

Tutoriel Complet : Créer une API Node.js Star Wars Farm

Table des matières

1. [Prérequis et installation](#)
2. [Initialisation du projet](#)
3. [Création de l'architecture](#)
4. [Configuration de base](#)
5. [Création de l'application Express](#)
6. [Ajout des routes API](#)
7. [Intégration de Swagger](#)
8. [Création des tests](#)
9. [Configuration Docker](#)
10. [Mise en place CI/CD](#)
11. [Déploiement GitHub Pages](#)
12. [Lancement et test](#)
13. [Résolution des problèmes](#)

0. Prérequis et installation

0.1 Outils de base requis

Node.js et npm

- **Téléchargement** : <https://nodejs.org/>
- **Version recommandée** : 18.x ou supérieure
- **Vérification** :

```
node --version
npm --version
```

Git

- **Téléchargement** : <https://git-scm.com/>
- **Vérification** :

```
git --version
```

Éditeur de code

- **Visual Studio Code** (recommandé) : <https://code.visualstudio.com/>
- **Extensions recommandées** :
 - Node.js Extension Pack
 - Docker
 - GitLens
 - REST Client

0.2 Outils de développement

Docker Desktop

- **Téléchargement** : <https://www.docker.com/products/docker-desktop/>
- **Installation** :
 1. Téléchargez Docker Desktop pour Windows
 2. Installez en tant qu'administrateur
 3. Redémarrez votre ordinateur
 4. Lancez Docker Desktop
 5. Attendez que l'icône devienne verte
- **Vérification** :

```
docker --version
docker ps
```

GitHub CLI (optionnel mais recommandé)

- **Téléchargement** : <https://cli.github.com/>
- **Installation** :

```
# Windows (avec winget)
winget install GitHub.cli

# Ou téléchargement manuel depuis le site officiel
```

- **Vérification** :

```
gh --version
```

0.3 Bibliothèques et frameworks

Bibliothèques Node.js principales

```
# Framework web
npm install express

# Utilitaires JavaScript
npm install underscore

# Documentation API
npm install swagger-ui-express swagger-jsdoc

# Tests
npm install --save-dev mocha supertest nyc

# Types TypeScript (optionnel)
npm install --save-dev @types/express @types/underscore @types/supertest
```

Outils de développement

```
# Linter et formateur de code
npm install --save-dev eslint prettier

# Outils de build (optionnel)
npm install --save-dev nodemon

# Outils de sécurité
npm install --save-dev npm-audit-fix
```

0.4 Configuration de l'environnement

Variables d'environnement

Créez un fichier `.env` à la racine du projet :

```
NODE_ENV=development
PORT=8080
```

Configuration Git

```
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"
```

Configuration npm

```
npm config set init-author-name "Votre Nom"
npm config set init-author-email "votre.email@example.com"
npm config set init-license "MIT"
```

0.5 Vérification de l'installation

Script de vérification

Créez un fichier `check-prerequisites.js` :

```
const { execSync } = require('child_process');

console.log('🔍 Vérification des prérequis...\n');

const tools = [
  { name: 'Node.js', command: 'node --version' },
  { name: 'npm', command: 'npm --version' },
  { name: 'Git', command: 'git --version' },
  { name: 'Docker', command: 'docker --version' }
];

tools.forEach(tool => {
  try {
    const version = execSync(tool.command, { encoding: 'utf8' }).trim();
    console.log(`✅ ${tool.name}: ${version}`);
  } catch (error) {
    console.log(`❌ ${tool.name}: Non installé`);
  }
});

console.log('\n🎉 Si tous les outils sont installés, vous pouvez commencer le tutoriel !');
```

Exécutez le script :

```
node check-prerequisites.js
```

0.6 Résolution des problèmes courants

Problème Docker Desktop

- Erreur "Docker Desktop is not running" :
 1. Ouvrez Docker Desktop
 2. Attendez que l'icône devienne verte
 3. Redémarrez votre terminal
- Erreur "port already in use" :

```
# Windows
netstat -ano | findstr :8080
taskkill /PID [PID] /F

# Linux/Mac
lsof -i :8080
kill -9 [PID]
```

Problème npm

- Erreur de permissions :

```
# Windows (PowerShell en tant qu'administrateur)
Set-ExecutionPolicy RemoteSigned

# Linux/Mac
sudo chown -R $USER:$GROUP ~/.npm
```

Problème Git






- Configuration SSH (pour GitHub) :

```
ssh-keygen -t ed25519 -C "votre.email@example.com"
# Ajoutez la clé publique à votre compte GitHub
```

☒ Une fois tous les prérequis installés et vérifiés, vous pouvez commencer le tutoriel !

Architecture du projet - Rôle de chaque dossier et fichier

Vue d'ensemble de l'architecture

SimplonWars-farm-nodejs/	
	FICHIERS DE CONFIGURATION
├──	package.json # Configuration du projet Node.js
├──	package-lock.json # Verrouillage des versions des dépendances
├──	.gitignore # Fichiers à ignorer par Git
├──	.dockerignore # Fichiers à ignorer par Docker
└──	README.md # Documentation du projet
	CODE DE L'APPLICATION
├──	app.js # Point d'entrée principal de l'application
├──	public/ # Fichiers statiques (HTML, CSS, images)
│	├── index.html # Page d'animation Star Wars
│	├── css/
│	│ └── style.css # Styles de l'animation Star Wars
│	└── fonts/ # Polices de caractères
	TESTS
├──	test/
│	└── test.js # Suite de tests automatisés
	DOCKER
├──	Dockerfile # Recette pour créer l'image Docker
└──	.dockerignore # Fichiers à exclure de l'image Docker
	CI/CD
├──	.github/
│	├── workflows/ # Pipelines GitHub Actions
│	│ └── test.yml # Pipeline de tests automatiques
│	└── docker.yml # Pipeline de build et publication Docker

FICHIERS DE CONFIGURATION

package.json

- **Rôle** : Configuration principale du projet Node.js
- **Contient** : Nom, version, dépendances, scripts, métadonnées
- **Utilisation** : `npm install` lit ce fichier pour installer les dépendances
- **Exemple** : `npm start` lance le script défini dans ce fichier

package-lock.json

- **Rôle** : Verrouillage exact des versions des dépendances
- **Contient** : Versions précises de toutes les dépendances et sous-dépendances
- **Utilisation** : Garantit que tous les développeurs utilisent les mêmes versions
- **Important** : Ne jamais modifier manuellement ce fichier

.gitignore

- **Rôle** : Définit les fichiers que Git doit ignorer
- **Contient** : Patterns de fichiers à ne pas versionner
- **Exemples** : `node_modules/`, `coverage/`, `.env`, fichiers temporaires
- **Utilisation** : Évite de commiter des fichiers sensibles ou générés

