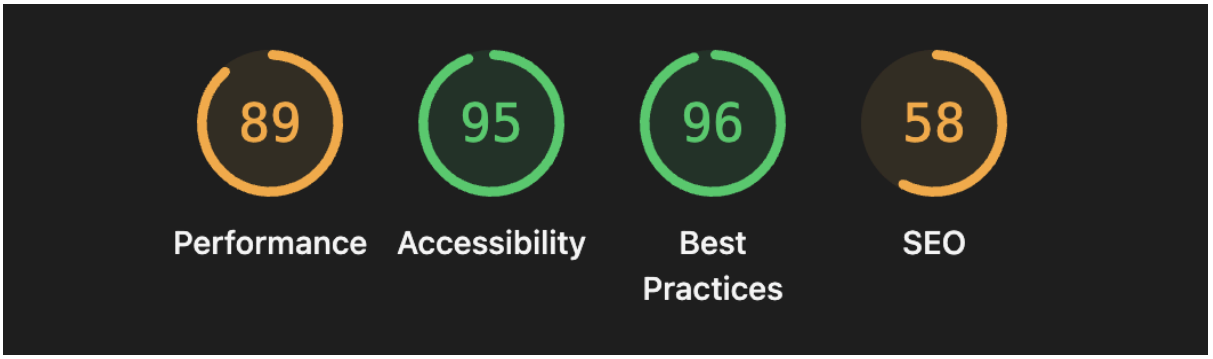
J'ai initialement testé le déploiement du back sur Vercel, mais cette plateforme impose des limitations de taille sur les fichiers uploadés ce qui m'a posé problème sur les formulaires d'inscription.
J'ai également expérimenté Railway, qui propose une offre gratuite, mais celle-ci est très limitée en ressources et peu adaptée à un usage continu.
Finalement, Render s'est avéré être la solution la plus adaptée car il permet de déployer une API Node.js complète, avec un serveur persistant, moins de contraintes sur la taille des fichiers et une configuration plus proche d'un environnement de production classique.

Le front de l'application a été déployé sur **Vercel**, une plateforme bien adaptée aux applications web modernes et simple d'utilisation.

Les procédures de déploiement sont rédigées dans des fichiers .md disponibles dans le chaque dossier du projet.

Post déploiement

A la suite du déploiement, une analyse light house est lancée:



Tests et validation

Difficultés rencontrées et solutions

Les tests ont été réalisés principalement sous forme de tests manuels fonctionnels, afin de vérifier le bon fonctionnement global de l'application côté front et back.
L'objectif était de s'assurer que les fonctionnalités principales répondent correctement, les données sont correctement échangées entre le front et l'API et que l'application reste stable après déploiement.

Outils utilisés

- Navigateur web (Chrome) et outils de développement (console, réseau)
- Postman pour tester les routes de l'API (GET, POST)
- Lighthouse pour analyser les performances, l'accessibilité et les bonnes pratiques
- Logs serveur (Render) pour vérifier le bon démarrage et le comportement de l'API

Tests réalisés

- Test des formulaires (envoi des données, validations, erreurs)
- Vérification de l'affichage des données dynamiques
- Test des routes de l'API et des réponses JSON
- Test des accès administrateur
- Vérification du comportement responsive sur différents formats d'écran
- Tests après déploiement pour vérifier le bon fonctionnement en production

Correction apportées

A la suite des tests, plusieurs corrections ont été effectuées:

- correction de problèmes d'affichage liés au responsive design
- amélioration de la gestion des erreurs côté back
- sécurisation de l'API via l'ajout de headers HTTP (Helmet)
- optimisation du chargement des images (format AVIF, hébergement Cloudinary)
- ajustement des règles CORS pour éviter les erreurs de communication entre le front et le back

Conclusion

Difficultés rencontrées et solutions

Tout au long du développement de l'application, de nombreux problèmes techniques ont été rencontrés. Il est difficile de tous les détailler, certains ayant été corrigés progressivement au cours du projet.
Les principales difficultés sont apparues après le déploiement de l'application. Plusieurs fonctionnalités qui fonctionnaient correctement en environnement local se sont comportées différemment une fois mises en production.
Des ajustements importants ont notamment été nécessaires et les problèmes ont été résolus grâce à l'analyse des erreurs affichées dans la console et les logs serveur, des tests répétés en production, des modifications de configuration et de code, et une phase importante de recherche et d'expérimentation afin de comprendre le comportement réel de l'application une fois déployée.
Ces difficultés ont permis de mieux appréhender les différences entre un environnement de développement local et un environnement de production, et ont représenté une étape clé dans l'apprentissage du déploiement d'une application web. que je n'avais absolument pas vu en cours.

