

Département Réseau et informatique  
Filière Licence : Ingénierie des applications web et mobiles

# Développement Fullstack + CI/CD avec GitHub Actions

Contrôle Continu N°2 (Pratique)

Réalisé Par :  
Elhamdouchi Maryem

Année universitaire : 2024/2025

## 1. Introduction :

Dans le cadre du cours **Déploiement des Applications Web et Mobiles**, ce projet vise à développer une application **Fullstack** moderne en utilisant les technologies **React.js** pour le frontend et **Express.js** pour le backend, couplée à une base de données **SQLite** pour le stockage des données. L'ensemble est conteneurisé avec **Docker** pour garantir une exécution cohérente dans différents environnements, et automatisé via un pipeline **CI/CD** avec **GitHub Actions** pour assurer une intégration et un déploiement continus.

L'objectif principal est de mettre en pratique les concepts clés du développement web et des méthodologies **DevOps**, en mettant l'accent sur :

- La **création d'une API REST** (CRUD) avec Express.js.
- La **gestion des données** via SQLite, une solution légère et sans serveur.
- La conception d'une interface utilisateur réactive avec React.js.
- La **conteneurisation** avec Docker pour une portabilité optimale.
- L'automatisation des tests et du déploiement avec **GitHub Actions**.

Ce rapport détaille les étapes de réalisation, les défis rencontrés et les solutions apportées, tout en illustrant l'importance des bonnes pratiques en **développement logiciel** et en **automatisation des processus**.

## 2. Développement Fullstack :

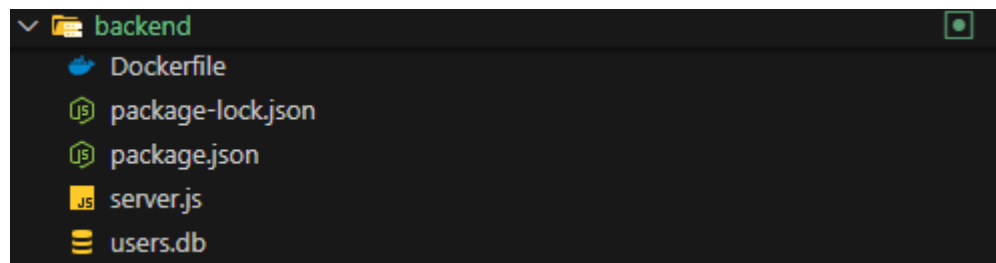
### 2.1 Backend – Express.js :

#### Architecture : Structure des dossiers et choix techniques

Le backend est structuré autour d'un serveur Express.js avec une base de données SQLite (initialement prévue pour MySQL, mais adaptée ici pour la simplicité).

#### Structure des dossiers :

```
backend/  
├── Dockerfile          # Configuration Docker pour le backend  
├── package.json        # Dépendances Node.js  
├── server.js           # Point d'entrée (routes, middleware, DB)  
└── users.db            # Base de données SQLite (persistante via Docker)
```



#### Choix techniques :

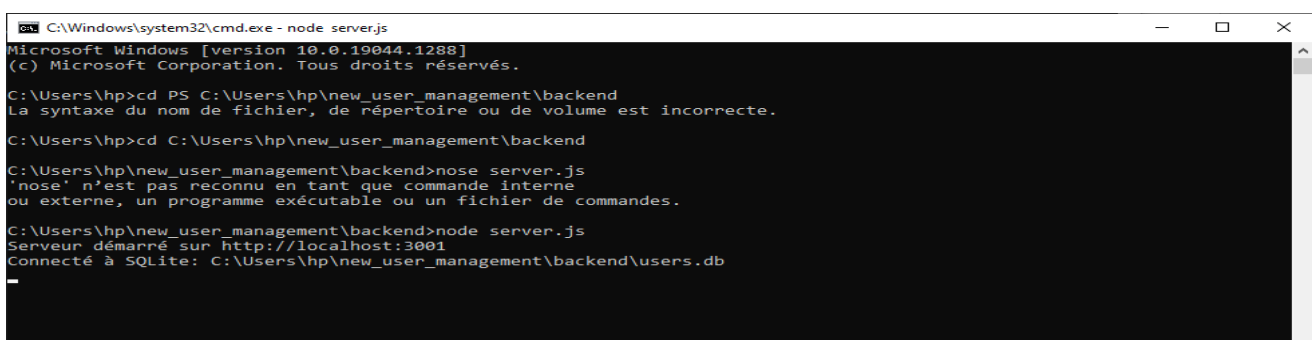
- Express.js : Framework Node.js pour la création d'API RESTful.
- SQLite : Base de données légère, utilisée ici pour simplifier le déploiement (remplaçable par MySQL).
- CORS sécurisé : Restriction des requêtes à http://localhost:3000 (frontend).
- Middleware body-parser : Pour traiter les données JSON des requêtes.