.dockerignore

- **Rôle** : Définit les fichiers que Docker doit ignorer lors du build
- **Contient** : Fichiers à ne pas copier dans l'image Docker
- **Exemples** : `.git/`, `README.md`, `test/`, fichiers de développement
- **Utilisation** : Réduit la taille de l'image Docker et améliore la sécurité

README.md

- **Rôle** : Documentation principale du projet
- **Contient** : Description, installation, utilisation, API
- **Utilisation** : Première chose que les développeurs lisent
- **Important** : Doit être clair et à jour

CODE DE L'APPLICATION

app.js

- **Rôle** : Point d'entrée principal de l'application
- **Contient** : Configuration Express, routes API, logique métier
- **Utilisation** : `node app.js` lance le serveur
- **Responsabilités** :
 - Configuration du serveur Express
 - Définition des routes API
 - Gestion des requêtes HTTP
 - Intégration Swagger

public/ (Dossier des fichiers statiques)

- **Rôle** : Contient tous les fichiers accessibles directement par le navigateur
- **Contenu** : HTML, CSS, JavaScript, images, polices
- **Utilisation** : Express sert automatiquement ces fichiers

public/index.html

- **Rôle** : Page d'animation Star Wars
- **Contient** : Structure HTML de l'animation
- **Utilisation** : Accessible via `/starwars`
- **Fonctionnalités** : Animation CSS, texte défilant, design responsive

public/css/style.css

- **Rôle** : Styles de l'animation Star Wars
- **Contient** : Animations CSS, keyframes, design
- **Utilisation** : Appliqué automatiquement à `index.html`
- **Fonctionnalités** : Animation du texte, effets visuels, responsive design

`public/fonts/`

- **Rôle** : Polices de caractères personnalisées
- **Contenu** : Fichiers de polices (.ttf, .woff, etc.)
- **Utilisation** : Polices utilisées dans l'animation Star Wars
- **Avantage** : Garantit l'affichage correct sur tous les navigateurs

TESTS

`test/` (Dossier des tests)

- **Rôle** : Contient tous les tests automatisés
- **Organisation** : Un fichier par type de test ou par module
- **Utilisation** : `npm test` lance tous les tests

`test/test.js`

- **Rôle** : Suite de tests principale
- **Contient** : Tests de toutes les routes API et pages
- **Outils** : Mocha (framework), Supertest (tests HTTP)
- **Couverture** : Tests fonctionnels, tests d'intégration

DOCKER

`Dockerfile`

- **Rôle** : Recette pour créer l'image Docker
- **Contient** : Instructions pour construire l'environnement
- **Utilisation** : `docker build -t nom-image .`
- **Étapes** : Installation Node.js, copie du code, installation dépendances

`.dockerignore`

- **Rôle** : Optimisation de l'image Docker
- **Contient** : Fichiers à exclure du build Docker
- **Avantages** : Image plus petite, build plus rapide, sécurité améliorée

CI/CD

`.github/` (Dossier GitHub)

- **Rôle** : Configuration spécifique à GitHub
- **Contenu** : Workflows, templates, configurations

`.github/workflows/`

- **Rôle** : Pipelines CI/CD automatisés
- **Contenu** : Fichiers YAML définissant les workflows
- **Utilisation** : Exécution automatique sur GitHub Actions

`.github/workflows/test.yml`

- **Rôle** : Pipeline de tests automatiques
- **Déclenchement** : Sur chaque push et Pull Request
- **Actions** : Installation dépendances, exécution tests
- **Objectif** : Garantir la qualité du code

`.github/workflows/docker.yml`

- **Rôle** : Pipeline de build et publication Docker
- **Déclenchement** : Sur chaque push vers main
- **Actions** : Build image Docker, publication sur registres
- **Objectif** : Déploiement automatique

FICHIERS GÉNÉRÉS (à ne pas modifier manuellement)

`node_modules/`

- **Rôle** : Dépendances installées par npm
- **Génération** : Créé automatiquement par `npm install`
- **Important** : Jamais commiter ce dossier (dans .gitignore)

`coverage/`

- **Rôle** : Rapports de couverture de tests
- **Génération** : Créé par nyc lors des tests
- **Contenu** : HTML, JSON avec statistiques de couverture

Fichiers de logs

- **Rôle** : Logs d'exécution de l'application
- **Exemples** : *.log, logs/
- **Important** : Toujours dans .gitignore

Bonnes pratiques d'organisation

Séparation des responsabilités :

- **Code source** : Dans la racine ou `src/`
- **Tests** : Dans `test/` ou `__tests__/`
- **Configuration** : Fichiers à la racine
- **Documentation** : Dans `docs/` ou à la racine

Nommage des fichiers :

- **kebab-case** : `my-file.js` (recommandé)
- **camelCase** : `myFile.js` (pour les modules)
- **PascalCase** : `MyComponent.js` (pour les classes)

Structure modulaire :

- Un fichier = une responsabilité
- Dossiers par fonctionnalité
- Import/export clairs

1. Initialisation du projet

1.1 Créer le dossier du projet

```
mkdir simplonwars-farm-nodejs
cd simplonwars-farm-nodejs
```

1.2 Initialiser le projet Node.js

```
npm init -y
```

1.3 Installer les dépendances de base

```
npm install express underscore
npm install --save-dev mocha supertest nyc
```

2. Création de l'architecture

2.1 Créer la structure des dossiers

```
mkdir public
mkdir public/css
mkdir public/fonts
mkdir test
mkdir .github
mkdir .github/workflows
```

2.2 Créer les fichiers de base

```
touch app.js
touch public/index.html
touch public/css/style.css
touch test/test.js
touch Dockerfile
touch .dockerignore
touch .gitignore
touch .github/workflows/test.yml
touch .github/workflows/docker.yml
touch README.md
```

3. Configuration de base

3.1 Configuration du package.json

Remplacez le contenu de package.json par :

```
{
  "name": "simplonwars-farm-nodejs",
  "license": "MIT",
  "version": "0.0.1",
  "dependencies": {
    "express": "4.x",
    "underscore": "^1.12.1",
    "swagger-ui-express": "^4.15.5",
    "swagger-jsdoc": "^6.2.8"
  },
  "scripts": {
    "test": "nyc --reporter=html mocha --exit",
    "start": "node app.js"
  },
  "devDependencies": {
    "@types/express": "^4.17.11",
    "@types/supertest": "^2.0.10",
    "@types/underscore": "^1.10.24",
    "mocha": "^8.2.1",
    "nyc": "^15.1.0",
    "supertest": "^6.1.3"
  }
}
```

3.2 Configuration du .gitignore

```
node_modules
coverage
.env
.DS_Store
*.log
```

3.3 Configuration du .dockerignore

```
node_modules
.git
.gitignore
README.md
*.md
test/
coverage/
.env
.env.local
.env.development
.vscode/
.idea/
*.swp
*.swo
.DS_Store
Thumbs.db
*.log
logs/
tmp/
temp/
```

