

Support temps réel

Vue d'ensemble

Le support utilise HTTP pour valider et persister les conversations, messages et tickets; Socket.IO diffuse les changements après leur sauvegarde. Le refactoring du 30 septembre 2026 sépare le temps réel du compte de celui du support.

Deux namespaces et un transport partagé

Responsabilité	Compte : /account	Support : /support
Backend	AccountGateway + PresenceService	SupportGateway + SupportService
Frontend	AccountSocketService	SupportSocketService
Événements reçus	user.status, user.presence, admin.status, notification.created	message.created, ticket.updated, typing
Actions émises	updateStatus, updateTelemetry, updateGamePresence	joinConversation, leaveConversation, typing, checkOnline

Les services frontend utilisent `RealtimeSocket`, qui centralise l'authentification, les observables, la reconnexion et les acknowledgements. Les namespaces d'un même serveur partagent la connexion sous-jacente, mais leurs rooms sont indépendantes.

```
flowchart LR
  UI[Interface] --> HTTP[API HTTP NestJS]
  HTTP --> S[SupportService]
  S --> DB[(MySQL)]
  S --> E[EventEmitter]
  E --> SG[SupportGateway /support]
  SG --> SS[SupportSocketService]
  SS --> UI
  AS[AccountSocketService] <--> AG[AccountGateway /account]
```

```
AG --> P[PresenceService]
N[NotificationService] --> AG
AS --> UI
```

Authentication, rooms et présence

Chaque gateway installe `socketAuthMiddleware(jwtService)` à l'initialisation du namespace. Le JWT est vérifié avant les handlers; le payload est placé dans `client.data.user`. Le token est accepté depuis `handshake.auth.token`, la query ou l'en-tête Authorization.

`AccountGateway` enregistre et retire les connexions dans `PresenceService`. Seuls les sockets du namespace compte comptent pour la présence. `SupportGateway` rejoint `user_<id>` et, pour les administrateurs, `admins`; il ne crée pas une seconde connexion de présence.

Pour `joinConversation`, un utilisateur doit posséder la conversation; les admins peuvent joindre les conversations de support. `typing` exige l'appartenance à `conversation_<id>` et ne vise que les autres sockets de cette room. `checkOnline` interroge `PresenceService`; un non-admin est limité aux admins assignés à ses tickets.

Message et réponse IA

1. L'interface envoie le message et la pièce jointe par HTTP.
2. `SupportService` valide la conversation et la pièce jointe puis sauvegarde le message USER.
3. `MESSAGE_CREATED` est émis; le gateway mappe le message avec `SupportMapper.toMessageDto` et diffuse `message.created` vers sa conversation.
4. Dans une conversation non escaladée, le service construit l'historique, la langue et les droits documentaires puis appelle `AIService.queryRAG`. La réponse ou le message système est persisté et diffusé par le même chemin.

Voir [Pipeline RAG et système IA](#) pour les clarifications et l'escalade lorsque le contexte est insuffisant.

Escalade et cycle de vie des tickets

L'escalade protège la transition `conversation.escalated: false → true` par une transaction atomique. Elle crée le ticket OPEN, sauvegarde un message SYSTEM, émet `MESSAGE_CREATED` et `TICKET_UPDATED`, puis demande un résumé asynchrone pour l'administration. Le premier admin qui répond peut devenir l'assigné.

```
stateDiagram-v2
[*] --> OPEN
OPEN --> IN_PROGRESS: assignTicket ou première réponse admin
OPEN --> RESOLVED: resolveTicket
IN_PROGRESS --> RESOLVED: resolveTicket
RESOLVED --> IN_PROGRESS: reopenTicket avec assigné
RESOLVED --> OPEN: reopenTicket sans assigné
```

Lecture et notifications

Les marqueurs `lastUserReadAt / lastUserReadMessageId` et `lastAdminReadAt / lastAdminReadMessageId` sont indépendants. L'utilisateur compte les messages ADMIN non lus; l'admin compte les messages USER non lus. Pour `ticket.updated`, le gateway obtient deux vues via `getTicketUpdates` et envoie la vue utilisateur à sa room, la vue admin à `admins`.

Les notifications de compte suivent `NotificationService → AccountGateway → AccountSocketService`. Elles sont distinctes des mises à jour de ticket. Voir [Notifications et Push](#).

Reconnexion et état frontend

`SupportSocketService` conserve l'ID de conversation active et rejoint sa room à la reconnexion. `AccountSocketService` remet l'utilisateur en ligne, réémet la télémétrie si le consentement est accordé et restaure le jeu actif. Le retrait du consentement envoie une télémétrie nulle; quitter le jeu efface sa présence. Les composants support restent centrés sur les messages, tickets et leur état local.

`RealtimeSocket` encapsule `connect/disconnect`, `on`, `emit`, `emitWithAck` et `close`. L'HTTP charge l'état initial et persiste les actions; les événements synchronisent l'interface.

Fichiers principaux

- Backend : `src/gateway/account.gateway.ts`, `support.gateway.ts`,
`src/guards/socketAuth.middleware.ts`, `src/service/presence.service.ts`, `support.service.ts`

- Angular : `service/realtime/realtime-socket.ts`, `service/account/account-socket.service.ts`, `service/support/support-socket.service.ts` et `support-state.service.ts`.
 - Contrats : `Models/Presence.ts`, `Models/Notification.ts`, `Models/Support.ts`; constantes : `account.constants.ts`, `realtime.constants.ts`, `support.constants.ts`.
-

Revision #5

Created 2026-09-17 20:58:48 UTC by Mathis

Updated 2026-09-30 14:41:49 UTC by Mathis