Documentation v0.3 – Gestion des demandes et les recherche optimisés

Sommaire

 Introduction . . . 2
 Grandes Fonctionnalités . . . 2
  2.1 Gestion des Demandes d’Intérêt & Notifications en Temps Réel . . . 3
  2.2 Recherche de Propositions . . . 4
  2.3 Recherche Avancée des Événements . . . 5
  2.4 Modification et Suppression des Événements . . . 7
 Petites Fonctionnalités . . . 8
  2.1 Page “Mes Demandes Envoyées” . . . 8
  2.2 Notifications en Temps Réel . . . 9
  2.3 Détails d’un Événement . . . 9
  2.4 Gestion des Images des Événements (Frontend) . . . 10
  2.5 Filtre par Villes . . . 10
  2.6 Améliorations UX/UI . . . 11
 Impact Métier & Valeur Ajoutée . . . 11
 Tests & Validation . . . 12
Documentation v1.0 – Signalement de dangers . . . 12
 Introduction . . . 12
 Fonctionnalités . . . 12
 Workflow du signalement de danger . . . 13
 Détails Techniques . . . 14
 Illustrations . . . 14
 Comparaison avec les plateformes existantes . . . 14
 Pourquoi nous sommes innovants ? . . . 15
 Tests réalisés . . . 15
 Conclusion . . . 16
 Documentation v1.0 – Gestion des Projets . . . 16
  Description . . . 16
  Flux Utilisateur . . . 16
  Endpoints Backend . . . 17
  Diagramme de Séquence : Gestion des Projets . . . 18
  Impact Métier & Valeur Ajoutée . . . 19
  Tests & Validation . . . 19

  Conclusion . . . 19
Documentation Technique – Release v1.1 - DevOps2 . . . 20
Objectif de la fonctionnalité . . . 20
 Nouveauté côté utilisateur . . . 20
 Fonctionnalités livrées . . . 20
Authentification Google (OAuth2) – Intégration initiale . . . 21
Objectif . . . 21
Fonctionnalités livrées . . . 21
Implémentation technique . . . 21
Migration SQL associée . . . 22
Configuration . . . 22
Intégration Google Calendar – Détails techniques . . . 22
1. Authentification OAuth . . . 22
2. Affichage des événements Google (dashboard) . . . 22
3. Création d'événement à la participation . . . 23
4. Backend (routes Node/Express) . . . 23
Architecture technique . . . 23
Problèmes rencontrés . . . 24
Bénéfices de cette feature dans la stratégie globale de notre application . . . 24
Tests réalisés . . . 24
Release v1.1 – Récapitulatif . . . 24

Introduction

La version v0.3 marque une avancée majeure pour l'application en intégrant un **système de notifications en temps réel**, une **gestion fluide des demandes d'intérêt**, et une **expérience utilisateur optimisée**.

Cette documentation couvre :

1. Les nouvelles fonctionnalités développées.
2. Les **flux utilisateurs** pour chaque action clé.
3. Les **endpoints backend** utilisés.
4. L'**impact métier** et la valeur ajoutée des améliorations.
5. Un **diagramme de séquence UML** pour illustrer le workflow.

Grandes Fonctionnalités

🔹 2.1 Gestion des Demandes d’Intérêt & Notifications en Temps Réel

Description

Cette fonctionnalité regroupe tout le flux des demandes d’intérêt, y compris les notifications en temps réel. Les utilisateurs peuvent envoyer et recevoir des demandes d’intérêt, puis être notifiés de l’acceptation ou du refus de la demande en temps réel.

Flux Utilisateur

1. **Envoi d’une Demande d’Intérêt**
   - L’utilisateur intéressé clique sur "Demander".
   - La demande est enregistrée en base (via **POST /interests**).
   - Une notification en temps réel est envoyée au proposeur.
2. **Consultation des Demandes d’Intérêt**
   - Le proposeur accède à "Mes Intérêts Reçus".
   - Il voit la demande et peut l’accepter ou la refuser.
3. **Notifications et Actions en Temps Réel**
   - Si la demande est acceptée, le demandeur est notifié avec les coordonnées du proposeur.
   - Si la demande est refusée, le demandeur est informé de la décision.

Endpoints Backend

Méthode	Endpoint	Description
POST	`/interests`	Créer une demande d’intérêt.
GET	`/interests/received/:userId`	Récupérer les demandes reçues.
GET	`/interests/sent/:userId`	Récupérer les demandes envoyées.
PUT	`/interests/:id`	Accepter ou refuser une demande.
GET	`/notifications/:userId`	Récupère toutes les notifications d’un utilisateur.
POST	`/notifications`	Crée une nouvelle notification.
DELETE	`/notifications/:notifId`	Supprime une notification spécifique.
DELETE	`/notifications/all/:userId`	Supprime toutes les notifications d’un utilisateur.