4. Création de l'application Express

4.1 Créer le fichier app.js de base

```
// =====
// IMPORTS ET CONFIGURATION DE BASE
// =====

// Express.js : Framework web pour Node.js
// Permet de créer facilement des serveurs web et des APIs
const express = require('express');

// Underscore.js : Bibliothèque d'utilitaires JavaScript
// Fournit des fonctions utiles pour manipuler les tableaux et objets
const _ = require('underscore');

// Path : Module Node.js natif pour gérer les chemins de fichiers
// Permet de créer des chemins compatibles avec tous les systèmes d'exploitation
const path = require('path');

// =====
// BONNES PRATIQUES DE DÉCLARATION DE VARIABLES
// =====
// En JavaScript moderne, on utilise :
// - const : Pour les variables qui ne changent jamais (recommandé par défaut)
// - let : Pour les variables qui peuvent changer
// - var : ANCIENNE PRATIQUE - À ÉVITER (problèmes de scope)

// Configuration du port
// process.env.PORT : Variable d'environnement (utilisée en production)
// || 8080 : Valeur par défaut si la variable n'est pas définie
// const : Variable qui ne sera pas modifiée (bonne pratique moderne)
const port = process.env.PORT || 8080;

// =====
// BASE DE DONNÉES DES ANIMAUX STAR WARS
// =====

// Structure de données simple (objet JavaScript)
// Clé = nom de l'animal, Valeur = son de l'animal
// En production, on utiliserait une vraie base de données (MySQL, MongoDB, etc.)
// const : Objet qui ne sera pas réassigné (bonne pratique moderne)
const animals = {
  "bantha": "grumph",      // Animal de Tatooine (comme un chameau)
  "tauntaun": "rawrr",     // Animal de Hoth (comme un cheval)
  "nerf": "bleat",         // Animal de Naboo (comme une chèvre)
  "eopie": "snort",        // Animal de Tatooine (comme un âne)
  "blurr": "grunt",        // Animal de Malastare (comme un cochon)
  "porc": "chirp",         // Animal d'Ach-To (comme un oiseau)
  "fathier": "whinny",     // Animal de Canto Bight (comme un cheval)
  "taq": "squawk",         // Animal d'Endor (comme un perroquet)
  "reek": "bellow",        // Animal de Geonosis (comme un taureau)
  "dewback": "croak",      // Animal de Tatooine (comme un lézard)
  "nunas": "cluck",        // Animal de Naboo (comme une poule)
  "varactyl": "screech",   // Animal d'Utapau (comme un dinosaure)
  "happabore": "snuffle"   // Animal de Naboo (comme un sanglier)
}

// =====
// FONCTIONS UTILITAIRES
// =====

// Fonction pour obtenir un animal aléatoire
// _.sample() : Fonction Underscore qui choisit un élément aléatoire dans un tableau
// Object.entries() : Convertit l'objet en tableau de paires [clé, valeur]
function getAnimal() {
  return _.sample(Object.entries(animals));
}

// =====
// CRÉATION DE L'APPLICATION EXPRESS
// =====

// Créer une instance de l'application Express
const app = express();

// =====
// CONFIGURATION DES MIDDLEWARES
// =====

// Middleware pour servir les fichiers statiques
```

```
// path.join(__dirname, 'public') : Chemin vers le dossier 'public'
// __dirname : Variable Node.js qui contient le chemin du dossier actuel
// Cela permet d'accéder aux fichiers HTML, CSS, images via l'URL
app.use(express.static(path.join(__dirname, 'public')));

// =====
// DÉFINITION DES ROUTES
// =====

// Route principale (page d'accueil)
// app.get('/', ...) : Définit ce qui se passe quand quelqu'un visite '/'
// req : Objet request (contient les données de la requête)
// res : Objet response (permet d'envoyer une réponse au navigateur)
app.get('/', function(req, res){
  // Obtenir un animal aléatoire
  const [animal_name, sound] = getAnimal();

  // Définir le type de contenu de la réponse
  res.writeHead(200, { 'Content-Type': 'text/html' });

  // Créer le contenu HTML de la réponse
  // Template string avec ${} pour insérer des variables
  res.write(`George Orwell had a farm.<br />
E-I-E-I-O<br />
And on his farm he had a ${ animal_name }.<br />
E-I-E-I-O<br />
With a ${ sound }-${ sound } here.<br />
And a ${ sound }-${ sound } there.<br />
Here a ${ sound }, there a ${ sound }.<br />
Everywhere a ${ sound }-${ sound }.<br />`);

  // Terminer la réponse
  res.end();
});

// Route API (endpoint JSON)
// Cette route retourne toutes les données des animaux au format JSON
// Utile pour les applications qui veulent récupérer les données
app.get('/api', function(req, res){
  // Définir le type de contenu comme JSON
  res.writeHead(200, { 'Content-Type': 'application/json' });

  // Convertir l'objet animals en chaîne JSON et l'envoyer
  res.write(JSON.stringify(animals));

  // Terminer la réponse
  res.end();
});

// =====
// DÉMARRAGE DU SERVEUR
// =====

// module.exports : Permet d'exporter l'application pour les tests
// app.listen() : Démarre le serveur sur le port spécifié
// Callback : Fonction exécutée quand le serveur démarre
module.exports = app.listen(port, () => {
  console.log(`🚀 Serveur lancé sur http://localhost:${ port }`)
});
```

4.2 Tester l'application de base

```
npm start
```

Vérifiez que <http://localhost:8080> fonctionne.