#### Endpoints implémentés :

- GET /api/users → Afficher tous les utilisateurs
- ○ POST /api/users → Ajouter un nouvel utilisateur
- ○ PUT /api/users/:id → Modifier un utilisateur existant
- ○ DELETE /api/users/:id → Supprimer un utilisateur

#### L'exécution de backend :

L'exécution via la commande : **node server.js**



```
C:\Windows\system32\cmd.exe - node server.js  
Microsoft Windows [version 10.0.19044.1288]  
(c) Microsoft Corporation. Tous droits réservés.  
  
C:\Users\hnp>cd PS C:\Users\hnp\new_user_management\backend  
La syntaxe du nom de fichier, de répertoire ou de volume est incorrecte.  
  
C:\Users\hnp>cd C:\Users\hnp\new_user_management\backend  
  
C:\Users\hnp\new_user_management\backend>nose server.js  
'nose' n'est pas reconnu en tant que commande interne  
ou externe, un programme exécutable ou un fichier de commandes.  
  
C:\Users\hnp\new_user_management\backend>node server.js  
Serveur démarré sur http://localhost:3001  
Connecté à SQLite: C:\Users\hnp\new_user_management\backend\users.db  
-
```

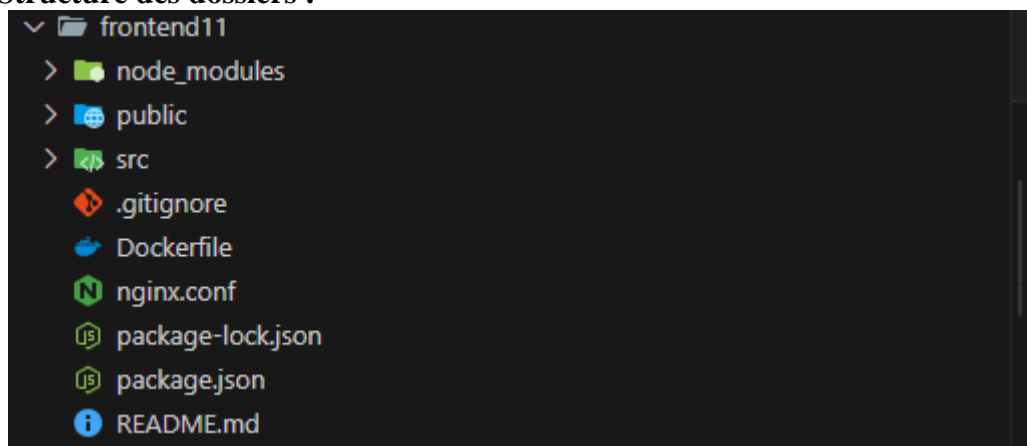
```
localhost:3001/api/users
impression automatique
{
  "data": [
    {
      "id": 3,
      "name": "assia",
      "email": "assiamari@gmail.com",
      "age": 20
    },
    {
      "id": 4,
      "name": "malak",
      "email": "malakbouyi@gmail.com",
      "age": 20
    },
    {
      "id": 5,
      "name": "maryem",
      "email": "maryemelhamdouchi12@gmail.com",
      "age": 20
    },
    {
      "id": 6,
      "name": "wissal",
      "email": "wissalbenmamoune@gmail.com",
      "age": 20
    },
    {
      "id": 7,
      "name": "hind",
      "email": "hindomari@gmail.com",
      "age": 21
    },
    {
      "id": 8,
      "name": "mariam",
      "email": "mariamfighri@gmail.com",
      "age": 22
    }
  ],
  "pagination": {
    "page": 1,
  }
}
```

## Frontend – React.js :

### Structure de l'Interface :

- L'interface React est conçue pour gérer les utilisateurs avec les fonctionnalités suivantes :
- Affichage des utilisateurs : Liste dynamique avec nom, email et âge.
- Ajout/Modification : Formulaire unique réutilisable pour les deux actions.
- Suppression : Bouton dédié avec confirmation implicite (alert).

### Structure des dossiers :



### Choix techniques :

- React : Hooks (useState, useEffect) pour une gestion moderne de l'état.
- Axios : Requêtes HTTP simplifiées vers l'API Express (meilleure gestion d'erreurs que fetch).
- Bootstrap : Classes utilitaires pour les boutons (btn-primary, btn-danger).
- CSS Intégré : Styles inline pour une personnalisation fine (ex: Flexbox pour la liste des utilisateurs).

### L'exécution de frontend :

L'execution via la commande : **npm start**

```
Windows PowerShell
Compiled successfully!

You can now view frontend11 in the browser.

Local:      http://localhost:3000
On Your Network:  http://172.30.32.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```



## 3. Dockerisation :

### 3.1 Configuration des Dockerfiles :

**Backend (backend/Dockerfile) :**

**Pourquoi utiliser un Dockerfile ?**

Le Dockerfile du backend permet de :

- Standardiser l'environnement : Utilisation d'une image Node.js 14 pour garantir la compatibilité.
- Sécuriser l'application : Exécution avec un utilisateur non-root (USER node) pour limiter les risques.
- Persister les données : Configuration explicite du volume pour la base de données SQLite (/data/users.db).
- Optimiser les builds : Copie séparée des package.json pour exploiter le cache Docker.

**Outils clés utilisés :**

- node:14-bullseye : Image officielle Node.js avec Debian stable.
- sqlite3 : Base de données légère intégrée.
- chown : Pour définir les permissions du fichier de base de données.

### Capture d'écran :

```
backend > Dockerfile > ...
1 # backend/Dockerfile
2 # Utiliser une version spécifique de Node.js pour plus de stabilité
3 FROM node:14-bullseye
4
5 # Définir le répertoire de travail
6 WORKDIR /app
7
8 # Installer les dépendances système nécessaires
9 RUN apt-get update && apt-get install -y \
10     python3 \
11     make \
12     g++ \
13     sqlite3 \
14     && rm -rf /var/lib/apt/lists/*
15
16 # Copier uniquement les fichiers package.json pour optimiser le cache Docker
17 COPY package.json package-lock.json ./
18
19 # Supprimer node_modules s'il existe (au cas où il serait copié depuis l'hôte)
20 RUN rm -rf node_modules
21
22 # Installer les dépendances Node.js, y compris sqlite3
23 RUN npm install --unsafe-perm
24
25 # Copier le reste de l'application
26 COPY . .
27
28 # Configurer la base de données avec les bonnes permissions
29 RUN mkdir -p /data && \
30     touch /data/users.db && \
31     chown -R node:node /data
32
```

### Frontend (frontend11/Dockerfile)

#### Pourquoi utiliser un Dockerfile ?

Le Dockerfile du frontend permet de :

- Optimiser la taille finale : Utilisation d'un multi-stage build (Node.js pour le build → nginx pour la production).
- Servir des fichiers statiques efficacement : Serveur NGINX configuré pour le routage (ex: redirection /api).
- Isoler les dépendances : Aucune installation inutile dans l'image finale (grâce à l'étape builder).

#### Outils clés utilisés :

- node:16-alpine : Image Node.js légère pour la phase de build.
- nginx:alpine : Serveur web minimaliste pour la production.
- nginx.conf : Configuration personnalisée pour le proxy vers le backend.

### Capture d'écran :