Diagramme de Séquence : Demande d’Intérêt et Notifications

```
@startuml
participant "Utilisateur Intéressé (par l'annonce)" as UI
participant "Frontend (React)" as FE
participant "Backend API" as API
participant "Base de Données" as DB
participant "WebSockets" as WS
participant "Proposeur (de l'annonce)" as P

== 1. L'utilisateur envoie une demande d'intérêt ==
UI -> FE: Clique sur "Demander"
FE -> API: **POST** /interests (proposition_id, interested_user_id)
API -> DB: Vérifie que la proposition existe
DB --> API: OK
API -> DB: Enregistre la demande avec statut **"pending"**
DB --> API: OK (id_interet)
API -> WS: **Émet une notification au proposeur**
WS --> P: "Nouvelle demande reçue"

== 2. Le proposeur consulte ses demandes ==
P -> FE: Accède à "Mes Intérêts Reçus"
FE -> API: **GET** /interests/received/:userId
API -> DB: Récupère toutes les demandes associées à l'utilisateur
DB --> API: Renvoie les demandes (id, titre, utilisateur intéressé)
API --> FE: Affiche la liste des demandes

== 3A. Le proposeur **accepte** la demande ==
P -> FE: Clique sur "**Accepter**"
FE -> API: **PUT** /interests/:id (status: accepted)
API -> DB: Met à jour le statut en **"accepted"**
DB --> API: OK
API -> WS: **Émet une notification avec le statut accepté**
WS --> UI: "** Votre demande a été acceptée ! Voici les contacts **"

== 3B. Le proposeur **refuse** la demande ==
P -> FE: Clique sur "**Refuser**"
FE -> API: **PUT** /interests/:id (status: rejected)
API -> DB: Met à jour le statut en **"rejected"**
DB --> API: OK
API -> WS: **Émet une notification avec le statut refusé**
WS --> UI: "** Votre demande a été refusée.**"
@enduml
```

2.2 Recherche de Propositions

Description

Cette fonctionnalité permet aux utilisateurs de rechercher des propositions en fonction de

plusieurs critères : mots-clés, catégorie et distance géographique.

Flux Utilisateur

1. L'utilisateur entre des mots-clés et sélectionne une catégorie de service.
2. Le système effectue une recherche floue sur les titres et descriptions des propositions.
3. Le système filtre les propositions par catégorie sélectionnée.
4. Le système calcule la distance géographique entre l'utilisateur et les propositions.
5. Les résultats sont affichés, triés par proximité géographique.

Endpoints Backend

Méthode	Endpoint	Description
GET	`/propositions/search`	Recherche des propositions en fonction des mots-clés, catégorie et distance.

Diagramme de Séquence : Recherche de Propositions

```
@startuml
actor "Utilisateur" as User
participant "Frontend (React)" as FE
participant "Backend API" as API
participant "Base de Données" as DB
participant "Fuse.js" as Fuse
participant "WebSocket (si notifications)" as WS

== 1. L'utilisateur effectue une recherche ==
User -> FE: Entre des mots-clés et sélectionne une catégorie
FE -> API: **GET** /propositions/search (mots-clés, catégorie, utilisateur_id)
API -> DB:  Récupère les propositions en fonction de la catégorie
DB --> API:  Liste des propositions filtrées par catégorie
API -> Fuse: Utilise Fuse.js pour recherche floue sur 'title' et 'description'
Fuse --> API:  Liste des propositions correspondant aux mots-clés
API -> DB:  Récupère les coordonnées de l'utilisateur (latitude, longitude)
DB --> API:  Coordonnées de l'utilisateur
API -> DB:  Calcule la distance entre l'utilisateur et chaque proposition
DB --> API:  Liste des propositions avec distances
API -> FE:  Affiche les résultats avec distance et pertinence
FE --> User: Montre les propositions filtrées

@enduml
```

2.3 Recherche Avancée des Événements

Description

Cette fonctionnalité permet aux utilisateurs de rechercher des événements en fonction de plusieurs critères : mots-clés, catégorie et ville. Grâce à la bibliothèque **Fuse.js**, la recherche est floue et permet de retrouver des événements qui correspondent partiellement aux mots-clés recherchés, même en cas d'erreur de frappe.

Le processus de recherche est optimisé pour une expérience utilisateur fluide :

1. L'utilisateur saisit un mot-clé (et optionnellement, sélectionne une catégorie ou une ville).
2. Le système filtre les événements en fonction de la catégorie et de la ville sélectionnées.
3. La recherche floue est effectuée sur les titres et descriptions des événements en utilisant Fuse.js, avec un seuil de pertinence réglable pour affiner les résultats.
4. Les résultats sont retournés et triés par pertinence.

Flux Utilisateur

1. L'utilisateur entre un mot-clé de recherche et, si souhaité, sélectionne une catégorie et/ou une ville.
2. La recherche floue est effectuée dans les titres et descriptions des événements.
3. Les événements sont filtrés en fonction de la catégorie et de la ville, si spécifiés.
4. Les résultats de recherche sont retournés, affichés par pertinence.
5. L'utilisateur peut cliquer sur un événement pour consulter son détail.

Endpoints Backend

Méthode	Endpoint	Description
GET	`api/events/search`	Recherche des événements en fonction des mots-clés, catégorie et ville.
GET	`api/events/:id`	Récupère les détails d'un événement spécifique.

