

Dev Guide

bjschool.tn — Architecture, stack technique, multi-tenant, sécurité et conventions de code de bjschool, pour les développeurs.

À propos de ce guide

Ce guide s'adresse aux développeurs travaillant sur le code de bjschool (nom technique interne : **SchoolBoost**). Il résume l'architecture, les conventions et les règles de sécurité définies dans les documents internes AMBJ Consulting (`CLAUDE.md` , `ARCHITECTURE.md` , `SECURITY.md` , `CODING.md` , `TESTING.md` , `GIT.md`). En cas de divergence, ces documents sources font foi.

Topologie des dépôts

bjschool est un **polyrepo** de trois dépôts frères, plus le site vitrine que vous lisez en ce moment :

Dépôt	Rôle
<code>SchoolBoost-infra</code>	Spécifications, plan de livraison, docker-compose, QA
<code>SchoolBoost-api</code>	Backend ASP.NET Core
<code>SchoolBoost-front</code>	SPA Vue 3 + Vite
<code>bjschool-site</code>	Site vitrine + manuels (ce dépôt)

Stack technique

Couche	Technologie
Framework backend	ASP.NET Core 10 — REST API, Controllers, Middleware, DI
ORM	Entity Framework Core 10 (migrations, filtre global multi-tenant)

Couche	Technologie
Base de données	PostgreSQL 16 — une base par tenant + <code>schoolboost_admin</code>
CQRS	MediatR — tous les cas d'usage passent par Command/Query
Validation	FluentValidation
Temps réel	SignalR (portail public de disponibilité)
Cache	Redis (sessions + backplane SignalR)
Sécurité	BCrypt cost ≥ 12 , JWT en cookies HttpOnly
Tests	xUnit + Moq (<code>MockBehavior.Strict</code>) + FluentAssertions + Testcontainers
Frontend	Vue 3 + Vite + Pinia + shadcn-vue/Tailwind v4/Reka UI, <code>@ambj-consulting/vue-core</code>

La langue du code est l'anglais partout (classes, méthodes, routes, logs) ; les documents de spécification et de planification restent en français, langue métier du produit tunisien.

Architecture Clean / CQRS

```
Domain ← Application ← Infrastructure ← API
      ↑
    (interfaces)
```

- **Domain** ne dépend d'aucun package externe (0 NuGet). Entités avec constructeur privé, factory method statique, mutations via méthodes-verbos métier, jamais de setters publics.
- **Application** contient les Commands/Queries MediatR (`VerbNomCommand` , `GetNomQuery` , `ListNomQuery`) et leurs Validators FluentValidation — **aucune Command sans validator**.
- **Infrastructure** implémente les interfaces définies dans Application (EF Core, BCrypt, Npgsql).

- **API** expose les Controllers, qui héritent de `BaseController` et n'accèdent jamais directement aux services.

Routes : `/api/v1/{resource}` (métier, filtré par tenant), `/api/v1/admin/{resource}` (Super-Admin uniquement), `/hubs/{nom}` (SignalR).

Multi-tenant — le cœur du système

Base de données par tenant

```
schoolboost_admin      → Tenants, Users, Licences (AdminDb)
schoolboost_tenant_{slug} → Données métier (TenantDb, une base par tenant)
```

Résolution du tenant (`app.bjschool.tn` / `slug.bjschool.tn`)

Le tenant est résolu par le `TenantResolutionMiddleware` selon une priorité stricte — **jamais** depuis un paramètre de requête HTTP ou un header client arbitraire :

1. **Claim JWT** `tenant_slug` (prioritaire) — porté par le cookie de session ; fonctionne même si un client utilise un domaine personnalisé.
2. **Sous-domaine du hostname** (repli) — pour les routes sans JWT ; `TenantSlugResolver` rejette les IP, `localhost`, le préfixe `api`, et exige au moins 3 composants (donc `centre-elamal.bjschool.tn` est valide, `bjschool.tn` seul ne l'est pas).

Ce mécanisme est ce qui permet à `app.bjschool.tn` de servir de portail d'accueil pendant que chaque centre a son propre espace isolé sur `{slug}.bjschool.tn`.

Bug P0-1 — piège classique à éviter

Le filtre global EF Core doit référencer la **propriété d'instance**

du contexte (`this.TenantId`), jamais une constante capturée à la construction du modèle — sinon EF Core fige le tenant du *premier* accès dans le modèle compilé (mis en cache par type de contexte), et toute requête suivante avec les mêmes `DbContextOptions` continue de

filtrer avec le mauvais tenant. C'est une fuite cross-tenant critique, couverte par un test de régression dédié.

`IgnoreQueryFilters()` est **interdit** en dehors du contexte Super-Admin explicitement justifié.

Sécurité

Règles non négociables (bloquantes avant tout merge) :

#	Règle
SEC-01	Tokens JWT en cookies <code>HttpOnly; Secure; SameSite=Strict</code> — jamais <code>localStorage</code>
SEC-02	Rate limiting sur les endpoints d'auth : 5 requêtes / 15 min / IP en production
SEC-03	Rotation atomique du refresh token après chaque usage
SEC-04	Filtre EF Core global <code>TenantId</code> sur toutes les entités <code>ITenantEntity</code>
SEC-05	Validation FluentValidation sur toutes les Commands
SEC-06	Mots de passe hachés BCrypt, cost ≥ 12
SEC-07	Secrets (<code>.env</code> , <code>appsettings.Local.json</code>) toujours hors git
SEC-08	Garde applicative sur <code>SaveChangesAsync</code> : toute écriture avec un <code>TenantId</code> différent du
contexte lève une exception d'isolation → 403 neutre + log critique	
SEC-09	Rôle PostgreSQL dédié par tenant, identifiants chiffrés AES-GCM

Durées de vie des tokens : access token 15 minutes, refresh token 7 jours (révocable par `jti`).