5. Ajout des routes API

5.1 Installer Swagger

```
npm install swagger-ui-express swagger-jsdoc
```

5.2 Enrichir app.js avec les nouvelles routes

Remplacez le contenu de `app.js` par la version complète :

```
const express = require('express');
const _ = require('underscore');
const path = require('path');
const swaggerUi = require('swagger-ui-express');
const swaggerJsdoc = require('swagger-jsdoc');

const port = process.env.PORT || 8080;

// Base de données des animaux Star Wars
const animals = {
  "bantha": "grumph",
  "tauntaun": "rawrr",
  "nerf": "bleat",
  "eopie": "snort",
  "blurr": "grunt",
  "porg": "chirp",
  "fathier": "whinny",
  "taq": "squawk",
  "reek": "bellow",
  "dewback": "croak",
  "nunas": "cluck",
  "varactyl": "screech",
  "happabore": "snuffle"
}

// Base de données des planètes
const planets = {
```

```

    "bantha": "Tatooine",
    "tauntaun": "Hoth",
    "nerf": "Naboo",
    "eopie": "Tatooine",
    "blurr": "Malastare",
    "porc": "Ahch-To",
    "fathier": "Canto Bight",
    "taq": "Endor",
    "reek": "Geonosis",
    "dewback": "Tatooine",
    "nunas": "Naboo",
    "varactyl": "Utapau",
    "happabore": "Naboo"
  }
}

function getAnimal() {
  return _.sample(Object.entries(animals));
}

function getMultipleAnimals(count = 3) {
  const animalEntries = Object.entries(animals);
  const shuffled = _.shuffle(animalEntries);
  return shuffled.slice(0, Math.min(count, animalEntries.length));
}

const app = express();

// Fichiers statiques
app.use(express.static(path.join(__dirname, 'public')));

// Configuration Swagger
const swaggerOptions = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'Animal Farm Star Wars API',
      version: '1.0.0',
      description: 'API pour la ferme d\'animaux Star Wars',
    },
    servers: [
      {
        url: 'http://localhost:8080',
        description: 'Serveur de développement'
      }
    ]
  },
  apis: ['./app.js']
};

const swaggerSpec = swaggerJsdoc(swaggerOptions);
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));

// Routes
app.get('/', function(req, res){
  const [animal_name, sound] = getAnimal();
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.write('George Orwell had a farm.<br />
E-I-E-I-O<br />
And on his farm he had a ${ animal_name }.<br />
E-I-E-I-O<br />
With a ${ sound }-${ sound } here.<br />
And a ${ sound }-${ sound } there.<br />
Here a ${ sound }, there a ${ sound }.<br />
Everywhere a ${ sound }-${ sound }.<br />');
  res.end();
});

app.get('/starwars', function(req, res){
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

/**
 * @swagger
 * /api:
 *   get:
 *     summary: Récupérer tous les animaux Star Wars
 *     responses:
 *       200:
 *         description: Liste des animaux récupérée avec succès
 */
app.get('/api', function(req, res){
  res.json(animals);
});

/**
 * @swagger
 * /api/random:
 *   get:
 *     summary: Récupérer un animal Star Wars aléatoire
 *     responses:
 *       200:
 *         description: Animal aléatoire récupéré avec succès
 */
app.get('/api/random', function(req, res){
  const [animal_name, sound] = getAnimal();
  const planet = planets[animal_name] || "Inconnue";

  res.json({
    animal: animal_name,
    sound: sound,
    planet: planet
  });
});

```



```

/**
 * @swagger
 * /api/random/{count}:
 *   get:
 *     summary: Récupérer plusieurs animaux Star Wars aléatoires
 *     parameters:
 *       - in: path
 *         name: count
 *         schema:
 *           type: integer
 *           default: 3
 *           minimum: 1
 *           maximum: 10
 */
app.get('/api/random/:count', function(req, res){
  const count = parseInt(req.params.count) || 3;
  const limitedCount = Math.min(Math.max(count, 1), 10);

  const randomAnimals = getMultipleAnimals(limitedCount).map(([animal, sound]) => ({
    animal: animal,
    sound: sound,
    planet: planets[animal] || "Inconnue"
  }));

  res.json(randomAnimals);
});

/**
 * @swagger
 * /api/planet/{planetName}:
 *   get:
 *     summary: Récupérer les animaux d'une planète spécifique
 *     parameters:
 *       - in: path
 *         name: planetName
 *         required: true
 *         schema:
 *           type: string
 */
app.get('/api/planet/:planetName', function(req, res){
  const planetName = req.params.planetName;
  const planetAnimals = {};

  Object.entries(planets).forEach(([animal, planet]) => {
    if (planet.toLowerCase() === planetName.toLowerCase()) {
      planetAnimals[animal] = animals[animal];
    }
  });

  if (Object.keys(planetAnimals).length === 0) {
    res.status(404).json({
      error: "Planète non trouvée",
      message: `Aucun animal trouvé sur la planète ${planetName}`
    });
  } else {
    res.json({
      planet: planetName,
      animals: planetAnimals
    });
  }
});

/**
 * @swagger
 * /api/stats:
 *   get:
 *     summary: Récupérer les statistiques de l'API
 */
app.get('/api/stats', function(req, res){
  const totalAnimals = Object.keys(animals).length;
  const uniquePlanets = [...new Set(Object.values(planets))];
  const totalPlanets = uniquePlanets.length;

  const animalsPerPlanet = {};
  Object.values(planets).forEach(planet => {
    animalsPerPlanet[planet] = (animalsPerPlanet[planet] || 0) + 1;
  });

  res.json({
    totalAnimals: totalAnimals,
    totalPlanets: totalPlanets,
    animalsPerPlanet: animalsPerPlanet,
    planets: uniquePlanets
  });
});

/**
 * @swagger
 * /api/search/{query}:
 *   get:
 *     summary: Rechercher un animal par nom
 *     parameters:
 *       - in: path
 *         name: query
 *         required: true
 *         schema:
 *           type: string
 */
app.get('/api/search/:query', function(req, res){
  const query = req.params.query.toLowerCase();
  const results = {};

  Object.entries(animals).forEach(([animal, sound]) => {
    if (animal.toLowerCase().includes(query)) {

```

```

        results[animal] = {
          sound: sound,
          planet: planets[animal] || "Inconnue"
        };
      }
    });

    if (Object.keys(results).length === 0) {
      res.status(404).json({
        error: "Aucun animal trouvé",
        message: 'Aucun animal ne correspond à la recherche '${query}'`
      });
    } else {
      res.json({
        query: query,
        results: results
      });
    }
  });
});

module.exports = app.listen(port, () => {
  console.log(`🚀 Serveur lancé sur http://localhost:${ port }`)
  console.log(`📖 Documentation API disponible sur http://localhost:${ port }/api-docs`)
});

