6. Script de création / modification de la base de données

> Version 1.0 — Ce document décrit **comment initialiser** la base MongoDB (référentiels, compte admin) et **assurer la création des index** au démarrage, à partir des scripts fournis dans le backend.

6.1 Objectifs

- **Tester la connexion** à MongoDB et arrêter proprement en cas d'échec.
- **Créer/mettre à jour les index** (unicité, géo, texte, partiels, collation).
- **Seeder les référentiels** (types, tailles, attributs) et **l'utilisateur admin**.

6.2 Pré-requis & variables d'environnement

- **MongoDB Atlas** (ou compatible) accessible via URI.
- Variables d'environnement attendues :
  - `MONGODB_URI` (ou équivalent utilisé par l'app)
  - `ADMIN_USERNAME`, `ADMIN_EMAIL`, `ADMIN_PASSWORD`
  - (éventuellement) `SMTP_*` si la vérification email est testée

> **Sécurité** : ne versionnez jamais vos valeurs réelles. Utilisez un `.env` local ou des *repository secrets* côté CI/CD.

6.3 Données seedées

Répertoires (backend) : `data/seeds/*.json`

- `cache_types.json`
- `cache_sizes.json`
- `cache_attributes.json`

> Les jeux de données sont insérés **si la collection est vide** (ou **forcés** avec l'option `--force`).

6.4 Index créés (via seeding)

Les index sont créés/assurés de façon **idempotente** (création, ou *drop & recreate* si options différentes).

- **users** : unicité insensible à la casse sur `username`, `email` (collation) ; index géo `location (2dsphere)` ; flags `is_active`, `is_verified`.
- **countries** : `name` (unique), `code` (unique partiel si string).
- **states** : `country_id` ; `(country_id, name)` unique ; `(country_id, code)` unique partiel.
- **cache_attributes** : `cache_attribute_id` (unique), `txt` (unique partiel), `name`.
- **cache_sizes** : `name` (unique), `code` (unique partiel).

- **cache_types** : `name` (unique), `code` (unique partiel).
- **caches** : `GC` (unique), `loc (2dsphere)`, `type_id`, `size_id`, `country_id`, `state_id`, `(country_id, state_id)`, `difficulty`, `terrain`, `placed_at (desc)`, index **texte** `(title, description_html)`, et combinaisons métier : `(attributes.attribute_doc_id, attributes.is_positive)`, `(type_id, size_id)`, `(difficulty, terrain)`.
- **found_caches** : `(user_id, cache_id)` (unique), `(user_id, found_date)`, `cache_id`.
- **challenges** : `cache_id` (unique), index **texte** `(name, description)`.
- **user_challenges** : `(user_id, challenge_id)` (unique), `user_id`, `challenge_id`, `status`, `(user_id, status, updated_at)`.
- **user_challenge_tasks** : `(user_challenge_id, order)`, `(user_challenge_id, status)`, `user_challenge_id`, `last_evaluated_at`.
- **progress** : `(user_challenge_id, checked_at)` (unique).
- **targets** : `(user_challenge_id, cache_id)` (unique), `(user_challenge_id, satisfies_task_ids)`, `(user_challenge_id, primary_task_id)`, `cache_id`, `(user_id, score)`, `(user_id, user_challenge_id, score)`, `loc (2dsphere)`, `(updated_at, created_at)`.

6.5 Exécution (local, sans Docker)

Depuis la racine du backend :

```
# Activez votre venv au besoin puis :
python -m app.db.seed_data             # ping + ensure_indexes + seed référentiels + admin
python -m app.db.seed_data --force     # idem, mais vide les collections de référentiels avant réinsertion
```

> **Attention** : `--force` réinitialise les collections de référentiels (pas les données utilisateur).

