

Ajouter une fonctionnalité Web de bout en bout

Objectif

Cette page donne un chemin pratique pour ajouter une fonctionnalité Web sans oublier une couche. Le principe est de suivre le flux Angular → API → service → données → retour UI.

Flux type

flowchart LR

```
UI[Page / Component Angular] --> FES[Service Angular]
FES --> AUTH[AuthInterceptor]
AUTH --> CTRL[Controller NestJS]
CTRL --> GUARD[Guard]
GUARD --> BES[Service backend]
BES --> DATA{Données}
DATA --> ORM[TypeORM / MySQL]
DATA --> REDIS[Redis]
DATA --> BC[BrainCloud]
BES --> CTRL
CTRL --> FES
FES --> UI
```

Étapes

1. Définir l'action utilisateur, les données entrantes/sortantes, le niveau d'accès et le type de persistance.

2. Côté Angular, placer l'écran dans `Pages/` ou le composant dans `Components/`, puis passer par un service Angular pour les appels backend.
3. Ajouter ou réutiliser la route dans `app.module.ts` et appliquer le guard frontend adapté.
4. Côté backend, ajouter l'endpoint au controller logique existant. Garder le controller mince.
5. Placer la logique métier dans `src/service/`.
6. Choisir le stockage : TypeORM/MySQL pour le relationnel persistant, Redis pour le runtime/temporaire, BrainCloud pour les données historiques de simulation déjà gérées là.
7. Ajouter Socket.IO/EventEmitter seulement si la mise à jour doit être temps réel.
8. Appliquer le guard backend approprié et valider les IDs et permissions côté serveur.
9. Centraliser les paramètres dans `constantsVariable.js` ou les fichiers `.env`.
10. Tester le cas nominal, les entrées invalides, les accès interdits et les erreurs de dépendances.

Choisir le stockage

flowchart TD

```
D[Besoin de stockage] --> P{Persistant relationnel ?}
P -- Oui --> SQL[Entity TypeORM + MySQL]
P -- Non --> RT{Temporaire / temps réel ?}
RT -- Oui --> REDIS[RedisService]
RT -- Non --> OLD{Déjà géré dans BrainCloud ?}
OLD -- Oui --> BC[BrainCloud]
OLD -- Non --> REVIEW[Revoir le modèle avant d'ajouter un stockage]
```

Checklist rapide

- UI dans le bon dossier.
- Service Angular réutilisé ou créé au bon endroit.
- Route et guards corrects.
- Controller mince.
- Logique dans un service backend.
- Stockage choisi selon le type de donnée.
- Temps réel ajouté seulement si nécessaire.
- Cas d'erreur et permissions testés.
- Configuration centralisée.
- Build/deploy vérifié.
- Documentation mise à jour si l'architecture change.

Revision #1

Created 2026-09-18 18:05:34 UTC by Mathis

Updated 2026-09-18 18:05:48 UTC by Mathis