

Vue d'ensemble du système Web

But de cette page

Cette page est la **carte d'entrée du système Web Uniksims**. Elle permet de comprendre rapidement quelles applications existent, comment elles communiquent, où sont stockées les données et dans quel chapitre chercher lorsqu'un problème survient.

Carte globale du système

flowchart LR

USER[Utilisateur] --> ANG[Frontend Angular]

ADMIN[Administrateur] --> ANG

ANG -->|HTTP| NEST[NestJS principal :3000]

ANG <-->|Socket.IO| WS[SupportGateway /support]

ANG <-->|Socket.IO| ACC[AccountGateway /account]

ANG --> UNITY[Unity WebGL]

UNITY -->|HTTP / heartbeat / données jeu| NEST

NEST --> CTRL[Controllers]

CTRL --> SVC[Services métier]

SVC --> ORM[TypeORM]

ORM --> MYSQL[(MySQL)]

SVC --> REDIS[(Redis)]

SVC --> BC[BrainCloud]

SVC --> FB[Firebase / FCM]

```
SVC --> MAIL[Mail]
SVC --> AI[Serveur IA :8106]

NEST --> ADMINEXP[Admin Server Express :8105]
ADMINEXP --> BC
ADMINEXP --> REDIS

AI --> CHROMA[(Chroma / embeddings)]
AI --> PROVIDER[Provider IA]

NEST --> STATIC[Frontend Angular compilé]
STATIC --> USER
```

Les grandes briques

Frontend Angular

Application principale côté navigateur dans `uniksimFrontEnd/`. Elle contient les pages utilisateur et admin, les composants UI, les services HTTP, l'authentification frontend, le support temps réel et l'intégration Unity.

Backend NestJS

Serveur HTTP principal dans `uniksim-back-end/`. Le point d'entrée est `src/main.ts` et l'assemblage global se fait dans `src/app.module.ts`. Il expose les APIs utilisateur, jeu, administration, support, notifications, push, mail et IA.

Admin Server Express

Serveur HTTP auxiliaire séparé dans `src/node.js/server/admin_server.js`, sur le port `8105`. Il sert principalement certaines opérations historiques/BrainCloud, Redis, copie de simulations et SSE.

Serveur IA

NestJS attend l'ouverture du listener IA sur `127.0.0.1:8106`, puis démarre. L'initialisation des providers et de Chroma se poursuit en arrière-plan; leur indisponibilité n'empêche pas NestJS ni

l'endpoint de santé de démarrer. Les requêtes retiennent les dépendances manquantes. Le serveur IA contient des contrats, factories, providers, tokenizer, pricing, `UsageMeter` et `CostGuard` sous `src/node.js/server/ai/`. Gemini utilise maintenant Vertex AI, avec génération et embeddings sélectionnés indépendamment.

Unity WebGL

Le jeu Unity est intégré au Web. Angular affiche le build WebGL et Unity communique avec le backend pour les versions, textes, données utilisateur, heartbeat, classements, images et crash logs.

Persistence

- **MySQL + TypeORM** : utilisateurs, rôles, produits, support, installations FCM, etc.
- **Redis** : présence, sessions, notifications, rankings et autres données runtime.
- **BrainCloud** : plusieurs données historiques et fonctions liées aux simulations.
- **Chroma** : recherche vectorielle du système IA/RAG.

Flux typique d'une requête Web

```
sequenceDiagram
    participant U as Utilisateur
    participant A as Angular
    participant C as Controller NestJS
    participant S as Service
    participant D as MySQL / Redis / BrainCloud

    U->>A: Action dans l'interface
    A->>C: Requête HTTP
    C->>C: Guard / validation
    C->>S: Appel métier
    S->>D: Lecture / écriture
    D-->>S: Résultat
    S-->>C: Données métier
    C-->>A: Réponse API
    A-->>U: Mise à jour de l'interface
```

Flux temps réel

Deux namespaces Socket.IO séparent les responsabilités : `/account` porte présence, télémétrie consentie, jeu actif et notifications; `/support` porte messages, tickets et typing. Le middleware JWT est commun, les rooms sont propres à chaque namespace et le frontend partage `RealtimeSocket`.

Seuls les sockets du compte sont enregistrés dans `PresenceService`. Voir [Support temps réel](#) et [Notifications et Push](#).

Toutes les notifications sont stockées dans une inbox Redis du compte, diffusées par `AccountGateway` puis poussées vers toutes ses installations FCM. Les notifications envoyées par le jeu passent par `GameNotificationService`, qui vérifie les membres de la simulation BrainCloud. Les subscriptions FCM par `productCode` ont été retirées.

Build et hébergement

Le build Angular de production est généré dans `uniksim-back-end/src/front`, puis NestJS est buildé à son tour. Le backend sert ensuite le frontend compilé avec `ServeStaticModule`. L'instance de développement hébergée est `sim.dev.uniksim.com`. Voir **Build et déploiement Web**.

Où chercher selon le problème

- **Écran, navigation ou composant** → chapitre Frontend Angular.
- **Endpoint, logique backend, auth** → Serveurs et API.
- **Donnée SQL ou Redis** → Base de données et persistance.
- **Constante, port, environnement** → Configuration, environnements et constantes.
- **Chat, WebSocket, RAG, provider IA** → Support et IA.
- **Unity, notifications, Firebase, services externes** → Intégrations et systèmes transversaux.
- **Déploiement / instance demo** → Build et déploiement Web.

Points d'entrée essentiels

- `uniksimFrontEnd/src/main.ts`
- `uniksimFrontEnd/src/app/app.module.ts`
- `uniksim-back-end/src/main.ts`
- `uniksim-back-end/src/app.module.ts`
- `uniksim-back-end/src/app.controller.ts`
- `uniksim-back-end/src/database.providers.ts`

- `uniksim-back-end/src/node.js/server/constantsVariable.js`
 - `uniksim-back-end/src/node.js/server/admin_server.js`
 - `uniksim-back-end/src/node.js/server/ai/`
-

Revision #4

Created 2026-09-18 17:29:47 UTC by Mathis

Updated 2026-09-30 14:41:44 UTC by Mathis