```
Dockerfile frontend11 2 x Dockerfile backend 1
frontend11 > Dockerfile > ...
1 # frontend11/Dockerfile
2 FROM node:16-alpine as builder
3
4 WORKDIR /app
5 COPY package*.json ./
6 RUN npm install
7 COPY . .
8 RUN npm run build
9
10 FROM nginx:alpine
11 COPY --from=builder /app/build /usr/share/nginx/html
12 COPY nginx.conf /etc/nginx/conf.d/default.conf
13 EXPOSE 80
14 CMD ["nginx", "-g", "daemon off;"]
```

## Explication de Docker Compose : Pourquoi utiliser Docker Compose ? Docker Compose permet de :

- Orchestrer plusieurs conteneurs (backend + frontend + DB) via un seul fichier YAML
- Définir les dépendances entre services (ex: frontend dépend de backend)
- Configurer facilement les réseaux, volumes et variables d'environnement
- Standardiser les environnements entre développement et production.

## Structure du docker-compose.yml :

```
docker-compose.yml > ...
1  version: '3.3'
2
3  services:
4    backend:
5      build: ./backend
6      ports:
7        - "3001:3001"
8      volumes:
9        - ./backend/users.db:/app/users.db
10     environment:
11       - NODE_ENV=production
12       - DB_PATH=/data/users.db
13
14    frontend:
15      build: ./frontend11
16      ports:
17        - "3000:80"
18      depends_on:
19        - backend
20
```

## Commandes de base :

1 : docker-compose build :

```
HPDESKTOP-34SV400 MINGW64 ~/new_user_management (main)
$ docker-compose build
Building backend
Step 1/14 : FROM node:14-bullseye
--> cc0450a76a60
Step 2/14 : WORKDIR /app
--> Using cache
--> 7cc37fa24b5b
Step 3/14 : RUN apt-get update && apt-get install -y python3 make g++ sqlite3 && rm -rf /var/lib/apt/lists/*
--> Using cache
--> 6e4a9b8fbc0a
Step 4/14 : COPY package.json package-lock.json ./
--> Using cache
--> 0747c8a4981d
Step 5/14 : RUN rm -rf node_modules
--> Using cache
--> ff410f3567f8
Step 6/14 : RUN npm install --unsafe-perm
--> Using cache
--> 8a2a454b7f4a
Step 7/14 : COPY . .
--> 972b18d7a634
Step 8/14 : RUN mkdir -p /data && touch /data/users.db && chown -R node:node /data
--> Running in 615e33e6ed94
Removing intermediate container 615e33e6ed94
```

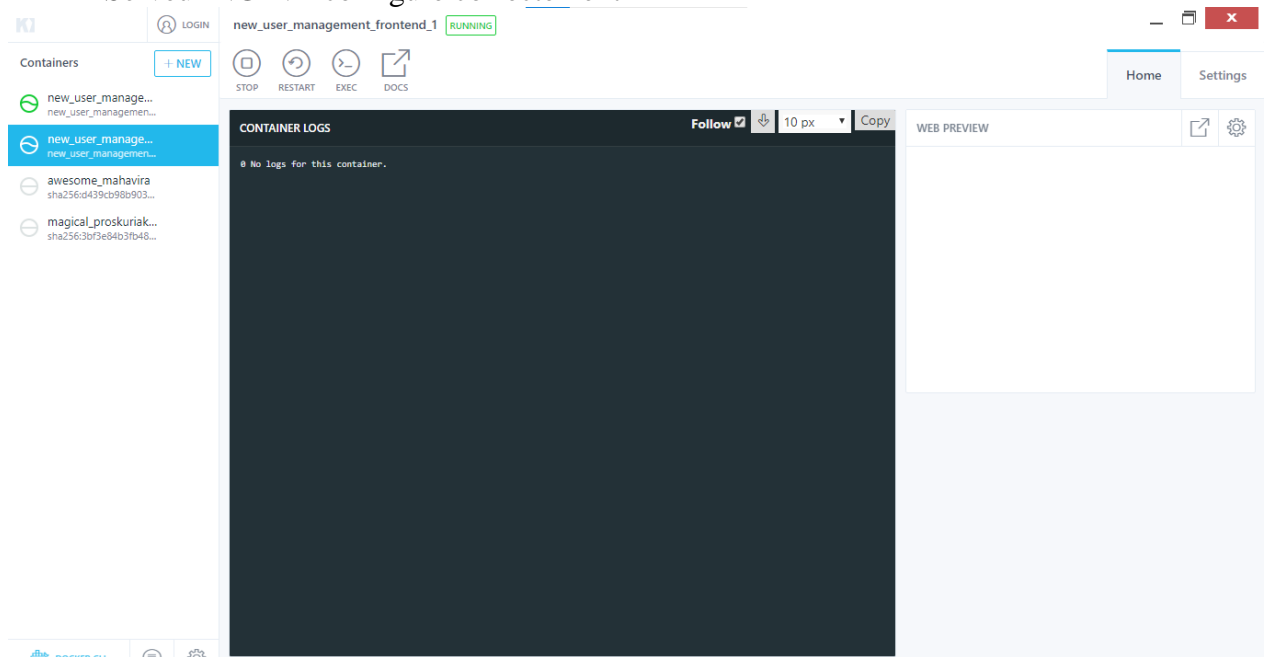
## 2 : docker-compose up :

