

API Port de Plaisance Russell



DEMO RENDER

VISIT

Application **Node.js + Express + MongoDB** pour la gestion :

- des **catways** (emplacements bateaux),
- des **réservations**,
- des **utilisateurs** (avec rôles **admin** / **user**),
- avec une **interface HTML/EJS** intégrée **et** une **documentation Swagger**.

L'authentification se fait via **JWT** (stocké dans un cookie **httpOnly** ou dans l'en-tête **Authorization: Bearer <token>**).



Versions disponibles

- **Version Développement (publique – ce repo)**
 - Exécution locale : `http://localhost:3000`
 - Sert de support pédagogique, sans secrets sensibles dans le code.
 - **Version Production (privée – déployée)**
 - Déployée automatiquement sur **Render**
 - URL : `https://devoir-api-port-de-plaisance-russell-prod.onrender.com`
 - Contient les vraies variables d'environnement (MongoDB Atlas, SECRET_KEY, etc.)
 - Accessible avec des comptes **user** et **admin** fournis pour les tests.
-



Installation & Lancement

1. Cloner le projet

```
git clone https://github.com/CL4P-TP-afk/Devoir_API_Port_de_Plaisance_Russell_Dev
```

2. Installer les dépendances

```
npm install
```

3. Configurer les variables d'environnement

Créer un fichier `env/.env` (ou adapter ton `.env` si tu utilises `env-cmd`) :

```
env
```

```
PORT=3000
MONGO_URI=mongodb+srv://<user>:<password>@<cluster>/<db>?
retryWrites=true&w=majority
MONGO_DBNAME=apinode
SECRET_KEY=un_secret_tres_long_et_complexe
```

4. Importation des jeux de données initiales

- ⚠ Réinitialise complètement la base et recharge le jeu de données immuable seed-init/ :

```
npm run seed:init
```

5. Lancer l'application

```
npm start
```

⚠ Nécessite une base MongoDB locale ou distante.

📁 Structure du projet

```
.
├── app.js                # Point d'entrée Express
├── bin/www               # Lanceur serveur
├── db/mongo.js           # Connexion MongoDB
├── routes/               # Définition des routes (API + vues HTML)
│   ├── auth.js
│   ├── catways.js
│   ├── reservations.js
│   ├── users.js
│   ├── dashboard.js
│   ├── reserver.js
│   └── docs.js
├── models/               # Schémas Mongoose (User, Catway, Reservation)
├── services/             # Logique métier (validation, règles)
├── middlewares/          # Authentification & rôles
├── views/                # Templates EJS (UI)
├── swagger.yaml           # Documentation OpenAPI 3
├── swagger.js            # Branche Swagger sur /api-docs
└── public/               # Fichiers statiques (CSS/JS)
```

🔑 Authentification & Rôles

JWT signé avec **SECRET_KEY**.

- Stocké dans cookie **httpOnly** ou envoyé dans **Authorization: Bearer** .

- Rôles :
 1. admin : accès complet (gestion utilisateurs, catways, réservations).
 2. user : accès limité (consultation et création de réservations).

Documentation API

- Version intégrée : <http://localhost:3000/docs> (affiche Swagger dans une page EJS avec bouton retour vers l'UI).
- Version Swagger UI brute : <http://localhost:3000/api-docs> (pleine page Swagger, directement générée depuis swagger.yaml).

Endpoints principaux

1. Auth

- POST /auth/login → connexion (JWT + cookie, redirection vers dashboard).
- GET /auth/logout → déconnexion (suppression cookie, retour à l'accueil).

2. Catways

- GET /catways → liste des catways (HTML).
- POST /catways → créer un catway.
- PUT /catways/{id} → modifier état/réservabilité.
- DELETE /catways/{id} → supprimer un catway.

3. Réservations

- GET /catways/{id}/reservations → liste des réservations d'un catway.
- POST /catways/{id}/reservations → créer une réservation.
- PUT /catways/{id}/reservations/{idReservation} → modifier une réservation.
- DELETE /catways/{id}/reservations/{idReservation} → supprimer une réservation.

4. Utilisateurs

- GET /users → liste/gestion des utilisateurs (HTML).
- POST /users → créer un utilisateur.
- PUT /users/{id} → mettre à jour un utilisateur.
- DELETE /users/{id} → supprimer un utilisateur.

Règles métier principales

1. Catways

- catwayNumber unique et ≥ 1 .
- catwayType $\in \{\text{long}, \text{short}\}$.
- isReservable = true/false (par défaut true).

2. Réservations

- clientName doit exister dans User.name.
- Refus si catway.isReservable === false.
- Refus si chevauchement de dates pour un même catway.
- Dates : format date ou date-time.

3. Utilisateurs

- name ≥ 3 caractères.
- email valide et unique.
- password fort (minuscule + majuscule + chiffre).
- role $\in \{\text{admin}, \text{user}\}$.
- password hashé avec bcrypt.



Scénario de test rapide

- Créer un compte admin (/users via interface ou seed).
- Connexion : /auth/login → cookie JWT.
- Dashboard : /dashboard → navigation vers menus.
- Créer un catway : /catways (ex. n°1, type short).
- Créer une réservation : /catways/{id}/reservations.
- Vérifier qu'un chevauchement de dates est refusé.
- Consulter la doc : /docs ou /api-docs.



Commandes utiles

Lancer avec env-cmd (env-cmd -f ./env/.env nodemon ./bin/www) :

```
env-cmd -f ./env/.env nodemon ./bin/www
```

Lancer en dev avec nodemon (`env-cmd -f ./env/.env.dev nodemon ./bin/www`) :

```
npm run dev
```


Lancer en prod avec nodemon (`env-cmd -f ./env/.env.prod nodemon ./bin/www`) :

```
npm run prod
```

Import/export de données :

- importer sans rien effacer (ajoute/ignore doublons) :


```
npm run seed
```

- importer en purgeant ( → reset complet) :

```
npm run seed:reset
```

- Importer le jeu de secours immuable seed-init/ :

```
npm run seed:init
```

- Sauvegarder l'état actuel de la base vers JSON ( → écrase seed/*.json avec les données en DB) :

```
npm run dump
```

Notes sécurité

- **JWT** stocké en cookie `httpOnly` → limite les risques **XSS**.
- Vérification stricte des rôles (`isAdmin`) sur routes sensibles.
- Validation de toutes les entrées côté serveur (Mongoose + services).
- Mot de passe jamais stocké en clair (toujours hashé).