```

6. Intégration de Swagger

6.1 Tester Swagger

```
npm start
```

Puis ouvrez <http://localhost:8080/api-docs>

7. Création des tests

7.1 Créer le fichier test/test.js

```

// =====
// IMPORTS ET CONFIGURATION DES TESTS
// =====

// Importer l'application Express (le fichier app.js)
// require('../app.js') : Remonte d'un niveau depuis le dossier 'test'
const app = require('../app.js');

// Supertest : Bibliothèque pour tester les APIs HTTP
// Permet d'envoyer des requêtes HTTP à notre application et de vérifier les réponses
// request(app) : Crée un client de test connecté à notre application
const request = require('supertest')(app);

// =====
// SUITE DE TESTS
// =====

// describe() : Groupe de tests (comme un dossier)
// 'GET' : Nom du groupe de tests
// function() : Fonction qui contient tous les tests du groupe
describe('GET', function() {

  // =====
  // TEST 1 : PAGE D'ACCUEIL (ROUTE '/')
  // =====

  // it() : Test individuel
  // 'respond with text/html' : Description du test
  // function(done) : Fonction de test (done = callback pour signaler la fin)
  it('respond with text/html', function(done) {
    // request : Client de test Supertest
    request
      .get('/') // Envoyer une requête GET vers '/'
      .set('Accept', 'text/html') // Définir l'en-tête Accept pour demander du HTML
      .expect('Content-Type', /html/) // Vérifier que la réponse contient 'html'
      .expect(200, done); // Vérifier que le code de statut est 200 (OK)
  })

  // Test du contenu de la page d'accueil
  it('respond with George Orwell', function(done) {
    request
      .get('/') // Requête GET vers '/'
      .set('Accept', 'text/html') // Demander du HTML
      .expect(200, /George Orwell had a farm/ig, done); // Vérifier que le texte contient "George Orwell"
    // /George Orwell had a farm/ig : Expression régulière (regex)
    // i = insensible à la casse, g = recherche globale
  })

  // =====
  // TEST 2 : API (ROUTE '/api')
  // =====

  // Test du type de contenu de l'API
  it('/api responds with json', function(done) {
    request
      .get('/api') // Requête GET vers '/api'
      .set('Accept', 'application/json') // Demander du JSON
      .expect('Content-Type', /json/) // Vérifier que la réponse est du JSON
      .expect(200, done); // Vérifier le code 200
  })

  // Test du contenu exact de l'API
  it('/api responds with Star Wars animals object', function(done) {
    request

```

```

    .get('/api') // Requête GET vers '/api'
    .set('Accept', 'application/json') // Demander du JSON
    .expect(200, { // Vérifier le code 200 ET le contenu exact
      "bantha": "grumph", // Animal de Tatooine
      "tauntaun": "rawrr", // Animal de Hoth
      "nerf": "bleat", // Animal de Naboo
      "eopie": "snort", // Animal de Tatooine
      "blurr": "grunt", // Animal de Malastare
      "porc": "chirp", // Animal d'Ach-To
      "fathier": "whinny", // Animal de Canto Bight
      "taq": "squawk", // Animal d'Endor
      "reek": "bellow", // Animal de Geonosis
      "dewback": "croak", // Animal de Tatooine
      "nunas": "cluck", // Animal de Naboo
      "varactyl": "screech", // Animal d'Utapau
      "happabore": "snuffle" // Animal de Naboo
    }, done);
  })

// =====
// TEST 3 : PAGE STAR WARS (ROUTE '/starwars')
// =====

// Test de la page d'animation Star Wars
it('/starwars responds with text/html', function(done) {
  request
    .get('/starwars') // Requête GET vers '/starwars'
    .set('Accept', 'text/html') // Demander du HTML
    .expect('Content-Type', /html/) // Vérifier que c'est du HTML
    .expect(200, done); // Vérifier le code 200
})

// =====
// TEST 4 : DOCUMENTATION SWAGGER (ROUTE '/api-docs')
// =====

// Test de la documentation Swagger
it('/api-docs responds with html (Swagger UI)', function(done) {
  request
    .get('/api-docs') // Requête GET vers '/api-docs'
    .set('Accept', 'text/html') // Demander du HTML
    .expect('Content-Type', /html/) // Vérifier que c'est du HTML
    .expect(200, done); // Vérifier le code 200
})

// =====
// TEST 5 : API ANIMAL ALÉATOIRE (ROUTE '/api/random')
// =====

// Test de l'endpoint d'animal aléatoire
it('/api/random responds with json and contains animal, sound, and planet', function(done) {
  request
    .get('/api/random') // Requête GET vers '/api/random'
    .set('Accept', 'application/json') // Demander du JSON
    .expect('Content-Type', /json/) // Vérifier que c'est du JSON
    .expect(200) // Vérifier le code 200
    .end(function(err, res) { // .end() : Gérer la réponse manuellement
      if (err) return done(err); // Si erreur, arrêter le test

      // Vérifier que la réponse contient les champs requis
      if (res.body.animal && res.body.sound && res.body.planet) {
        done(); // Test réussi
      } else {
        // Test échoué : manque des champs requis
        done(new Error('Missing required fields in random animal response'));
      }
    });
})
})

```

7.2 Tester les tests

```
npm test
```

8. Configuration Docker

8.1 Créer le Dockerfile

```

#
# DOCKERFILE - CONFIGURATION DU CONTENEUR
#
# Le Dockerfile est un script qui décrit comment construire une image Docker
# Une image Docker est comme un "modèle" pour créer des conteneurs

# ÉTAPE 1 : IMAGE DE BASE
# -----
# FROM : Spécifie l'image de base à utiliser
# node:14 : Image officielle Node.js version 14
# Cette image contient déjà Node.js, npm et les outils nécessaires
FROM node:14

# ÉTAPE 2 : RÉPERTOIRE DE TRAVAIL
# -----
# WORKDIR : Définit le répertoire de travail dans le conteneur
# /usr/src/app : Chemin standard pour les applications Node.js
# Toutes les commandes suivantes s'exécuteront dans ce répertoire
WORKDIR /usr/src/app

# ÉTAPE 3 : COPIE DES FICHIERS DE DÉPENDANCES
# -----
# COPY : Copie des fichiers du système hôte vers le conteneur
# package*.json : Copie package.json ET package-lock.json
# ./ : Destination dans le répertoire de travail actuel
# On copie d'abord les dépendances pour optimiser le cache Docker
COPY package*.json ./

# ÉTAPE 4 : INSTALLATION DES DÉPENDANCES
# -----
# RUN : Exécute une commande dans le conteneur
# npm install : Installe toutes les dépendances listées dans package.json
# Cette étape est mise en cache par Docker si package.json n'a pas changé
RUN npm install

# ÉTAPE 5 : COPIE DU CODE SOURCE
# -----
# COPY . . : Copie TOUS les fichiers du projet vers le conteneur
# Premier . : Répertoire source (dossier actuel sur votre machine)
# Deuxième . : Répertoire destination (dans le conteneur)
# On copie le code après les dépendances pour optimiser le cache
COPY . .

# ÉTAPE 6 : EXPOSITION DU PORT
# -----
# EXPOSE : Documente le port utilisé par l'application
# 8080 : Port sur lequel notre application Node.js écoute
# Note : EXPOSE ne fait que documenter, il ne publie pas réellement le port
EXPOSE 8080

# ÉTAPE 7 : COMMANDE DE DÉMARRAGE
# -----
# CMD : Commande par défaut à exécuter quand le conteneur démarre
# [ "node", "app.js" ] : Tableau avec la commande et ses arguments
# node app.js : Démarre notre application Node.js
CMD [ "node", "app.js" ]

```

8.2 Tester Docker

```

docker build -t simplonwars-farm-nodejs .
docker run -p 8080:8080 simplonwars-farm-nodejs

```

9. Mise en place CI/CD

9.0 Concepts CI/CD pour débutants

Qu'est-ce que la CI/CD ?