```
hp@DESKTOP-345V4Q0 MINGW64 ~/new_user_management (main)
$ docker-compose up
Creating network "new_user_management_default" with the default driver
Creating new_user_management_backend_1 ... done
Creating new_user_management_frontend_1 ... done
Attaching to new_user_management_backend_1, new_user_management_frontend_1
backend_1 | Serveur démarré sur http://localhost:3001
frontend_1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
frontend_1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
frontend_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
frontend_1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
frontend_1 | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
frontend_1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
frontend_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
frontend_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
frontend_1 | /docker-entrypoint.sh: Configuration complete; ready for start up
frontend_1 | 2025/04/10 10:34:16 [notice] 1#1: using the "epoll" event method
frontend_1 | 2025/04/10 10:34:16 [notice] 1#1: nginx/1.27.4
frontend_1 | 2025/04/10 10:34:16 [notice] 1#1: built by gcc 14.2.0 (Alpine 14.2.0)
frontend_1 | 2025/04/10 10:34:16 [notice] 1#1: OS: Linux 4.19.130-boot2docker
frontend_1 | 2025/04/10 10:34:16 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
frontend_1 | 2025/04/10 10:34:16 [notice] 1#1: start worker processes
frontend_1 | 2025/04/10 10:34:16 [notice] 1#1: start worker process 28
```

## Résultat de la Dockerisation :

### Frontend fonctionnel :

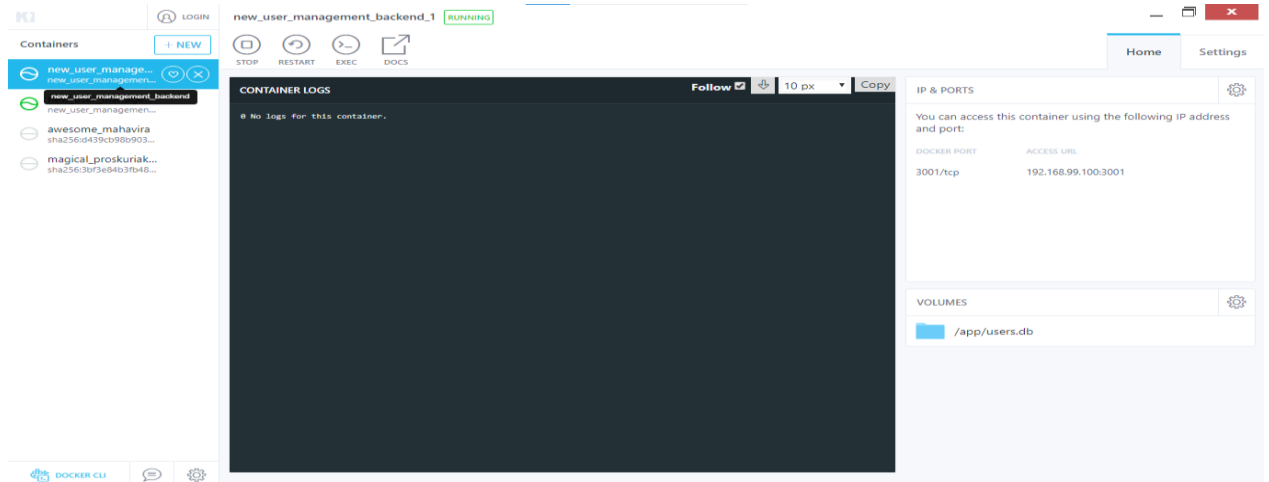
- Accessible sur <http://localhost:3000>
- Serveur NGINX configuré correctement



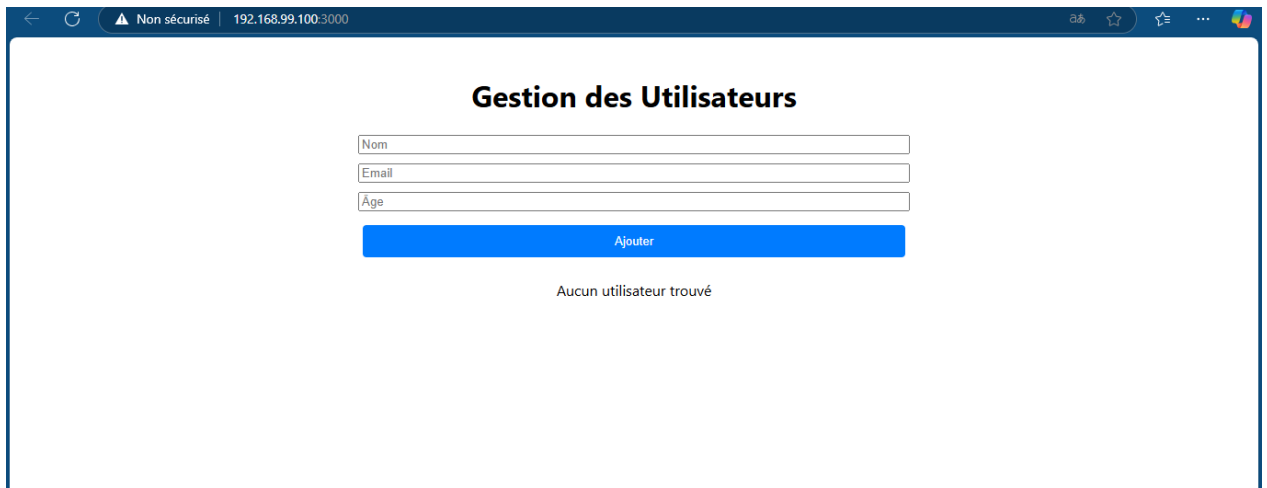
### Problème avec le backend :

- Le serveur semble démarrer mais l'API n'est pas accessible.
- Malgré l'application de plusieurs solutions recommandées (configuration de msvs\_version, réinstallation de Visual Studio avec les outils nécessaires, suppression des dépendances, reconstruction des conteneurs Docker, etc.), l'erreur liée à la compilation du module sqlite3 persiste. Ces blocages techniques nous ont empêchés de finaliser complètement le fonctionnement du backend, malgré nos efforts méthodiques pour résoudre le problème.





Affichage de frontend vers : [React App\(http://192.168.99.100:3000/\)](http://192.168.99.100:3000/)



Affichage de backend vers : <http://192.168.99.100:3001/>



## 4. Tests :

Dans cette partie, nous avons mis en place une stratégie de test pour garantir la fiabilité du backend. Nous avons utilisé **Mocha** comme framework de test, associé à **Chai** pour les assertions, ainsi que **Sinon** pour simuler certains comportements. Deux types de tests ont été implémentés : les **tests unitaires** et les **tests d'intégration**.

### Implémentation d'un test unitaire :

- Nous avons implémenté un test unitaire dans `test/unit/userController.test.js`, qui teste la logique de la fonction `getUserById`. Cette fonction appartient au contrôleur utilisateur et interagit directement avec la base de données.