Diagramme de Séquence : Recherche Avancée des Événements

```
@startuml
actor "Utilisateur" as User
participant "Frontend (React)" as FE
participant "Backend API" as API
participant "Base de Données" as DB
participant "Fuse.js" as Fuse

== 1. L'utilisateur effectue une recherche ==
User -> FE: Saisit un mot-clé et sélectionne une catégorie ou une ville
FE -> API: **GET** api//events/search (mot-clé, catégorie, ville)
API -> DB:  Récupère tous les événements en fonction de la catégorie et de la ville
DB --> API:  Liste des événements filtrés
API -> Fuse: Recherche floue sur 'title' et 'description'
```

```
Fuse --> API:  Liste des événements correspondant aux mots-clés
API -> FE:  Affiche les résultats de la recherche
FE --> User: Montre les événements filtrés par pertinence

== 2. L'utilisateur consulte un événement ==
User -> FE: Clique sur un événement
FE -> API: **GET** api/events/:id
API -> DB:  Récupère les détails de l'événement avec l'ID
DB --> API:  Détails de l'événement
API -> FE:  Affiche les détails de l'événement
FE --> User: Montre les détails de l'événement

@enduml
```

2.4 Modification et Suppression des Événements

Description

Les utilisateurs peuvent désormais **modifier** ou **supprimer** leurs événements à partir de l'interface. Cela permet une gestion complète des événements, incluant l'actualisation ou la suppression de données obsolètes.

Flux Utilisateur

1. **Modification**
   - L'utilisateur ouvre les détails de son événement.
   - Il clique sur le bouton "**Modifier**".
   - Un formulaire pré-rempli s'affiche avec les informations actuelles.
   - Après modification, il clique sur "**Enregistrer**" pour sauvegarder les modifications.
2. **Suppression**
   - L'utilisateur ouvre les détails de son événement.
   - Il clique sur le bouton "**Supprimer**".
   - Une confirmation s'affiche avant suppression définitive.

Endpoints Backend

Méthode	Endpoint	Description
PUT	`/api/events/:id`	Met à jour un événement existant.
DELETE	`/api/events/:id`	Supprime un événement spécifique.

Diagramme de Séquence : Modification et Suppression des Événements

```
@startuml
actor "Utilisateur" as User
participant "Frontend (React)" as FE
participant "Backend API" as API
participant "Base de Données" as DB

== 1. Modification ==
User -> FE: Ouvre les détails de l'événement
FE -> API: **GET** /api/events/:id
API -> DB: Récupère les données de l'événement
DB --> API: Renvoie les données de l'événement
API --> FE: Affiche les détails
User -> FE: Clique sur "Modifier" et enregistre les modifications
FE -> API: **PUT** /api/events/:id (modifications)
API -> DB: Met à jour l'événement
DB --> API: Confirme la mise à jour
API --> FE: Notifie le succès de la modification

== 2. Suppression ==
User -> FE: Clique sur "Supprimer"
FE -> API: **DELETE** /api/events/:id
API -> DB: Supprime l'événement
DB --> API: Confirme la suppression
API --> FE: Notifie le succès de la suppression
@enduml
```

🚀 Petites Fonctionnalités

📌 2.1 Page “Mes Demandes Envoyées” 📩

Description

Ajout d’une nouvelle section permettant aux utilisateurs de **suivre leurs demandes** et voir si elles sont **acceptées ou refusées**.

Flux Utilisateur

1. L’utilisateur consulte **la section “Mes demandes envoyées”**.
2. Il voit **toutes ses demandes** avec leur statut actuel.
3. **Si la demande est acceptée**, il accède aux **coordonnées du proposeur**.

Endpoints Backend

Méthode	Endpoint	Description

GET	`/interests/sent/:userId`	Retourne les demandes envoyées par l'utilisateur.
PUT	`/interests/:id`	Met à jour le statut d'une demande.

🔔 2.2 Notifications en Temps Réel 🔔

Description

Les notifications sont envoyées en temps réel à l'utilisateur lorsqu'une action importante se produit (acceptation/refus d'une demande, etc.). Cela permet une interaction fluide et réactive avec l'application.

Flux Utilisateur

1. L'utilisateur effectue une action qui génère une notification.
2. Une notification apparaît instantanément dans le panneau des notifications.
3. L'utilisateur peut la consulter et la supprimer.

Endpoints Backend

Méthode	Endpoint	Description
POST	`/notifications`	Crée une nouvelle notification.
GET	`/notifications/:userId`	Récupère toutes les notifications d'un utilisateur.
DELETE	`/notifications/:notifId`	Supprime une notification spécifique.
DELETE	`/notifications/all/:userId`	Supprime toutes les notifications d'un utilisateur.

📅 2.3 Détails d'un Événement

Description

Les utilisateurs peuvent désormais visualiser les détails d'un événement. Cette page affiche les informations complètes de l'événement sélectionné, comme son titre, sa description, sa date, son lieu, sa catégorie, et son image associée.

Flux Utilisateur

1. L'utilisateur clique sur un événement dans la liste des événements.
2. Une fenêtre modale s'affiche, contenant les détails complets de l'événement.

Endpoints Backend

Méthode	Endpoint	Description
GET	`/api/events/:id`	Récupère les détails d'un événement spécifique.