CI (Continuous Integration) = Intégration Continue

- **Définition** : Automatiser la vérification du code à chaque modification
- **Objectif** : Détecter les problèmes rapidement
- **Exemple** : Tests automatiques à chaque commit

CD (Continuous Deployment) = Déploiement Continu

- **Définition** : Automatiser la mise en production
- **Objectif** : Livrer rapidement et en toute sécurité
- **Exemple** : Déploiement automatique après validation des tests

 Quand se déclenchent les pipelines ?

Déclencheurs automatiques :

1. **Pipeline de Tests** (.github/workflows/test.yml):
 - ☒ Sur chaque git **push** vers la branche **main**
 - ☒ Sur chaque **Pull Request** vers la branche **main**
 - ☒ Sur chaque **modification** de code
2. **Pipeline Docker** (.github/workflows/docker.yml):
 - ☒ Sur chaque git **push** vers la branche **main** uniquement
 - ☒ **PAS** sur les **Pull Requests** (sécurité)
 - ☒ Seulement quand le **code est validé**

Exemple concret :

```
# 1. Vous modifiez le code
git add .
git commit -m "feat: ajouter un nouvel animal"

# 2. Vous poussez sur GitHub
git push origin main

# 3. AUTOMATIQUEMENT :
#   → Pipeline de tests se lance
#   → Si tests OK → Pipeline Docker se lance
#   → Image Docker publiée sur Docker Hub
```

🔗 Séquence de déclenchement :

```
1. Développeur fait un commit et push
  |
2. GitHub détecte le changement
  |
3. Pipeline de tests se déclenche AUTOMATIQUEMENT
  |
4. Tests s'exécutent sur un serveur Ubuntu
  |
5. Si tests OK → Pipeline Docker se déclenche
  |
6. Image Docker construite et publiée
  |
7. Application disponible partout !
```

Pourquoi utiliser la CI/CD ?

✅ Avantages pour les développeurs :

- Détection rapide des bugs
- Confiance dans le code déployé
- Réduction du stress de mise en production
- Historique des déploiements

✅ Avantages pour l'équipe :

- Code toujours fonctionnel
- Collaboration facilitée
- Livraison plus rapide
- Moins de régressions

Workflow CI/CD typique :

```
1. Développeur fait un commit
  |
2. Pipeline CI se déclenche automatiquement
  |
3. Tests automatiques s'exécutent
  |
4. Si tests OK → Build de l'application
  |
5. Si build OK → Déploiement automatique
  |
6. Application disponible en production
```

Outils utilisés dans ce tutoriel :





- **GitHub Actions** : Plateforme CI/CD intégrée à GitHub
- **Docker** : Conteneurisation pour la portabilité
- **Mocha/Supertest** : Tests automatisés
- **npm** : Gestion des dépendances et scripts

🔗 Comment surveiller les pipelines ?

1. Interface GitHub Actions :

- Allez sur votre repository GitHub
- Cliquez sur l'onglet **"Actions"**
- Vous verrez tous vos pipelines en cours et terminés

2. Statuts des pipelines :

-  **Vert** : Pipeline réussi, tout fonctionne
-  **Rouge** : Pipeline échoué, problème à corriger
-  **Jaune** : Pipeline en cours d'exécution
-  **Gris** : Pipeline en attente

3. Détails d'un pipeline :

- Cliquez sur un pipeline pour voir les détails
- Chaque étape est listée avec son statut
- Logs détaillés pour comprendre les erreurs

4. Notifications :

- GitHub vous envoie un email en cas d'échec
- Vous pouvez configurer des notifications Slack/Disord
- Intégration possible avec d'autres outils

🔗 Déclenchement manuel (optionnel) :

Vous pouvez aussi déclencher un pipeline manuellement :

1. Allez dans l'onglet **"Actions"**
2. Cliquez sur le pipeline souhaité
3. Cliquez sur **"Run workflow"**
4. Choisissez la branche et lancez

Cas d'usage :

- Tester une branche sans la merger
- Relancer un pipeline qui a échoué pour une raison externe
- Tester des modifications de configuration

9.1 Créer .github/workflows/test.yml

```
#
# ~~~~~
# PIPELINE CI/CD - TESTS AUTOMATIQUES
# ~~~~~
# Ce fichier définit un pipeline GitHub Actions
# Un pipeline est une série d'étapes automatisées qui s'exécutent sur GitHub

# NOM DU PIPELINE
# -----
# name : Nom affiché dans l'interface GitHub Actions
name: SimplonWars Farm Node.js CI

# DÉCLENCHEURS (WHEN)
# -----
# on : Définit quand le pipeline doit s'exécuter
on:
  # Déclenchement sur push vers la branche main
  push:
    branches:
      - main
  # Déclenchement sur création/modification de Pull Request vers main
  pull_request:
    branches:
      - main

# JOBS (TÂCHES)
# -----
# jobs : Définit les tâches à exécuter
# Un job est une unité de travail qui s'exécute sur un runner (serveur)
jobs:
  # Nom du job (peut y en avoir plusieurs)
  build:
    # TYPE DE RUNNER
    # -----
    # runs-on : Type de serveur sur lequel exécuter le job
    # ubuntu-latest : Serveur Linux Ubuntu (gratuit, fourni par GitHub)
    # Autres options : windows-latest, macos-latest
    runs-on: ubuntu-latest

    # ÉTAPES DU JOB
    # -----
    # steps : Liste des étapes à exécuter dans l'ordre
    steps:

      # ÉTAPE 1 : RÉCUPÉRATION DU CODE
      # -----
      # name : Nom de l'étape (affiché dans l'interface)
      - name: Checkout repository
        # uses : Action GitHub à utiliser
        # actions/checkout@v2 : Action officielle pour récupérer le code
        # @v2 : Version de l'action (spécifique pour la stabilité)
        uses: actions/checkout@v2

      # ÉTAPE 2 : CONFIGURATION DE NODE.JS
      # -----
      - name: Use Node.js
        # actions/setup-node@v1 : Action pour installer Node.js
        uses: actions/setup-node@v1
        # with : Paramètres de l'action
        with:
          node-version: '18.x' # Version de Node.js à installer

      # ÉTAPE 3 : INSTALLATION DES DÉPENDANCES
      # -----
      - name: Install dependencies
        # run : Commande shell à exécuter
        # npm install : Installe les dépendances du projet
        run: npm install

      # ÉTAPE 4 : EXÉCUTION DES TESTS
      # -----
      - name: Run tests
        # npm test : Lance la suite de tests définie dans package.json
        # Si les tests échouent, le pipeline s'arrête (fail fast)
        run: npm test
```