```
it('getUserById devrait retourner un utilisateur existant', async () => {  
  const mockUser = { id: 1, name: 'Test User' };  
  sinon.stub(db, 'query').resolves([mockUser]); // Mock de la base de données  
  sinon.stub(db, 'query').resolves([mockUser]); // Stub de la base de données  
  
  const user = await getUserById(1);  
  expect(user).to.deep.equal(mockUser);  
});
```

Nous avons également testé le comportement en cas d'absence d'utilisateur, en vérifiant que l'erreur « Utilisateur non trouvé » est bien levée.

### Ajout de tests d'intégration :

```
describe('API Routes - Integration Tests', () => {  
  it('GET /api/users devrait retourner 200', (done) => {  
    chai.request(app)  
      .get('/api/users')  
      .end((err, res) => {  
        expect(res).to.have.status(200);  
        expect(res.body).to.be.an('array');  
        done();  
      });  
  });  
  
  it('POST /api/users devrait créer un nouvel utilisateur', async () => {  
    const res = await chai.request(app)  
      .post('/api/users')  
      .send({ name: 'Test', email: 'test@example.com', age: 25 });  
  
    expect(res).to.have.status(201);  
    expect(res.body).to.have.property('id');  
  });  
});
```

### Configuration de la couverture de code :

- Nous avons installé **nyc** (Istanbul) pour mesurer la couverture des tests avec la commande suivante :

```
C:\Users\hp\new_user_management\backend>npm install --save-dev nyc
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 130 packages, and audited 511 packages in 17s

71 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- Et ajouté ce script dans `package.json` :

```
"scripts": {
  "test": "mocha test/**/*.test.js --exit --timeout 10000",
  "test:unit": "mocha test/unit/*.test.js",
  "test:integration": "mocha test/integration/*.test.js"
},
```

### Lancement des tests localement :

- Les tests sont lancés en exécutant la commande suivante à la racine du projet :

```
C:\Users\hp\new_user_management\backend>npm test

> backend@1.0.0 test
> mocha test/**/*.test.js --exit --timeout 10000

Exception during run: Error: Cannot find module 'chai-as-promised'
Require stack:
- C:\Users\hp\new_user_management\backend\test\unit\UserController.test.js
    at Function._resolveFilename (node:internal/modules/cjs/loader:1244:15)
    at Function._load (node:internal/modules/cjs/loader:1070:27)
    at TracingChannel.traceSync (node:diagnostics_channel:322:14)
    at wrapModuleLoad (node:internal/modules/cjs/loader:217:24)
    at Module.require (node:internal/modules/cjs/loader:1335:12)
    at require (node:internal/modules/helpers:136:16)
    at Object.<anonymous> (C:\Users\hp\new_user_management\backend\test\unit\UserController.test.js:3:24)
    at Module._compile (node:internal/modules/cjs/loader:1562:14)
    at Object.<js> (node:internal/modules/cjs/loader:1699:10)
    at Module.load (node:internal/modules/cjs/loader:1313:32)
    at Function.load (node:internal/modules/cjs/loader:1123:12)
    at TracingChannel.traceSync (node:diagnostics_channel:322:14)
    at wrapModuleLoad (node:internal/modules/cjs/loader:217:24)
    at cjsLoader (node:internal/modules/esm/translators:263:5)
    at ModuleWrap.<anonymous> (node:internal/modules/esm/translators:196:7)
    at ModuleJob.run (node:internal/modules/esm/module_job:271:25)
    at async onImport.tracePromise.__proto__ (node:internal/modules/esm/loader:547:26)
    at async formattedImport (C:\Users\hp\new_user_management\backend\node_modules\mocha\lib\nodejs\esm-utils.js:9:14)
    at async exports.requireOrImport (C:\Users\hp\new_user_management\backend\node_modules\mocha\lib\nodejs\esm-utils.js:42:28)
    at async exports.loadFilesAsync (C:\Users\hp\new_user_management\backend\node_modules\mocha\lib\nodejs\esm-utils.js:100:20)
    at async singleRun (C:\Users\hp\new_user_management\backend\node_modules\mocha\lib\cli\run-helpers.js:162:3)
    at async exports.handler (C:\Users\hp\new_user_management\backend\node_modules\mocha\lib\cli\run.js:375:5) {
  code: 'MODULE_NOT_FOUND',
  requireStack: [
    'C:\\Users\\hp\\new_user_management\\backend\\test\\unit\\UserController.test.js'
  ]
}
```

