Configuration de l'environnement (Cr7)

Cr7.1 – Exactitude de l'installation

L'environnement de développement a été configuré avec précision afin d'assurer une compatibilité totale avec les environnements de production.

Versions et outils utilisés :

- **Node.js v20+** : Télécharger Node.js
- **npm v10+** : inclus avec Node.js
- **Visual Studio Code** : Télécharger VSCode
- **Docker Desktop** : Télécharger Docker
- **Git** : Télécharger Git

Dépendances principales :

- express → création du serveur HTTP et gestion des routes
- mongoose → connexion et gestion des modèles MongoDB
- mariadb / mysql2 → gestion des modèles relationnels SQL
- dotenv → gestion sécurisée de la configuration (variables d'environnement)
- swagger-ui-express & swagger-jsdoc → documentation automatique de l'API REST
- bcryptjs & jsonwebtoken → sécurité et gestion des tokens JWT

Docker & Docker Compose :
L'environnement repose sur des conteneurs isolés et reproductibles :

- conceptionbackend-mariadb-1 → **MariaDB 10.11**
- conceptionbackend-mongo-1 → **MongoDB 6**

Tests de débogage :

- Autocomplétion et breakpoints configurés dans **VSCode** via un fichier launch.json.
- Débogage Node.js fonctionnel en mode développement.

Cr7.2 – Utilisation du terminal

Toutes les commandes liées à l'environnement sont exécutées via le terminal :

```
# Installation des dépendances
npm install

# Lancement du serveur local (mode dev)
npm run dev

# Démarrage complet de l'environnement Docker
docker compose up -d

# Visualisation des logs de l'API
docker logs conceptionbackend-api-1
```

Les commandes sont compatibles avec un environnement **Dockerisé** et ne nécessitent aucune dépendance installée localement hors Docker.

Cr7.3 – Personnalisation de l'IDE

L'environnement de développement a été adapté aux besoins spécifiques du projet :

Extensions installées dans VSCode :

- ESLint (formatage et linting automatique)
- Prettier (mise en forme du code)
- Docker (gestion et logs de conteneurs)

- MongoDB for VSCode (requêtes et visualisation)
- REST Client / Thunder Client (tests d'API)

Personnalisation de l'interface :

- Thème : *Dark+ personnalisé*
- Raccourcis clavier configurés pour le débogage Node.js
- Snippets internes pour générer rapidement un **controller**, **model** ou **route**

Conception et implémentation de la solution Back-End (Cr8)

Cr8.1 – Qualité de la conception POO

Le projet applique une architecture orientée objet claire :

- **Modèles Sequelize-like** (MariaDB) et **Mongoose** (MongoDB) pour l'encapsulation des entités.
- **Contrôleurs** dédiés à chaque entité (User, Game, Config).
- **Services** manipulant les modèles pour une séparation stricte des responsabilités.

Principes appliqués :

- **Encapsulation** : les données sont gérées uniquement via leurs modèles.
- **Polymorphisme** : méthodes réutilisables pour manipuler différents types d'entités.
- **Responsabilité unique** : chaque couche a un rôle défini (contrôleur → logique métier, modèle → structure de données).

Cr8.2 – Utilisation de l'architecture MVC

Le projet respecte le pattern **MVC (Model – View – Controller)** :

```
src/
 ├── controllers/   → Logique métier (authController.js, gamesController.js…)
 ├── models/        → Schémas MariaDB et MongoDB
 ├── routes/        → Endpoints REST protégés par JWT
 ├── middlewares/   → Authentification, gestion des erreurs
 ├── swagger/       → Génération dynamique de la documentation Swagger
 └── server.js      → Point d'entrée du serveur
```

Cr8.3 – Clarté du code

- Indentation : 2 espaces (standard JavaScript ES6)
- Variables explicites : getGames, addGameToUser, registerUser, etc.
- Commentaires JSDoc sur les fonctions principales.
- Documentation interactive Swagger disponible à :
  http://localhost:3000/api/docs

Configuration des serveurs Web (Cr9)

Cr9.1 – Gestion des ressources

L'orchestration via Docker Compose permet une isolation et une gestion optimisée des ressources :

Volumes persistants :

```
volumes:
  - mariadb_data:/var/lib/mysql
  - mongo_data:/data/db
```

Ports exposés :

- API : 3000
- MariaDB : 3306
- MongoDB : 27017

Les conteneurs peuvent être redémarrés sans perte de données.