🖼️ 2.4 Gestion des Images des Événements (Frontend)

Description

La prise en charge des images d'événements a été ajoutée dans : - Le formulaire de création et de modification des événements. - La page de détails des événements.

Les utilisateurs peuvent visualiser une image par défaut (si aucune image n'est fournie) ou une image personnalisée associée à l'événement.

Flux Utilisateur

1. Lors de la création ou modification d'un événement, l'utilisateur peut spécifier l'URL d'une image.
2. Si l'utilisateur ne renseigne pas d'image, une image par défaut est utilisée.
3. La page de détails affiche l'image associée à l'événement.

Endpoints Backend

Méthode	Endpoint	Description
GET	`/api/events/:id`	Récupère les détails de l'événement, y compris l'URL de l'image.
POST	`/api/events`	Permet de créer un événement avec une image associée.
PUT	`/api/events/:id`	Permet de modifier l'image associée à un événement.
GET	`/api/validate-image`	Permet de vérifier si une URL d'image est valide.

🌍 2.5 Filtre par Villes

Description

Un filtre par villes a été ajouté pour permettre aux utilisateurs de rechercher des événements en fonction de leur localisation.

Flux Utilisateur

1. L'utilisateur sélectionne une ville dans la liste déroulante des filtres.
2. Les événements affichés sont automatiquement filtrés pour correspondre à la ville sélectionnée.

Endpoints Backend

Méthode	Endpoint	Description
GET	`/cities`	Récupère les villes disponibles pour les événements.

Note : Les filtres sont appliqués côté frontend en combinant les critères de recherche pour offrir une expérience utilisateur optimale.

🎨 2.6 Améliorations UX/UI 🎨

L'application a été **remaniée graphiquement** pour une **meilleure expérience utilisateur** :

- 🔹 **Nouvelle navbar fixe** avec **navigation fluide**.
- 🔹 **Popup de notifications stylée** avec **mise en forme propre**.
- 🔹 **Suppression du bleu flashy** et **adoption d'un design plus épuré**.
- 🔹 **Animations CSS** pour un rendu **plus dynamique**.
- 🔹 **Espacement et marges ajustés** pour **une meilleure lisibilité**.

🚀 Impact Métier & Valeur Ajoutée

Fonctionnalité	Valeur Ajoutée
🔔 Notifications en temps réel	Permet aux utilisateurs d'être informés instantanément des actions importantes.
🤝 Gestion des demandes d'intérêt	Simplifie l'interaction entre utilisateurs, rendant le processus plus intuitif.
📊 Suivi des demandes envoyées	Apporte de la transparence sur l'état des interactions.
🎨 Expérience utilisateur améliorée	Favorise l'adoption de la plateforme grâce à une interface plus intuitive et agréable.
🔍 Recherche avancée des événements	Permet une recherche rapide et précise des événements grâce à la recherche floue, même avec des erreurs typographiques.

□ Tests & Validation

- **Notifications en temps réel** : Fonctionnent sans latence.
- **Gestion des statuts (pending, accepted, rejected)** : Bien mise à jour en base.
- **UI et UX fluides** : Interface réactive et intuitive.

Documentation v1.0 – Signalement de dangers

v1.0, Février 2025 :toc: :toc-title: Sommaire

□ Introduction

La fonctionnalité de **signalement de dangers** permet aux utilisateurs de **remonter en temps réel des incidents** dans leur quartier. Cette feature repose sur un **workflow rapide** et efficace pour assurer une réactivité maximale.

□ **Objectif** : Offrir une plateforme où les résidents peuvent signaler **instantanément** des problèmes de sécurité et autres nuisances, avec **des notifications en temps réel** via WebSockets.

Pourquoi cette feature ? - □ **Faciliter la communication locale** : les utilisateurs peuvent informer leurs voisins d'un danger potentiel. - □ **Réactivité immédiate** : les signalements sont visibles immédiatement et les dangers critiques envoient une notification. - □ **Amélioration de la sécurité** : plus de transparence et de réactivité sur les incidents urbains.

□□ Fonctionnalités

□ **1. Section Signalement rapide** -Via un formulaire dédié, les utilisateurs peuvent signaler un problème en quelques clics : - Sélection d'une **catégorie** parmi : * □ **Dangers & Sécurité** (vol, bagarre, accident…) * □ **Problèmes Urbains** (routes endommagées, lampadaires HS…) * □ **Nuisances Sonores** (fête bruyante, klaxons…) * □ **Problèmes de stationnement** (véhicule gênant, parking saturé…) - Description courte et **zone du quartier** concernée. - Option □ **Critique** : Si activé par l'utilisateur lors de la saisie du formulaire, on envoie une notification immédiate aux résidents.

□ **2. Section pour l'affichage des signalements** - □ Les **5 derniers signalements** sont visibles sur le **Dashboard**, mis à jour en temps réel. - □ A l'aide d'un bouton "voir plus", l'utilisateur peut voir en détail tous les signalements qui ont été faits, sur la **page dédiée aux signalements**.