### Problème rencontré : test d'intégration échoué malgré les efforts :

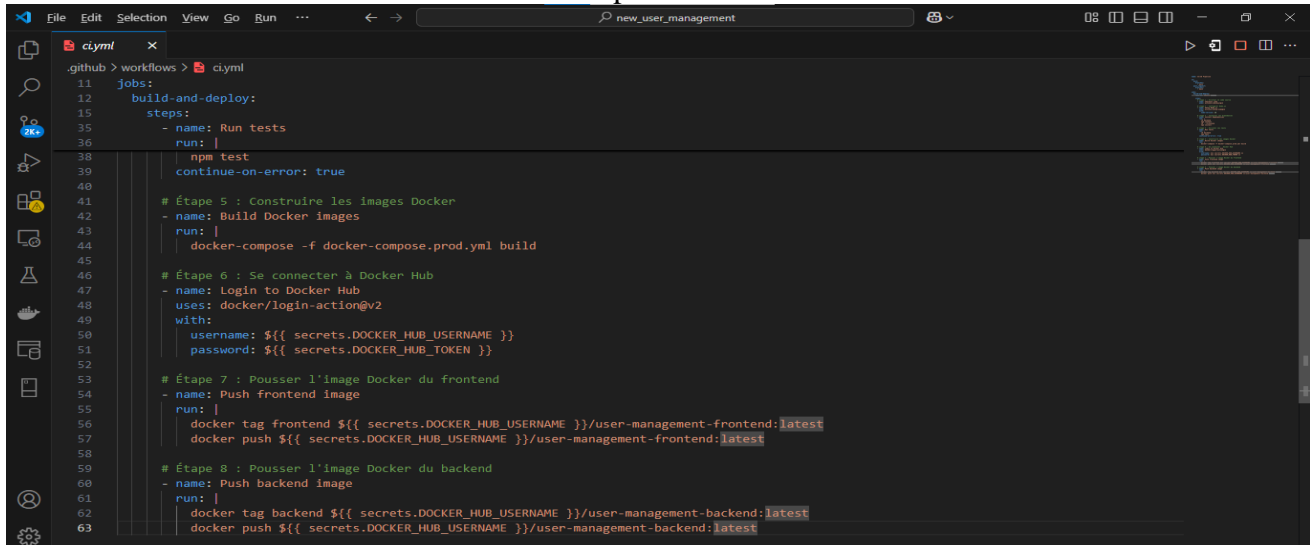
- Lors de l'exécution du test d'intégration, une erreur s'est produite sur la route GET /api/users, même si celle-ci fonctionne bien en utilisation normale via Postman.
- Hypothèses explorées :
- Le fichier `server.js` exportait l'appel `app.listen()` au lieu de l'instance `app`.** Cela empêche `chai.request(app)` de fonctionner correctement, car l'application n'est pas accessible dans le contexte de test.
- ✓ Solution tentée : modifier `server.js` pour qu'il exporte uniquement `app`, et utiliser `app.listen()` dans un fichier séparé comme `index.js`.

- **Problème de base de données verrouillée** : la base SQLite (users.db) pouvait être utilisée par un processus en arrière-plan, ce qui empêche les tests de lire ou écrire.
- ✓ Solution tentée : utilisation d'une base temporaire en mémoire (:memory:) pour les tests.
- **Conflits de port ou instance de serveur déjà lancée.**
- ✓ Solution tentée : s'assurer qu'aucune instance du serveur n'est démarrée dans les tests et que seule app est utilisée par chai-http.

## 5. GitHub Actions – CI/CD :

### Création du fichier .github/workflows/ci.yml :

- Le fichier doit contenir les étapes suivantes :