9.2 Créer .github/workflows/docker.yml

```
#
# ~~~~~
# PIPELINE CI/CD - BUILD ET PUBLICATION DOCKER
# ~~~~~
# Ce pipeline construit automatiquement une image Docker
# et la publie sur Docker Hub et GitHub Container Registry

# NOM DU PIPELINE
# -----
name: Publish Docker image

# DÉCLENCHEURS
# -----
# Se déclenche uniquement sur push vers main (pas sur les Pull Requests)
# Pour éviter de publier des images non testées
on:
  push:
    branches:
      - main

# JOBS
# ----
jobs:
  # Job de construction et publication
  build-and-push:
```

```

# Runner Ubuntu (gratuit)
runs-on: ubuntu-latest

# ÉTAPES DU PIPELINE
# -----
steps:

# ÉTAPE 1 : RÉCUPÉRATION DU CODE
# -----
- name: Checkout
  # Récupère le code source depuis le repository
  uses: actions/checkout@v2

# ÉTAPE 2 : CONFIGURATION QEMU
# -----
- name: Set up QEMU
  # QEMU : Émulateur pour construire des images multi-architecture
  # Permet de créer des images pour Linux, Windows, ARM, etc.
  uses: docker/setup-qemu-action@v1

# ÉTAPE 3 : CONFIGURATION DOCKER BUILDX
# -----
- name: Set up Docker Buildx
  # Buildx : Extension Docker pour construire des images avancées
  # Permet la construction multi-architecture et le cache distribué
  uses: docker/setup-buildx-action@v1

# ÉTAPE 4 : CONSTRUCTION ET PUBLICATION
# -----
- name: Build and push
  # Action officielle Docker pour construire et publier
  uses: docker/build-push-action@v2
  with:
    context: .          # Répertoire de contexte (dossier actuel)
    file: ./Dockerfile  # Chemin vers le Dockerfile
    push: false         # Ne pas publier pour l'instant (sécurité)
    tags: simplonwars-farm-nodejs:latest # Étiquette de l'image

# =====
# CONFIGURATION AVANCÉE (OPTIONNELLE)
# =====
# Pour publier réellement l'image, il faut :
# 1. Créer un compte Docker Hub
# 2. Ajouter les secrets GitHub :
#    - DOCKER_USERNAME : Votre nom d'utilisateur Docker Hub
#    - DOCKER_PASSWORD : Votre mot de passe Docker Hub
# 3. Modifier le pipeline :
#    push: true
#    tags: |
#      votre-username/simplonwars-farm-nodejs:latest
#      ghcr.io/votre-username/simplonwars-farm-nodejs:latest
---

## 10. Déploiement GitHub Pages

### 10.1 Qu'est-ce que GitHub Pages ?

**GitHub Pages** est un service gratuit qui permet d'héberger des sites web statiques directement depuis votre repository GitHub. C'est parfait pour :
- Présenter votre projet
- Créer une documentation en ligne
- Avoir un site web professionnel
- Montrer vos compétences

### 10.2 Configuration du pipeline de déploiement

#### **Fichier de workflow :*.github/workflows/deploy.yml`

```yaml
=====
GITHUB ACTIONS - DÉPLOIEMENT GITHUB PAGES
=====

name: Deploy to GitHub Pages

Déclencheurs
on:
 push:
 branches: [main]
 workflow_run:
 workflows: ["Tests and Build"]
 types: [completed]

jobs:
 deploy:
 runs-on: ubuntu-latest

 # Permissions nécessaires pour GitHub Pages
 permissions:
 contents: write
 pages: write
 id-token: write

 steps:
 - name: Checkout
 uses: actions/checkout@v4

 - name: Setup Node.js
 uses: actions/setup-node@v4
 with:
 node-version: '18'
 cache: 'npm'

 - name: Install dependencies

```

```
run: npm ci

- name: Run tests
 run: npm test

- name: Create static site
 run: |
 mkdir -p docs
 # Création du site statique...

- name: Deploy to GitHub Pages
 uses: peaceiris/actions-gh-pages@v3
 with:
 github_token: ${ secrets.GITHUB_TOKEN }
 publish_dir: ./docs
 force_orphan: true
```

### 10.3 Création du site statique

Le pipeline crée automatiquement un site web avec :

#### Page d'accueil moderne

- Design responsive avec thème Star Wars
- Présentation de l'API et de ses endpoints
- Interface utilisateur intuitive
- Couleurs et animations cohérentes

#### Contenu généré

- Documentation complète de l'API
- Liste de tous les endpoints disponibles
- Exemples d'utilisation
- Liens vers le code source et la documentation Swagger

### 10.4 Activation de GitHub Pages

#### Étapes de configuration :

1. **Aller dans les paramètres du repository :**
  - GitHub > Votre repository > Settings
2. **Configurer GitHub Pages :**
  - Section "Pages" dans le menu de gauche
  - Source : "Deploy from a branch"
  - Branch : "gh-pages"
  - Folder : "/"(root)"
3. **Vérifier l'activation :**
  - Le site sera accessible après le premier déploiement
  - URL : `https://votre-username.github.io/votre-repo`

### 10.5 Surveillance du déploiement

#### Vérifier le statut :

1. **Onglet Actions :** Voir les pipelines en cours
2. **Onglet Settings > Pages :** Voir l'état du déploiement
3. **Branche gh-pages :** Voir les fichiers déployés

#### Logs de déploiement :

```
Voir les actions récentes
gh run list

Voir les détails d'une action
gh run view [ID]
```

### 10.6 Personnalisation du site

#### Modifier le design :

Le site est généré dans l'étape "Create static site" du pipeline. Vous pouvez :

1. **Changer les couleurs :** Modifier le CSS dans le workflow
2. **Ajouter du contenu :** Modifier le HTML généré
3. **Changer la structure :** Réorganiser les sections

#### Ajouter des fonctionnalités :

- Formulaire de contact
- Galerie d'images
- Blog intégré
- Statistiques d'utilisation

## 11. Lancement et test

### 11.1 Lancer l'application

```
npm start
```

### 11.2 Tester toutes les routes

- **Page d'accueil :** `http://localhost:8080/`
- **Intro Star Wars :** `http://localhost:8080/starwars`
- **Documentation API :** `http://localhost:8080/api-docs`
- **API animaux :** `http://localhost:8080/api`
- **Animal aléatoire :** `http://localhost:8080/api/random`
- **Plusieurs animaux :** `http://localhost:8080/api/random/5`
- **Animaux par planète :** `http://localhost:8080/api/planet/Tatooine`
- **Statistiques :** `http://localhost:8080/api/stats`



- **Recherche** : `http://localhost:8080/api/search/ban`

### 11.3 Lancer les tests

```
npm test
```

## 12. Résolution des problèmes

### 12.1 Problèmes de tests

#### Erreur "port already in use"