Bilan et perspectives

Ce projet m'a permis d'acquérir une expérience complète sur un site web réel: développement du front et du back, gestion des bases de données, déploiement, et mise en place de mesures de sécurité.
Le projet continue d'évoluer avec une nouvelle version déjà en production déployée cette fois sur un VPS qui permet une meilleure performance générale. De nouvelles pages et fonctionnalités ont été créées pour enrichir l'expérience utilisateur globale. J'espère que cette plateforme deviendra le prochain site officiel du festival pour l'édition 2027.

Annexes

Extraits de code commentés:

Backend, avec exemple de controllers pour gérer les artistes (CRUD complet)

```
// Récupérer les artistes validés pour la page publique
export const getPublicArtists = async (req, res) => {
   try {
       // avec filtre des artistes validés
       // un tri insensible à la casse et aux accents (utile pour noms français)
       // un tri alphabétique croissant
       const artists = await Artist.find({ isValidated: true })
       .collation({ locale: 'fr', strength: 2 })
       .sort({ projectName: 1 })

       if (!artists) return res.status(404).json({ message: 'Aucun artiste validé
trouvé' })

       // Retourne les données en JSON au front
       res.json(artists)
   } catch (error) {
       // try/catch : capture les erreurs serveur et renvoie un code 500
       res.status(500).json({ message: error.message })
   }
}

// Créer ou mettre à jour un artiste via le formulaire public
export const createOrUpdateArtist = async (req, res) => {
   try {
       // Nettoyage des champs texte pour éviter doublons
       ['projectName','firstName','lastName'].forEach(f => {
       if (req.body[f]) req.body[f] = req.body[f].trim()
       })

       // Upload de la photo promo si présente
       if (req.files?.promoPhoto) {
       // req.files.promoPhoto[0].buffer : contient le fichier en mémoire (buffer)
       // upload vers Cloudinary pour hébergement en ligne
       const result = await uploadImageToCloudinary(
           req.files.promoPhoto[0].buffer,
           'PromoPhoto',       // dossier Cloudinary
           req.body.projectName // nom du fichier
       )
       req.body.promoPhoto = result.secure_url // on stocke l'URL sécurisée en base
```

```
        }

        // findOneAndUpdate avec upsert: true
        // - Si l'artiste existe déjà, on le met à jour
        // - Sinon, on le crée
        // runValidators: true => applique les validations définies dans le schéma
Mongoose
        const artist = await Artist.findOneAndUpdate(
        { projectName: req.body.projectName },
        req.body,
        { new: true, runValidators: true, upsert: true, collation: { locale: 'fr',
strength: 2 } }
        )

        res.status(200).json(artist)
    } catch (error) {
        console.error('Erreur createOrUpdateArtist:', error)
        res.status(400).json({ message: error.message })
    }
}

// Suppression d'un artiste (route admin)
export const deleteArtist = async (req, res) => {
    try {
        // Récupère l'identifiant passé dans l'URL (peut être l'ID MongoDB ou le nom
du projet)
        const { id } = req.params

        // Essaie d'abord de supprimer l'artiste par son ID MongoDB
        // Si ça échoue (ex: ID invalide), on retourne null grâce au .catch(() =>
null)
        // Ensuite, si aucun artiste trouvé par ID, on essaie de supprimer par le
nom du projet
        const artist = await Artist.findByIdAndDelete(id).catch(() => null) ||
                        await Artist.findOneAndDelete({ projectName: id })

        // Si aucun artiste trouvé dans la base de données, on renvoie une erreur
404
        if (!artist) return res.status(404).json({ message: 'Artiste non trouvé' })

        // Si l'artiste a été trouvé et supprimé, on renvoie un message de
confirmation
        res.json({ message: 'Artiste supprimé' })
    } catch (error) {
```

```
        // En cas d'erreur serveur inattendue, renvoie un code 500 avec le message
de l'erreur
        res.status(500).json({ message: error.message })
    }
}
```