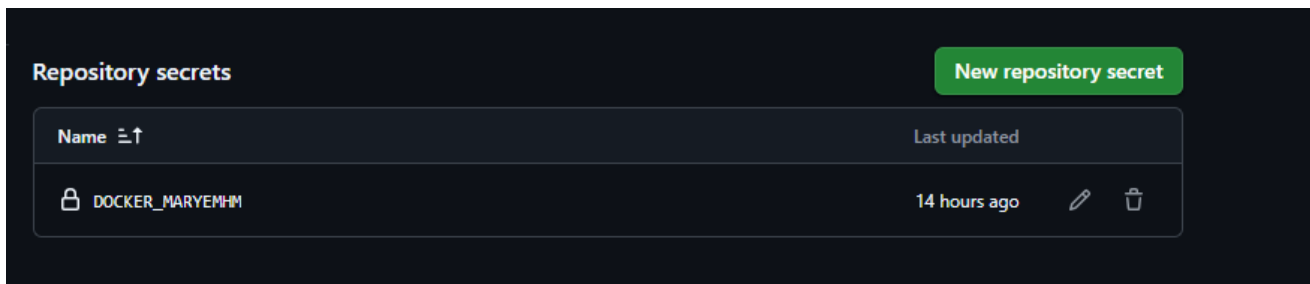


```

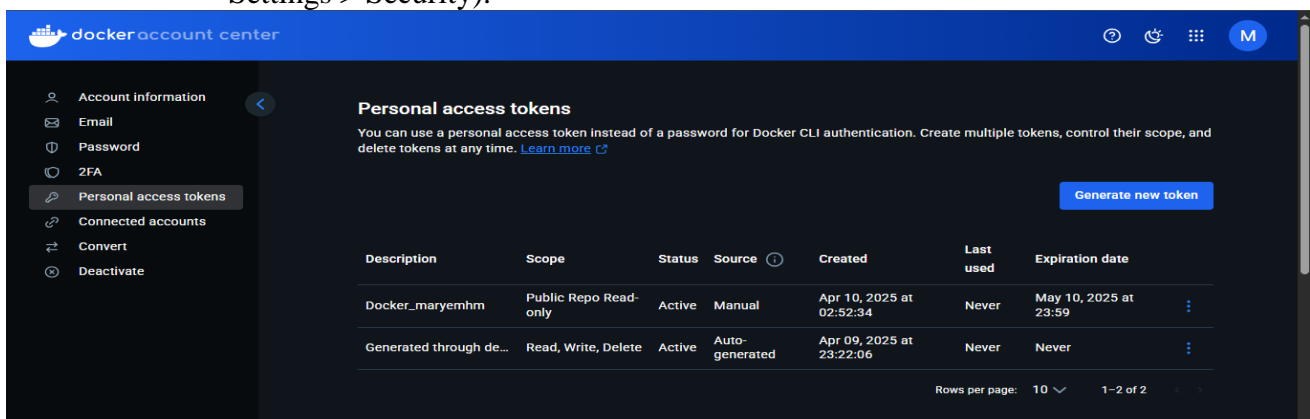
11 jobs:
12   build-and-deploy:
13     steps:
14       - name: Run tests
15         run: |
16           npm test
17           continue-on-error: true
18
19 # Étape 5 : Construire les images Docker
20 - name: Build Docker images
21   run: |
22     docker-compose -f docker-compose.prod.yml build
23
24 # Étape 6 : Se connecter à Docker Hub
25 - name: Login to Docker Hub
26   uses: docker/login-action@v2
27   with:
28     username: ${{ secrets.DOCKER_HUB_USERNAME }}
29     password: ${{ secrets.DOCKER_HUB_TOKEN }}
30
31 # Étape 7 : Pousser l'image Docker du frontend
32 - name: Push frontend image
33   run: |
34     docker tag frontend ${{ secrets.DOCKER_HUB_USERNAME }}/user-management-frontend:latest
35     docker push ${{ secrets.DOCKER_HUB_USERNAME }}/user-management-frontend:latest
36
37 # Étape 8 : Pousser l'image Docker du backend
38 - name: Push backend image
39   run: |
40     docker tag backend ${{ secrets.DOCKER_HUB_USERNAME }}/user-management-backend:latest
41     docker push ${{ secrets.DOCKER_HUB_USERNAME }}/user-management-backend:latest
  
```

### Configuration des secrets GitHub :

- Secrets requis (à ajouter dans Settings > Secrets > Actions) :



- DOCKER\_HUB\_USERNAME : Votre nom d'utilisateur Docker Hub.
- DOCKER\_HUB\_TOKEN : Token d'accès Docker Hub (généré dans Account Settings > Security).



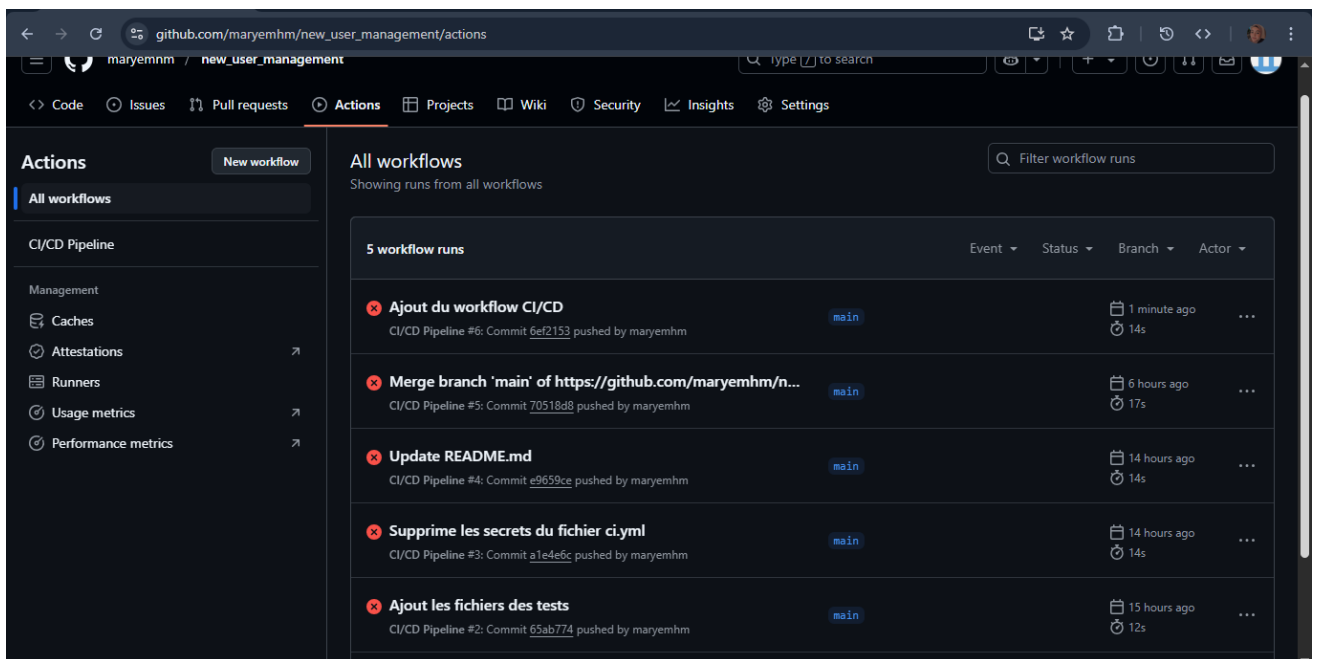
## Étapes de Test et Validation :