6.6 Exécution (via Docker Compose)

Exemples indicatifs (adapter le service et le runner Python selon votre compose) :

```
# 1) Bâtir et lancer les conteneurs
docker compose up -d --build

# 2) Exécuter le seeding dans le conteneur backend
# (remplacez <backend> par le nom réel du service backend dans le docker-compose.yml)
docker compose exec <backend> python -m app.db.seed_data --force
```

> Les secrets (URI Mongo, admin) doivent être injectés au conteneur via l'environnement (compose, variables Railway, etc.).

6.7 Composants clés (extraits commentés)

6.7.1 Assurer les index (idempotent)

```
# app/db/seed_indexes.py — extrait simplifié
from pymongo import ASCENDING, DESCENDING, TEXT
from pymongo.operations import IndexModel
from pymongo.collation import Collation
from app.db.mongodb import get_collection

COLLATION_CI = Collation(locale="en", strength=2)  # insensible à la casse

def ensure_index(coll_name, keys, *, name=None, unique=None, partial=None,
collation=None):
    coll = get_collection(coll_name)
    # ... comparaison index existant / options ...
    opts = {}
    if name: opts['name'] = name
    if unique is not None: opts['unique'] = unique
    if partial: opts['partialFilterExpression'] = partial
    if collation is not None: opts['collation'] = collation
    coll.create_indexes([IndexModel(keys, **opts)])

# Exemple : unicité insensible à la casse
ensure_index('users', [('username', ASCENDING)], name='uniq_username_ci',
unique=True, collation=COLLATION_CI)
```

6.7.2 Seeding des référentiels & admin

```
# app/db/seed_data.py — extrait simplifié
from app.db.mongodb import db as mg_db, get_collection
from app.db.seed_indexes import ensure_indexes
from app.core import security

def seed_collection(file_path, collection_name, force=False):
    count = mg_db[collection_name].count_documents({})
    if count > 0 and not force:
        return
    if force:
        mg_db[collection_name].delete_many({})
    mg_db[collection_name].insert_many(json.load(open(file_path)))

def seed_admin_user():
    coll = get_collection("users")
    pwd_hash = security.pwd_context.hash(os.getenv("ADMIN_PASSWORD"))
    coll.update_one({"username": os.getenv("ADMIN_USERNAME")},
                    {"$set": {..., "password_hash": pwd_hash, "role":
"admin"},
                     "$setOnInsert": {"created_at": now()}}, upsert=True)
```

```
if __name__ == "__main__":
    test_connection()   # ping Mongo
    ensure_indexes()    # création/MAJ idempotente des index
    seed_referentials(force="--force" in sys.argv)
```

6.8 Bonnes pratiques & sécurité

- **Idempotence** : relancer le script ne casse pas les index existants si la configuration n'a pas changé.
- **Collation** : utilisez des collations cohérentes pour les unicités *case-insensitive* (ex. utilisateurs).
- **Indexes géo & texte** : un seul index **texte** par collection ; vérifiez la présence des `2dsphere` pour les requêtes cartographiques.
- **Quotas & coûts** : la multiplication d'index a un **coût d'écriture** ; validez les index réellement utiles via vos workloads.
- **Secrets** : stockez l'URI Mongo et le mot de passe admin via variables d'environnement (jamais en clair).

6.9 Vérification après exécution

Checklist rapide (via mongosh) :

```
use <your_db>
// Exemples
db.users.getIndexes()
db.caches.getIndexes()
db.challenges.getIndexes()
```

Vous devez retrouver les index listés en **6.4**.

6.10 Tests associés au seeding

Un test Pytest permet de vérifier que l'environnement backend accède bien à MongoDB :

```
# backend/tests/test_connectivity.py
from app.db.mongodb import client as mg_client

def test_backend_can_access_mongo():
    dbs = mg_client.list_database_names()
    assert isinstance(dbs, list)
```

- Vérifie que la connexion fonctionne et qu'une liste de bases est renvoyée.
- Peut être lancé seul pour diagnostiquer un problème de connexion.
- Intégré dans la suite Pytest pour automatiser le contrôle lors du CI/CD.