Frontend avec la page admin qui récupère tous les artistes et leurs infos

```
// Import nécessaire à la page
import { useEffect, useState } from "react"
import Button from "../../components/Button"
import Modal from "../../components/Modal"
import { AdminHeader } from "../../components/AdminHeader"
import type { Artist } from "../../types/Artist"
import { artistApi } from "../../services/api"

export default function ArtistPage() {
 // Gestion des états pour stocker des données
 const [artists, setArtists] = useState<Artist[]>([]) // Artistes récupérés depuis
l'API
 const [filteredArtists, setFilteredArtists] = useState<Artist[]>([]) // Liste
filtrée selon la recherche (affichée à l'écran)
 const [searchTerm, setSearchTerm] = useState('') // Texte tapé dans la barre de
recherche
 const [loading, setLoading] = useState(true) // Chargement en cours ou terminé
 const [error, setError] = useState<string | null>(null) // Stocke le message
d'erreur
 const [selectedArtist, setSelectedArtist] = useState<Artist | null>(null) //
Artiste sélectionné pour afficher le panneau latéral
 const [showDeleteModal, setShowDeleteModal] = useState(false) // modal de
suppression
 const [showConfirmationModal, setShowConfirmationModal] = useState(false) //modal
de confirmation de suppression
 const [artistToDelete, setArtistToDelete] = useState<string | null>(null) // ID de
l'artiste à supprimer

 // Fonction pour formater les numéros de téléphone
 const formatPhone = (phone: string) => {
   if (!phone) return "" // Si vide, retourne vide
   // Retire espaces, tirets, points, parenthèses
   const clean = phone.replace(/[\s\-().]/g, "")
   // Groupe les chiffres par 2 avec un espace
   return clean.replace(/(\d{2})/g, "$1 ").trim()
 }
```

```
// Fonction qui récupère la liste d'artiste
const fetchArtists = async () => {
  try {
    setLoading(true)
    const datas = await artistApi.getAll()

    setArtists(datas) // Stocke la liste complète
    setFilteredArtists(datas) // Initialise la liste filtrée avec tous les
artistes
  } catch (error) {
    const err = error as Error
    setError(err.message)
  } finally {
    setLoading(false)
  }
}

// Fonction pour appeler la modal de confirmation de suppression
const confirmDelete = (id: string) => {
  setArtistToDelete(id) // Stocke l'ID de l'artiste à supprimer
  setShowDeleteModal(true)
}

// Fonction pour supprimer l'artiste
const deleteArtist = async () => {
  if (!artistToDelete) return // Si pas d'ID, on arrête la fonction

  try {
    await artistApi.delete(artistToDelete)

    // Mise à jour de l'état local (sans recharger la page)
    setArtists((prev) => prev.filter((artist) => artist._id !== artistToDelete))
// Filtre pour garder tous les artistes SAUF celui supprimé

    setSelectedArtist(null)
    setShowDeleteModal(false)
    setShowConfirmationModal(true)
    setArtistToDelete(null)
  } catch (error) {
    const err = error as Error
    setError(err.message)
  }
}

useEffect(() => {
```

```
    fetchArtists() // UseEffect qui récupère les artistes dès que la page s'affiche
  }, [])

  useEffect(() => {
    // Filtre les artistes dont le nom contient le texte recherché
    const filtered = artists.filter(artist =>
      (artist.projectName || '').toLowerCase().includes(searchTerm.toLowerCase()) //
Comparaison en minuscules
    )
    setFilteredArtists(filtered) // Met à jour la liste affichée
  }, [searchTerm, artists]) // Dépendances : se déclenche quand searchTerm ou
artists change


  if (loading) return <p>Chargement des artistes en cours…</p>
  if (error) return <p style={{ color: "red" }}>{error}</p>

  // Si pas de chargement ni d'erreur, on affiche la page complète
  return (
    <>
    <AdminHeader />
    <main className="artists-page">
      <div className="main-wrap">
      <h1>Artistes 2026</h1>

      <div className="actions-bar">
        <input
          type="text"
          placeholder="Rechercher un artiste..."
          value={searchTerm} // Valeur affichée = état searchTerm
          onChange={(e) => setSearchTerm(e.target.value)} // À chaque frappe, met à
jour searchTerm
          className="search-input"
          id="search"
          name="search"
        />
        <Button to={("/admin/artists/new")} className="btn add-btn">+ Ajouter un
artiste</Button>
      </div>


{/* Tableau desktop */}
      <div className="table-list desktop-view">
        <table>
          <thead>
```

```
          <tr>
            <th>PROJET</th>
            <th>NOM</th>
            <th>PRÉNOM</th>