### Pousser le code sur GitHub :

```
git add .github/workflows/ci.yml
git commit -m "Ajout du workflow CI/CD"
git push origin main
```

```
PS C:\Users\hp\new_user_management\backend> git push origin main
Fatal: HttpRequestException encountered.
Fatal: HttpRequestException encountered.
Counting objects: 2612, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2564/2564), done.
Writing objects: 100% (2612/2612), 3.28 MiB | 830.00 KiB/s, done.
Total 2612 (delta 754), reused 0 (delta 0)
remote: Resolving deltas: 100% (754/754), completed with 6 local objects.
To https://github.com/maryemhm/new_user_management.git
70518d8..6ef2153 main -> main
```

### Resultat du workflow :



The screenshot shows the GitHub Actions interface for the repository 'maryemhm/new\_user\_management'. The 'All workflows' tab is selected, displaying a list of workflow runs. The left sidebar shows the 'Actions' section with a 'New workflow' button and a list of workflow runs. The main area shows a table of workflow runs with columns for Event, Status, Branch, and Actor. The table lists five workflow runs, all with a status of 'Failed' (indicated by a red 'X' icon). The runs are: 'Ajout du workflow CI/CD', 'Merge branch 'main' of https://github.com/maryemhm/n...', 'Update README.md', 'Supprime les secrets du fichier ci.yml', and 'Ajout les fichiers des tests'.

Workflow Name	Status	Branch	Actor	Time
Ajout du workflow CI/CD	Failed	main	maryemhm	1 minute ago
Merge branch 'main' of https://github.com/maryemhm/n...	Failed	main	maryemhm	6 hours ago
Update README.md	Failed	main	maryemhm	14 hours ago
Supprime les secrets du fichier ci.yml	Failed	main	maryemhm	14 hours ago
Ajout les fichiers des tests	Failed	main	maryemhm	15 hours ago

### Problème rencontré :

- Malgré la configuration adéquate du workflow GitHub Actions et la vérification des commits associés, certains détails des Large Commits restent masqués par défaut, limitant ainsi la visibilité complète des modifications apportées. Bien que cette option vise à optimiser l'affichage des logs, elle peut compliquer le débogage en occultant des informations critiques. Pour contourner cette limitation, il est recommandé d'utiliser la fonction de recherche (searchbox) afin d'accéder au contenu caché ou de fractionner les commits volumineux en plusieurs parties plus petites. Ces mesures permettraient d'assurer une transparence totale dans le suivi des pipelines CI/CD, évitant ainsi des blocages potentiels lors de l'analyse des erreurs.

## 6. Conclusion :

En conclusion, ce projet a été une excellente opportunité pour appliquer de manière concrète les connaissances acquises dans le cadre du cours sur le déploiement des applications web et mobiles. L'objectif principal était de développer une application Fullstack moderne, en combinant React.js pour le frontend, Express.js pour le backend, SQLite pour la base de données, et en assurant la conteneurisation avec Docker. Le tout a été intégré dans une démarche DevOps grâce à la mise en place d'un pipeline CI/CD via GitHub Actions.

Ce travail m'a permis de mieux comprendre l'architecture d'une application web complète, depuis la création d'une API REST, jusqu'à la gestion des utilisateurs à travers une interface réactive, en passant par l'orchestration des conteneurs. L'utilisation de Docker et Docker Compose a été particulièrement enrichissante, car elle m'a permis de standardiser les environnements de développement et de déploiement.

Cependant, plusieurs difficultés techniques sont survenues, notamment lors de l'intégration du backend. L'un des problèmes majeurs a été l'erreur liée à la compilation du module `sqlite3`, malgré l'application de nombreuses solutions recommandées : modification de la configuration `msvs_version`, réinstallation de Visual Studio avec les outils requis, suppression et réinstallation des dépendances, ainsi que la reconstruction complète des conteneurs Docker. Malgré ces efforts, l'erreur persistait, rendant l'API du backend inaccessible.

Par ailleurs, des problèmes sont également apparus lors des tests d'intégration. Le fichier `server.js` exportait directement l'écoute du serveur au lieu de l'instance de l'application, ce qui empêchait les tests via `chai-http`. Cette erreur a été corrigée en réorganisant les fichiers pour séparer l'initialisation du serveur de l'exportation de l'application. D'autres obstacles ont été rencontrés, notamment une base de données verrouillée par un processus actif, ce qui a nécessité l'utilisation d'une base temporaire en mémoire pour les tests.

Malgré tous les efforts méthodiques pour corriger ces problèmes, certaines fonctionnalités, notamment l'accessibilité complète de l'API backend dans l'environnement Dockerisé, n'ont pas pu être finalisées avec succès. Ces blocages ont limité le bon fonctionnement global de l'application.

Néanmoins, ce projet m'a permis de renforcer mes compétences techniques, de développer ma capacité à diagnostiquer et résoudre des problèmes complexes, et de mieux appréhender les enjeux liés au déploiement et à l'automatisation dans les projets web. Ces apprentissages me seront précieux pour la suite de mon parcours académique et professionnel.