□ **3. Ajout de notifications WebSockets pour signaler le danger** - **Si le signalement est critique**, une notification en **temps réel** est envoyée à **tous les utilisateurs**. - **Mise à jour automatique** du compteur de notifications. - **Pas besoin de recharger la page** : le signalement et les notifs sont **instantanément visibles**, ce qui permet à l'utilisateur de recevoir l'information sans faire d'effort particulier.

 4. Section Mes signalements - Les utilisateurs peuvent consulter **tous leurs signalements** passés. - **Marquer un signalement comme résolu** pour indiquer que le problème a été traité. - Synchronisation avec la liste globale : Si l'alerte est résolue, elle apparaît aussi comme "résolue" pour tous.

Workflow du signalement de danger

```
@startuml
participant "Utilisateur" as UI
participant "Frontend (React)" as FE
participant "Backend API" as API
participant "Base de Données" as DB
participant "WebSockets" as WS
participant "Autres utilisateurs" as USERS

==  1. Signalement d'un danger ==
UI -> FE: Remplit le formulaire et valide
FE -> API: **POST** /signalements (catégorie, description, critique…)
API -> DB:  Enregistre le signalement
DB --> API:  OK

==  2. Notification en temps réel si critique ==
API -> WS:  **Émettre une notification à tous les utilisateurs**
WS --> USERS:  **Notification "Problème signalé"**
USERS -> FE: **Mise à jour immédiate du compteur de notifications**

==  3. Mise à jour du tableau de bord ==
API -> WS: **Mise à jour "Derniers signalements"**
WS --> FE:  Mettre à jour **sans recharger**** 

==  4. Gestion des signalements ==
UI -> FE: Accède à " Mes signalements"
FE -> API: **GET** /signalements/utilisateur/{user_id}
API -> DB:  Récupère les signalements de l'utilisateur
DB --> API:  Renvoie la liste
API --> FE: Affichage des signalements

==  5. Marquer un signalement comme résolu ==
UI -> FE: Clique sur "✔ Marquer comme résolu"
FE -> API: **PUT** /signalements/:id/resoudre
API -> DB:  Met à jour le statut "Résolu"
DB --> API: **OK**

@enduml
```

🔧 Détails Techniques

🗄 **Base de données** - **Table** `signalements` : * `id` (INT, PRIMARY KEY) * `user_id` (INT, FOREIGN KEY vers `users`) * `categorie` (ENUM) * `description` (TEXT) * `critique` (BOOLEAN) * `quartier` (TEXT) * `resolu` (BOOLEAN, DEFAULT FALSE) * `date_creation` (DATETIME, DEFAULT CURRENT_TIMESTAMP)

- **Table** `notifications` (ajout du type `danger_alert`)
  - `id`
  - `user_id`
  - `type` (ENUM)
  - `message`
  - `related_entity_id`
  - `created_at`

🖥 **Backend API (Node.js, Express, MySQL)** - **POST** `/signalements` → Crée un nouveau signalement - **GET** `/signalements` → Récupère tous les signalements - **PUT** `/signalements/:id/resoudre` → Marque un signalement comme résolu - **WebSockets** : Notification temps réel via `io.emit("notification-global", {⋯})`

🎨 **Frontend (React)** - **Composants** * `SignalementForm.jsx` → Formulaire de signalement * `SignalementsList.jsx` → Affichage des signalements * `Dashboard.jsx` → Intégration des signalements récents * `Notifications.jsx` → Gestion des alertes en temps réel

---

📸 Illustrations

🖼 **Wireframe** image::images/wireframe_signalement.png[]

🖼 **Capture d'écran du site** image::images/signalements_dashboard.png[]

🔍 Comparaison avec les plateformes existantes

Notre solution se distingue par son approche **temps réel** et son **interface ultra-réactive**. Voici comment elle se positionne face aux alternatives existantes :

Plateforme	Type de signalement	Instantanéité des mises à jour	Notifications aux résidents	Suivi des signalements
AlloVoisins / Nextdoor	Discussions entre voisins, annonces de services	Non (les publications sont statiques)	Non (les notifications concernent uniquement des interactions sociales)	Non (pas de suivi des incidents)
DansMaRue (Paris)	Signalements urbains (voirie, éclairage public, etc.)	Non (validation requise par la mairie)	Non (aucune notification directe aux citoyens)	Oui (suivi possible après traitement)
FixMyStreet	Problèmes d'infrastructure (routes, mobilier urbain)	Non (mises à jour manuelles)	Non (seules les autorités locales reçoivent les alertes)	Oui (gestion par les services municipaux)
Notre application	Dangers, nuisances et incidents du quotidien	**Oui** (mise à jour automatique en temps réel)	**Oui** (alerte immédiate aux résidents en cas de danger critique)	**Oui** (gestion et résolution directe par les utilisateurs)

Pourquoi nous sommes innovants ?

 Rapidité & Instantanéité Notre solution utilise **les WebSockets** pour une mise à jour immédiate des signalements et une **notification instantanée** aux résidents.

 Autonomie des utilisateurs L'utilisateur **peut signaler, suivre et clôturer un incident** sans intervention administrative.

 Notifications intelligentes Seuls les signalements **critiques** déclenchent une alerte pour éviter le spam tout en maintenant un haut niveau de réactivité.