            <th>TEL</th>
            <th>Nb de pers</th>
            <th>SCÈNE</th>
            <th>Date/heure de jeu</th>
          </tr>
        </thead>
        <tbody>
          {filteredArtists.map((artist) => (
            <tr //ligne
              key={artist._id} // key unique obligatoire
              onClick={() => setSelectedArtist(artist)} // Au clic on stocke
l'artiste dans selectedArtist
              className="clickable-row"
            >
              <td><strong>{artist.projectName || "-"}</strong></td>
              <td>{artist.lastName || "-"}</td>
              <td>{artist.firstName || "-"}</td>
              <td>
                {artist.phone ? (
                  <a href={`tel:${artist.phone}`}>{formatPhone(artist.phone)}</a>
// Lien cliquable pour appeler
                ) : "-"}
              </td>
              <td>{artist.numberOfPeople || "-"}</td>
              <td>{artist.stage || "-"}</td>
              <td>{artist.performanceDateTime || "-"}</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>

{/* Cards mobile/tablette */}
    <div className="cards-list mobile-view">
      {filteredArtists.map((artist) => (
        <div
          key={artist._id}
          className="cards"
          onClick={() => setSelectedArtist(artist)}
        >
          <h3>{artist.projectName || "-"}</h3>
```

```
          <p><strong>Nom :</strong> {artist.lastName || "-"}</p>
          <p><strong>Prénom :</strong> {artist.firstName || "-"}</p>
          <p><strong>Email :</strong> {artist.email || "-"}</p>
          <p><strong>Tél :</strong> {artist.phone ? <a
href={`tel:${artist.phone}`} onClick={(e) =>
e.stopPropagation()}>{formatPhone(artist.phone)}</a> : "-"}</p>
          <p><strong>Nb de pers :</strong> {artist.numberOfPeople || "-"}</p>
          <p><strong>Scène :</strong> {artist.stage || "-"}</p>
          <p><strong>Date/heure de jeu :</strong> {artist.performanceDateTime ||
"-"}</p>
        </div>
      ))}
    </div>

{/* Panneau latéral  */}
    {selectedArtist && (
      <>
        <div className="overlay" onClick={() => setSelectedArtist(null)} />
        <aside className="side-panel">
          <button className="close" onClick={() => setSelectedArtist(null)}
aria-label="Fermer">✕</button>
          {/* Affiche la photo uniquement si elle existe */}
          {selectedArtist.promoPhoto && (
            <img
              src={selectedArtist.promoPhoto}
              alt={selectedArtist.projectName}
              className="artist-photo"
            />
          )}
          <h2>{selectedArtist.projectName}</h2>
          <p id='email'>{selectedArtist.email || "-"}</p>
          <div className="tableBtns">
            <Button type="button" className="btn"
to={`/admin/artist-edit/${selectedArtist._id}`}>Modifier</Button>

            <Button type="button"  className="btn btn-delete" onClick={() =>
confirmDelete(selectedArtist._id)}>Supprimer</Button>
          </div>
          {/* Liste des infos détaillés de l'artiste sélectionné */}
          <h3>Logistique</h3>
          <p><strong>Run arrivée :</strong>{selectedArtist.arrivalRun || "-"}</p>
          <p><strong>Run départ :</strong>{selectedArtist.departureRun || "-"}</p>
          <p><strong>Logement :</strong>{selectedArtist.accommodation || "-"}</p>

          <h3>Technique</h3>
```

```
          <p><strong>Matériel Setup :</strong> {selectedArtist.setup || "-"}</p>
          <p><strong>Temps Setup :</strong> {selectedArtist.setupTime || "-"}</p>
          <p><strong>Soundcheck :</strong> {selectedArtist.needsSoundcheck ? 'Oui'
: 'Non'}</p>
          <p><strong>Soundcheck date et heure:</strong>
{selectedArtist.soundcheckDateTime || "-"}</p>
          <p><strong>Accord pour enregistrer la prestation :</strong>
{selectedArtist.canRecordSet ? 'Oui' : 'Non'}</p>
          <p><strong>Rider technique :</strong></p>
            {selectedArtist.riderTechUrl ? (
              // Cas 1 : L'artiste a fourni une URL
              <a
                href={selectedArtist.riderTechUrl}
                target="_blank"
                rel="noopener noreferrer"
                style={{ fontSize: "0.85rem",color: "#666"}}
              >
                Ouvrir le rider technique (URL)
              </a>
            ) : selectedArtist.riderTechUpload ? (
              // Cas 2 : Un fichier a été uploadé
                <p>Fichier uploadé dispo dans le Kdrive</p>
            ) : (
              // Cas 3 : Rien du tout
              <span>-</span>
            )}