Cr9.2 – Sécurité

- Variables sensibles stockées dans .env (jamais versionné).
- Prise en charge HTTPS via certificat local ou proxy Nginx.
- Connexions sécurisées à la base de données avec mots de passe forts.
- Authentification **JWT obligatoire** sur toutes les routes protégées (middleware authMiddleware.js).

Conception et administration des BD relationnelles (Cr10)

Cr10.1 – Conformité du schéma

Le schéma **MariaDB** répond aux besoins fonctionnels du projet :

- Tables : users, games, user_games
- Relations **1-N** entre utilisateurs et jeux
- Contraintes : clés primaires, étrangères et index uniques

Cr10.2 – Normalisation

- Données respectant la **3e forme normale (3NF)**

- Aucune redondance de données
- Utilisation de clés étrangères cohérentes
- Intégrité référentielle assurée

Cr10.3 – Performance et sécurité

- Indexation sur user_id et game_id
- Requêtes préparées → protection contre les injections SQL
- Rôles et droits restreints au niveau de l'API pour chaque entité

Conception et administration des BD NoSQL (Cr11)

Cr11.1 – Adéquation de l'architecture NoSQL

MongoDB gère la configuration dynamique des jeux pour chaque utilisateur :

```
{
  "userId": 1,
  "gameId": 3,
  "settings": {
    "screenWidth": 1920,
    "screenHeight": 1080,
    "fps": 60,
    "volume": 80,
    "difficulty": "medium"
  }
}
```

Structure flexible, adaptée à l'évolution des paramètres utilisateurs.

Cr11.2 – Optimisation

- Indexation sur userId et gameId
- Scalabilité horizontale grâce au **sharding** et à la réplication
- Requêtes rapides via **Mongoose ORM**
- Séparation claire entre données structurées (SQL) et dynamiques (NoSQL)

Arbitrage des solutions (Cr12)

Cr12.1 – Prise de décision

Critère	MariaDB	MongoDB
Structure	Données relationnelles, cohérentes	Données dynamiques, flexibles
Performance	Optimisée pour les transactions	Rapide pour la lecture JSON
Coût	Open Source	Open Source
Évolutivité	Verticale	Horizontale (sharding, cluster)
Sécurité	Auth SQL native, gestion d'utilisateurs	Auth intégrée et isolation

Arbitrage final :

- **MariaDB** pour la gestion structurée (utilisateurs, jeux)
- **MongoDB** pour les données dynamiques (configurations utilisateurs)

Stratégie de sauvegarde et de récupération (Cr13)

Cr13.1 – Rigueur du plan de sauvegarde

Une stratégie de sauvegarde complète a été mise en place à l'aide d'un script Python automatisé :

Fichier : scripts/backup_databases.py
Fonctionnalités :

- Création automatique d'un dossier horodaté : backups/YYYY-MM-DD_HH-MM/
- Sauvegarde des bases :
    - backup_mariadb.sql via mysqldump
    - backup_mongo.archive via mongodump

Exécution manuelle ou planifiée :

python scripts/backup_databases.py

Cr13.2 – Alignement avec la sécurité

- Sauvegardes locales hors du répertoire web.
- Fichiers datés pour assurer la traçabilité et la restauration.
- Alignement avec les politiques de sécurité et de continuité d'activité.

Restauration des bases :

```
# Restauration MariaDB
docker exec -i conceptionbackend-mariadb-1 sh -c 'mysql -u root
-p"$MYSQL_ROOT_PASSWORD" maets' < backup_mariadb.sql

# Restauration MongoDB
docker exec -i conceptionbackend-mongo-1 sh -c 'mongorestore --archive=<
backup_mongo.archive'
```