 Expérience utilisateur optimisée Interface fluide, ergonomique et conçue pour une utilisation rapide **depuis un mobile ou un desktop**.

Notre application comble un **manque majeur** dans la gestion des signalements en quartiers : **l'instantanéité et l'autonomie des citoyens**.

 Conclusion : Contrairement à d'autres plateformes, notre application offre **une communication rapide, directe et communautaire**.

Tests réalisés

- **Tests unitaires** : Vérification du bon enregistrement d'un signalement en base.

- **Tests d’intégration** : Simulation d’une notification critique et validation de son affichage en WebSockets.
- **Tests REST API** (Postman) :
  - Envoi d’un signalement → **200 OK**
  - Marquer un signalement comme résolu → **200 OK**
  - Récupération des notifications en temps réel → **Fonctionnel**

Conclusion

 Bilan de la feature : - **Instantanéité & efficacité** avec **WebSockets**. - **Expérience utilisateur fluide** (mise à jour automatique des signalements et notifications). - **Modularité & évolutivité** (possibilité d’ajouter des filtres par quartier, historique des signalements…).

 Prochaines améliorations possibles : - Ajouter une **cartographie** interactive des signalements. - Permettre aux utilisateurs de **commenter et réagir** aux signalements. - **Statistiques** sur les types de signalements les plus fréquents. - Ajouter le temps réel pour dire à tous les utilisateurs qu’un signalement est désormais terminé.

 Feature livrée avec succès !

Documentation v1.0 – Gestion des Projets

Description

Cette fonctionnalité introduit la gestion complète des projets au sein de l’application. Les utilisateurs peuvent **créer, modifier et supprimer** des projets communautaires, voter pour un projet et suivre leur évolution. Les projets sont **rattachés aux quartiers** pour favoriser des initiatives locales et renforcer l’engagement des résidents.

Flux Utilisateur

1. **Création d’un Projet**
   - L’utilisateur clique sur **“+ Créer un projet”**.
   - Il remplit un formulaire comprenant : **titre, description, catégorie, date limite**.
   - Le projet est automatiquement **associé au quartier** de l’utilisateur.
   - Une fois validé, le projet apparaît dans la liste des projets de son quartier.
2. **Affichage des Projets**
   - Par défaut, seuls les **projets du quartier** de l’utilisateur sont affichés.

- Une case à cocher **"Afficher tous les projets"** permet de voir **l'ensemble des projets** disponibles.

3. **Détails d'un Projet**
   - Un utilisateur peut **cliquer sur un projet** pour voir ses détails complets (créateur, description, votes, date limite).
   - Si l'utilisateur est le créateur du projet, il peut **le modifier ou le supprimer**.
4. **Modification d'un Projet (seulement pour le créateur)**
   - L'utilisateur accède aux détails de son projet et clique sur **"Modifier"**.
   - Un **formulaire pré-rempli** lui permet de mettre à jour les informations.
   - Après validation, les modifications sont **enregistrées en base** et **affichées en temps réel**.
5. **Suppression d'un Projet (seulement pour le créateur)**
   - L'utilisateur clique sur **"Supprimer"**.
   - Une **confirmation** s'affiche pour éviter toute suppression accidentelle.
   - Le projet est définitivement supprimé.
6. **Votes sur un Projet**
   - Les utilisateurs peuvent **voter pour ou contre** un projet (**Upvote** ou **Downvote**).
   - Un utilisateur **ne peut pas voter pour son propre projet**.
   - Les votes sont **mis à jour en temps réel** sans rechargement de la page.
   - Une fois la période de votes terminée, un projet est **accepté ou rejeté** en fonction du nombre de votes positifs/négatifs.

Endpoints Backend

Méthode	Endpoint	Description
POST	`/api/projects`	Créer un projet
GET	`/api/projects`	Récupérer tous les projets (avec option quartier/tous les projets)
GET	`/api/projects/:id`	Récupérer les détails d'un projet
PUT	`/api/projects/:id`	Modifier un projet **(seulement si l'utilisateur est le créateur)**
DELETE	`/api/projects/:id`	Supprimer un projet **(seulement si l'utilisateur est le créateur)**
POST	`/api/projects/:id/vote`	Voter pour un projet (/)