          <p><strong>Commentaire artiste :</strong> {selectedArtist.comments ||
"-"}</p>


          <h3>Administratif</h3>
          <p><strong>Infos admin : </strong>{selectedArtist.specialInfo ||
"-"}</p>
          <p><strong>Nom invité :</strong>{selectedArtist.guestName || "-"}</p>
          <p><strong>Contrat :</strong> {selectedArtist.contract || "-"}</p>
          <p><strong>Feuille de route :</strong> {selectedArtist.roadmap ||
"-"}</p>
          <p><strong>Fiche Sacem :</strong> {selectedArtist.sacemForm || "-"}</p>

          <p><strong>Infos paiement :</strong> {selectedArtist.paymentInfo ||
"-"}</p>
          <p><strong>Facture :</strong> {selectedArtist.invoice || "-"}</p>
          <p><strong>Cachet :</strong> {selectedArtist.fee || "-" } €</p>
```

```
          <p><strong>Frais déplacement :</strong> {selectedArtist.travelExpenses
|| "-"} €</p>
          <p><strong>Total TTC :</strong> {selectedArtist.totalTTC || "-"}</p>

          <h3>Promo</h3>
          <p><strong>Style :</strong> {selectedArtist.musicalStyle || "-"}</p>
          <p><strong>Réseaux sociaux :</strong></p>
          <ul>
            {(() => {
            // Étape 1 : Transformation des données si nécessaire
            // Les réseaux sociaux peuvent arriver sous 2 formats différents :
            // - Soit comme un objet JavaScript : { instagram: "...", soundcloud:
"..." }
            // - Soit comme du texte JSON :
'{"instagram":"...","soundcloud":"..."}'
            // On vérifie le type et on convertit en objet si besoin
            const links = typeof selectedArtist.socialLinks === 'string'
              ? JSON.parse(selectedArtist.socialLinks) // Convertit le texte en
objet
              : selectedArtist.socialLinks; // Déjà un objet, on le garde

            // Étape 2 : Liste de tous les réseaux sociaux qu'on veut afficher
            const socialFields = ['instagram', 'soundcloud', 'spotify',
'facebook', 'website', 'youtube'];

            // Étape 3 : Vérifie si au moins UN réseau social existe
            // some() retourne true si au moins un élément du tableau remplit la
condition
            const hasAnyLink = socialFields.some(field => links?.[field]);

            return (
              <>
                {hasAnyLink ? (
                  // Si au moins un lien existe, on parcourt tous les réseaux
                  socialFields.map((field) =>
                    links?.[field] ? ( // Si ce réseau spécifique a un lien
                      <li key={field}>
                        <a href={links[field]} target="_blank" rel="noopener
noreferrer">
                          {links[field]}
                        </a>
                      </li>
                    ) : null // Si pas de lien pour ce réseau, on n'affiche rien
                  )
                ) : (
```

```
                  // Si aucun lien n'existe, on affiche juste un tiret
                  <li>-</li>
                )}
              </>
            );
          })()}
          </ul>
          <p><strong>Statut :</strong> {selectedArtist.isValidated ? '✅ Validé' :
'❌ Non validé'}</p>

          <h3>Métadonnées</h3>
          <p><strong>Source des données :</strong> {selectedArtist.dataSource ||
'-'}</p>
          <p><strong>Dernière modification par :</strong>
{selectedArtist.lastModifiedBy || '-'}</p>
          <p><strong>Créé le :</strong> {selectedArtist.createdAt ? new
Date(selectedArtist.createdAt).toLocaleString('fr-FR') : '-'}</p>
          <p><strong>Modifié le :</strong> {selectedArtist.updatedAt ? new
Date(selectedArtist.updatedAt).toLocaleString('fr-FR') : '-'}</p>

          <div className='close-btn'>
            <Button onClick={() => setSelectedArtist(null)}
className="btn">Fermer</Button>
          </div>
        </aside>
      </>
    )}

    {/* Modal de confirmation de suppression si showDeleteModal = true */}
    {showDeleteModal && (
      <Modal
        text="Veux-tu vraiment supprimer cet artiste de la base de données ?"
        onConfirm={deleteArtist} // Fonction appelée si on clique sur "Confirmer"
        onClose={() => { // Fonction appelée si on clique sur fermer
          setShowDeleteModal(false) // Ferme le modal
        }}
      />
    )}

    {showConfirmationModal && (
      <Modal text="L'artiste a bien été supprimé"
      onClose={()=>{
        setArtistToDelete(null) // Réinitialise l'ID
        setShowConfirmationModal(false)
      }}
```

```
        />
      )}
      </div>
    </main>
    </>
  )
}
```

Merci de m’avoir lu!