```
Identifier le processus qui utilise le port
netstat -ano | findstr :8080 # Windows
lsof -i :8080 # Linux/Mac

Tuer le processus
taskkill /PID [PID] /F # Windows
kill -9 [PID] # Linux/Mac
```

#### Tests qui échouent

- **Problème de routage** : Vérifier que les routes sont correctement définies
- **Problème de contenu** : Les tests acceptent maintenant l'animation Star Wars ou le texte George Orwell
- **Problème Swagger** : Le test accepte les codes 200 et 301

### 12.2 Problèmes Docker

#### Erreur de permissions

```
Windows : Exécuter en tant qu'administrateur
Linux/Mac : Ajouter l'utilisateur au groupe docker
sudo usermod -aG docker $USER
```

#### Erreur de build

```
Nettoyer le cache Docker
docker system prune -a

Reconstruire l'image
docker build --no-cache -t simplonwars-farm-nodejs .
```

### 12.3 Problèmes CI/CD

#### Erreur 403 sur GitHub Pages

- Vérifier que les permissions sont configurées dans le workflow
- S'assurer que GitHub Pages est activé dans les paramètres du repository

#### Erreur de connexion Docker Hub

- Les connexions aux registres sont désactivées par défaut
- Pour activer : configurer les secrets GitHub et décommenter les étapes

### 12.4 Problèmes de déploiement

#### Site GitHub Pages ne s'affiche pas

1. Vérifier que la branche `gh-pages` a été créée
2. Vérifier les paramètres GitHub Pages
3. Attendre quelques minutes pour la propagation

#### Erreur de permissions GitHub Actions

```
Ajouter dans le workflow
permissions:
 contents: write
 pages: write
 id-token: write
```

## Félicitations !

Vous avez créé une application Node.js complète avec :

- ☒ API REST avec 8 endpoints
- ☒ Documentation Swagger interactive
- ☒ Tests automatisés robustes
- ☒ Configuration Docker
- ☒ Pipeline CI/CD complet
- ☒ Déploiement GitHub Pages
- ☒ Architecture propre et maintenable

## Comment vérifier que tout fonctionne

### 1. Vérification locale

```
Lancer l'application
npm start

Tester les endpoints
curl http://localhost:8080/api
curl http://localhost:8080/api/random
curl http://localhost:8080/api/stats

Lancer les tests
npm test
```

### 2. Vérification Docker

```
Construire l'image
docker build -t simplonwars-farm-nodejs .

Lancer le conteneur
docker run -p 8080:8080 simplonwars-farm-nodejs

Tester depuis un autre terminal
curl http://localhost:8080/api
```

### 3. Vérification CI/CD




#### 1. Poussez votre code sur GitHub :

```
git add .
git commit -m "feat: initial commit with CI/CD"
git push origin main
```

#### 2. Vérifiez les Actions GitHub :

- Allez sur votre repository GitHub
- Cliquez sur l'onglet "Actions"
- Vous devriez voir vos pipelines en cours d'exécution

#### 3. Comprendre les résultats :

-  **Vert** : Tout fonctionne correctement
-  **Rouge** : Il y a un problème à corriger
-  **Jaune** : Pipeline en cours d'exécution

### 4. Vérification GitHub Pages

#### 1. Vérifier le déploiement :

- Settings > Pages > Voir l'URL du site
- Le site devrait être accessible en ligne

#### 2. Tester le site :

- Vérifier que toutes les sections s'affichent
- Tester les liens vers l'API
- Vérifier le design responsive



## Prochaines étapes

### Pour approfondir vos connaissances :

#### 1. Ajouter des tests plus avancés :

- Tests de performance
- Tests d'intégration avec une vraie base de données
- Tests de sécurité

#### 2. Améliorer la CI/CD :

- Ajouter des tests de qualité de code (ESLint, SonarQube)
- Déploiement automatique sur un serveur de staging
- Notifications Slack/Email en cas d'échec

#### 3. Optimiser Docker :

- Images multi-stage pour réduire la taille
- Sécurité des images (scan de vulnérabilités)
- Orchestration avec Docker Compose

#### 4. Monitoring et observabilité :

- Logs structurés
- Métriques de performance
- Alertes automatiques

#### 5. Améliorer GitHub Pages :

- Ajouter un blog intégré
- Statistiques d'utilisation
- Formulaire de contact
- Galerie de projets

### Concepts à explorer :

- **Microservices** : Diviser l'application en services indépendants
- **Kubernetes** : Orchestration de conteneurs à grande échelle
- **Infrastructure as Code** : Terraform, CloudFormation
- **GitOps** : Gestion de l'infrastructure via Git



## Ressources supplémentaires

### Documentation officielle :

- [Documentation Express.js \(https://expressjs.com/\)](https://expressjs.com/)
- [Documentation Swagger \(https://swagger.io/\)](https://swagger.io/)
- [Documentation Docker \(https://docs.docker.com/\)](https://docs.docker.com/)
- [Documentation GitHub Actions \(https://docs.github.com/en/actions\)](https://docs.github.com/en/actions)
- [Documentation GitHub Pages \(https://docs.github.com/en/pages\)](https://docs.github.com/en/pages)

### Tutoriels recommandés :

- [Node.js Best Practices \(https://github.com/goldbergonyi/nodebestpractices\)](https://github.com/goldbergonyi/nodebestpractices)
- [Docker Tutorial \(https://docs.docker.com/get-started/\)](https://docs.docker.com/get-started/)
- [GitHub Actions Tutorial \(https://docs.github.com/en/actions/learn-github-actions\)](https://docs.github.com/en/actions/learn-github-actions)
- [GitHub Pages Tutorial \(https://docs.github.com/en/pages/getting-started-with-github-pages\)](https://docs.github.com/en/pages/getting-started-with-github-pages)

### Outils utiles :

- [Postman \(https://www.postman.com/\)](https://www.postman.com/) : Tester les APIs
- [Docker Desktop \(https://www.docker.com/products/docker-desktop/\)](https://www.docker.com/products/docker-desktop/) : Interface Docker
- [VS Code \(https://code.visualstudio.com/\)](https://code.visualstudio.com/) : Éditeur de code

---

## Support et communauté


### En cas de problème :

1. Vérifiez les logs d'erreur
2. Consultez la documentation officielle
3. Recherchez sur Stack Overflow
4. Posez des questions sur les forums de la communauté

### Communautés recommandées :

- [Node.js Community \(https://nodejs.org/en/community/\)](https://nodejs.org/en/community/)
- [Docker Community \(https://www.docker.com/community/\)](https://www.docker.com/community/)
- [GitHub Community \(https://github.com/orgs/community/discussions\)](https://github.com/orgs/community/discussions)

---

 Vous êtes maintenant prêt à créer vos propres applications avec CI/CD !