Diagramme de Séquence : Gestion des Projets

```
@startuml
actor "Utilisateur" as User
participant "Frontend (React)" as FE
participant "Backend API" as API
participant "Base de Données" as DB

== 🏗 1. Création d'un Projet ==
User -> FE: Clique sur "Créer un projet"
FE -> API: **POST** /api/projects (titre, description, catégorie, deadline,
quartier_id)
API -> DB: 🔍 Vérifie les données et insère le projet
DB --> API: ✅ OK (id_projet)
API --> FE: Confirme la création et met à jour la liste des projets

== 📋 2. Affichage des Projets ==
User -> FE: Accède à la page "Projets"
FE -> API: **GET** /api/projects?quartier_id=X
API -> DB: 🔍 Récupère les projets du quartier
DB --> API: 📋 Liste des projets filtrés
API --> FE: Affichage des projets

== ✏ 3. Modification d'un Projet ==
User -> FE: Ouvre son projet et clique sur "Modifier"
FE -> API: **PUT** /api/projects/:id (nouvelles valeurs)
API -> DB: 🔄 Met à jour le projet
DB --> API: ✅ Confirme la mise à jour
API --> FE: Affichage des nouvelles valeurs

== 🗑 4. Suppression d'un Projet ==
User -> FE: Clique sur "Supprimer"
FE -> API: **DELETE** /api/projects/:id
API -> DB: 🗑 Supprime le projet
DB --> API: ✅ Suppression confirmée
API --> FE: Met à jour la liste des projets

== 👍 5. Vote sur un Projet ==
User -> FE: Clique sur "👍" ou "👎"
FE -> API: **POST** /api/projects/:id/vote (vote=up/down, user_id)
API -> DB: 🔍 Vérifie si l'utilisateur a déjà voté
DB --> API: ✅ OK
API -> DB: 🔄 Met à jour le vote
DB --> API: 📊 Retourne le nouveau compteur de votes
API --> FE: Affichage des votes mis à jour

@enduml
```

□ Impact Métier & Valeur Ajoutée

Fonctionnalité	Valeur Ajoutée
□ **Projets rattachés aux quartiers**	Favorise les initiatives locales et renforce le lien social.
□ **Gestion complète (CRUD)**	Permet aux utilisateurs de créer, modifier et supprimer leurs projets en toute autonomie.
□□ **Votes en temps réel**	Donne un retour direct sur l'intérêt du projet auprès de la communauté.
□ **Visibilité optimisée**	Les projets sont mis en avant selon leur popularité et leur pertinence.

□ Tests & Validation

- **Tests unitaires** :
  - Création, modification et suppression d'un projet → □ OK
  - Votes sur un projet → □ OK
- **Tests d'intégration** :
  - Validation de l'affichage des projets filtrés par quartier → □ Fonctionnel
  - Test de l'option "Afficher tous les projets" → □ Fonctionnel
- **Tests REST API (Postman)** :
  - `POST /api/projects` → **201 Created**
  - `GET /api/projects` (avec quartier_id) → **200 OK**
  - `PUT /api/projects/:id` (modification) → **200 OK**
  - `DELETE /api/projects/:id` → **200 OK**
  - `POST /api/projects/:id/vote` → **200 OK**

□ Conclusion

□ **Bilan de la feature :** - □ Création et gestion des projets **simple et fluide** - □ **Filtrage intelligent** des projets selon le quartier - □ **Système de votes participatif** pour la validation des projets - □ **Interface optimisée et ergonomique**

□ **Prochaines améliorations possibles :** - Ajout d'une **gestion des tâches** par projet (Kanban). - Système de **commentaires** sur les projets. - Ajout d'un **statut de projet** (`En cours`, `Terminé`, etc.).

□ **Feature livrée avec succès !** □

Documentation Technique – Release v1.1 - DevOps2

Objectif de la fonctionnalité

Permettre aux utilisateurs de :

1. Se connecter à leur compte Google via un bouton dédié.
2. Visualiser leurs événements Google Calendar dans le dashboard de la plateforme.
3. Ajouter automatiquement un événement à leur Google Calendar lorsqu'ils cliquent sur le bouton "Participer" à un événement.

Cette intégration offre une expérience fluide et connectée, évitant aux utilisateurs d'avoir à gérer manuellement leur emploi du temps après s'être inscrits à un événement.

Nouveauté côté utilisateur

Avant cette release, l'utilisateur ne pouvait ni s'inscrire à un événement, ni le quitter, et aucune synchronisation n'existait avec son agenda personnel.

Avec cette intégration, il peut désormais :

- Participer ou quitter un événement local directement depuis l'interface KnockNShare ;
- Ajouter automatiquement cet événement à son propre Google Calendar (avec lieu, date, heure, description) ;
- Et surtout, visualiser en temps réel ses événements Google, y compris ceux ajoutés via KnockNShare, depuis le dashboard de l'application.

Cette avancée rapproche la plateforme d'un véritable assistant de vie communautaire connecté, conforme à notre vision de simplification des interactions sociales au sein des quartiers.

Fonctionnalités livrées

- Connexion OAuth2.0 à Google (frontend) avec affichage des événements à venir.
- Ajout automatique d'un événement Google Calendar lors du clic sur "Participer".
- Conservation du token d'accès dans un contexte React (`GoogleAuthContext`) avec mise à jour automatique.
- Bouton "Participer" fonctionnel : interaction avec la base de données + appel API Google Calendar.

Authentification Google (OAuth2) – Intégration initiale

Objectif

Permettre aux utilisateurs de se connecter à KnockNShare via leur compte Google, sans avoir à créer un compte ou à renseigner un mot de passe. Cette étape est également un prérequis technique à l'intégration du calendrier Google.

Fonctionnalités livrées

- Redirection de l'utilisateur vers la page d'authentification Google.
- Décodage du `id_token` pour obtenir les données de base (`email`, `name`, `google_id`).
- Vérification de l'existence de l'utilisateur en base, création automatique si inexistant.
- Génération d'un JWT signé, transmis au frontend via redirection.
- Stockage du `userId` et de l'`access_token` dans le `localStorage` (clé `googleAccessToken`) pour les requêtes vers l'API Calendar.