Écart connu à corriger avant commercialisation

Le rejet effectif des accès pour un tenant

Suspendu

ou

Résilie

(règle R-MT-03) n'est, au moment de la rédaction de ce guide, pas encore implémenté au niveau du middleware — seul le statut est modélisé côté domaine (`Tenant.Status` , `SuspendTenantCommand` ...). Un tenant suspendu ne se voit donc pas encore bloqué techniquement.

Acceptable tant qu'aucun tenant payant réel n'existe ; à corriger avant le premier client facturé. Voir aussi la checklist du [guide administrateur](#).

Autres points de vigilance avant une mise en production commerciale : headers de sécurité HTTP (CSP, HSTS, X-Frame-Options) côté Caddy, hardening SSH/UFW/fail2ban de la VM, et décision sur le câblage réel de Redis comme backplane SignalR.

Conventions de code

- **Nommage** : Command = `VerbNomCommand` , Query liste = `ListNomQuery` , Query détail = `GetNomQuery` , DTO = `NomDto` , Handler = `{Command}Handler` .
- **Tables PostgreSQL** : `snake_case` pluriel (`clients` , `lignes_commande` , `grilles_tarifaires`), index explicites sur toutes les FK et colonnes filtrées fréquemment.
- **EF Core** : toujours des migrations, jamais `EnsureCreated()` en production ; `SaveChangesAsync()` uniquement via `IUnitOfWork` .
- **SignalR** : un Hub authentifié rejoint un groupe dérivé d'un claim JWT (jamais d'un paramètre client) ; le Hub du portail public est une exception assumée et documentée (aucune donnée personnelle n'y transite).

- **Composants Vue** : les composants shadcn-vue générés dans `src/components/ui/` ne se modifient pas à la main — on les régénère via `npx shadcn-vue@latest add <nom> --overwrite`. Les tokens de marque vivent uniquement dans `src/styles/theme.css`, source de vérité unique des couleurs.
- **i18n** : toutes les chaînes visibles passent par les fichiers `src/locales/fr.json` et `src/locales/ar.json` (jamais de texte en dur dans un composant) ; l'arabe est en RTL complet.

Tests

- **xUnit + Moq** (`MockBehavior.Strict`, aucun mock permissif) + **FluentAssertions** (rester en 7.x — la v8 est passée en licence commerciale) + **Testcontainers** pour les tests d'intégration avec une vraie base PostgreSQL éphémère.
- Chaque cas d'usage critique métier a un test `UC_{CODE}_{Titre}_{Condition}` dans `SchoolBoost.Domain.Tests/Scenarios/`.
- Un **test IDOR** est obligatoire avant toute mise en production : `GET /api/v1/{ressource}/{id-tenant-A}` avec un jeton du tenant B doit renvoyer `404`, jamais `200`.
- Frontend : Vitest (unitaire) + Playwright (bout-en-bout, `playwright.config.ts`).

Environnement de développement local

```
# Backend
cd SchoolBoost-api
dotnet restore
docker compose -f docker-compose.dev.yml up -d # PostgreSQL + Redis locaux
dotnet run --project src/SchoolBoost.API

# Frontend
cd SchoolBoost-front
npm install
npm run dev # serveur de développement Vite
npm run build # vue-tsc -b && vite build
npm run test # Vitest
```

En local, l'API applique la politique CORS `Dev` (origine `http://localhost:5173`) ; en production, uniquement `https://app.bjschool.tn` (ou le domaine du tenant concerné). Les cookies de session sont émis avec `Secure = !Host.Equals("localhost")`.

Packages internes AMBJ Core

```
<PackageReference Include="AMBJ.Core.Domain" Version="3.1.0" />
<PackageReference Include="AMBJ.Core.Application" Version="3.1.0" />
<PackageReference Include="AMBJ.Core.Infrastructure" Version="3.1.0" />
```

Source : `git@github.com:ambj-consulting/ambj-consulting-core.git`, distribué également via GitHub Packages (`https://nuget.pkg.github.com/ambj-consulting/index.json`, authentification par PAT scope `SchoolBoost-api / SchoolBoost-front`).

Plafonds de version pour raisons de licence

`MediatR` reste sur la branche 12.x (licence commerciale depuis la v13) et `FluentAssertions` sur la branche 7.x maximum (licence commerciale depuis la v8) — ne pas monter au-delà sans validation explicite du coût de licence avec l'utilisateur.

Ce dépôt (bjschool-site)

Contrairement à `SchoolBoost-front`, ce site vitrine est volontairement simple : pages HTML statiques, CSS écrit à la main (pas de pipeline Tailwind — l'accès au registre npm était indisponible dans l'environnement où ce site a été généré ; à réévaluer si l'équipe souhaite adopter Tailwind comme sur `bjmarket-site`). Aucune étape de build n'est nécessaire : les fichiers sont servis tels quels par le Caddy de la VM Dataxon, sur le même modèle que `bjmarket-site`. Le contenu des manuels (`manuel/*.html`) est généré depuis des sources Markdown (`build_docs.py` et `content_*.py`) — modifiez le Markdown, pas le HTML généré.