Implémentation technique

Backend (Node.js/Express)

- Ajout des routes suivantes :

```
GET  /api/auth/google           // redirection vers Google
GET  /api/auth/google/callback  // traitement du code + création/utilisateur
```

- Ajout du scope :

```
scope: "openid profile email https://www.googleapis.com/auth/calendar.readonly"
```

- Décodage du `id_token` avec `jsonwebtoken`, génération d'un JWT interne avec `userId`, `email`, etc.
- Enregistrement des utilisateurs Google en base (table `users`) avec `google_id`, sans mot de passe.

Frontend (React)

- Ajout d'un bouton "Se connecter avec Google" sur la page de connexion (`LoginPage.jsx`).
- Ajout d'une page `OAuthSuccess.jsx` qui :
- lit le `token` et l'`access_token` dans l'URL,
- les stocke dans `localStorage`,

- met à jour l'`AuthContext`,
- redirige l'utilisateur vers le dashboard.

Migration SQL associée

Ajout de la colonne `google_id` dans la table `users` et passage de `password` en nullable :

```
ALTER TABLE users ADD COLUMN google\_id VARCHAR(255);
ALTER TABLE users MODIFY COLUMN password VARCHAR(255) NULL;
```

Fichier de migration : `update_users_schema.sql`

Configuration

Ajout des variables suivantes dans le fichier `.env` :

```
GOOGLE\_CLIENT\_ID=...
GOOGLE\_CLIENT\_SECRET=...
GOOGLE\_CALLBACK\_URL=http://localhost:3000/api/auth/google/callback
```

Le `access_token` Google est désormais stocké dans le navigateur et peut être utilisé par les autres fonctionnalités (ex : synchronisation Calendar).

Intégration Google Calendar – Détails techniques

1. Authentification OAuth

Le composant `GoogleAuthProvider.jsx` initialise et configure `gapi.auth2` :

- Récupère et stocke le token.
- Écoute les connexions/déconnexions avec `auth.isSignedIn.listen`.
- Expose `signIn()` et `token` via `GoogleAuthContext`.

Scope utilisé :

```
const SCOPES = "https://www.googleapis.com/auth/calendar.events";
```

2. Affichage des événements Google (dashboard)

Dans `Dashboard.jsx` :

- Le bouton **Connecter Google Calendar** lance `signIn()`.
- Si un `token` est présent, les événements sont récupérés via :

```
gapi.client.calendar.events.list({...})
```

- Le composant `DashboardCalendar.jsx` affiche ces événements.

3. Création d'événement à la participation

Dans `EventPage.jsx`, lors du clic sur **Participer** :

- Ajout du participant via :

```
POST /api/events/participate
```

- Création d'un événement Google Calendar :

```
POST https://www.googleapis.com/calendar/v3/calendars/primary/events
Headers: Authorization: Bearer access_token
```

Payload envoyé :

```
{
  "summary": "Titre",
  "description": "Description",
  "location": "Adresse",
  "start": { "dateTime": "...", "timeZone": "Europe/Paris" },
  "end": { "dateTime": "...", "timeZone": "Europe/Paris" }
}
```

4. Backend (routes Node/Express)

```
POST   /api/events/participate   // ajoute à participants
DELETE /api/events/leave         // supprime de participants
```

Gestion de la base MySQL avec vérification des doublons.

Architecture technique

Composant/Fichier	Rôle
`GoogleAuthProvider.jsx`	Gère l'authentification OAuth et expose le token

Composant/Fichier	Rôle
`Dashboard.jsx`	Affiche les événements et déclenche la connexion
`DashboardCalendar.jsx`	Affiche les événements Google Calendar
`EventPage.jsx`	Gère la logique de participation + ajout Google Calendar
`/api/events/participate`	Backend : enregistre la participation
`/api/events/leave`	Backend : supprime la participation

Problèmes rencontrés

- **Expiration du token** : contournée avec écoute `isSignedIn` dans le contexte
- **Client ID multiple** : risque de conflit, à sécuriser dans un `.env`.
- **Perte de session inter-composant** : résolue avec `GoogleAuthContext`.

Bénéfices de cette feature dans la stratégie globale de notre application

Cette fonctionnalité améliore considérablement l'expérience utilisateur :

- Meilleure expérience pour l'utilisateur : automatisation de l'ajout d'événements, la participation est plus simple et rapide, et apparaît dans le calendrier personnel de l'utilisateur -Gain de temps → plus besoin de noter l'événement dans un agenda externe. -Moins d'oubli → les rappels automatiques Google s'appliquent. -Notre plateforme KnockNshare devient un vrai outil d'organisation de vie communautaire.

Tests réalisés

- Connexion OAuth fonctionnelle.
- Visualisation des événements dans le calendrier.
- Ajout d'un événement fonctionnel avec token valide.
- Déconnexion/reconnexion gérée automatiquement.

Release v1.1 – Récapitulatif

- Intégration complète OAuth (Google Calendar)
- Ajout automatique d'événements
- Visualisation des événements Google
- Reconnexion et gestion de session

- Base backend synchronisée