AP3 - Documentation technique

Sommaire

1) Technologie

Pour le site M2L a été fait sous react pour le front et le backend est en JavaScript (NodeJS).
Le site contient :

- HTML
- CSS (Bootstrap 5)
- JS

2) Installation

1. Mettre à jour votre Ubuntu
sudo apt update && sudo apt upgrade

2. Installer Git, NodeJS, npm et mariadb-server
sudo apt install git nodejs npm mariadb-server

3. Configurer votre mariadb-server avec la commande
'mysql_secure_installation' sudo mysql_secure_installation

4. Avec git, récupérez votre backend se trouvant sur github et mettez le
dans le dossier /opt
cd /opt/
sudo git clone NomDuRepo

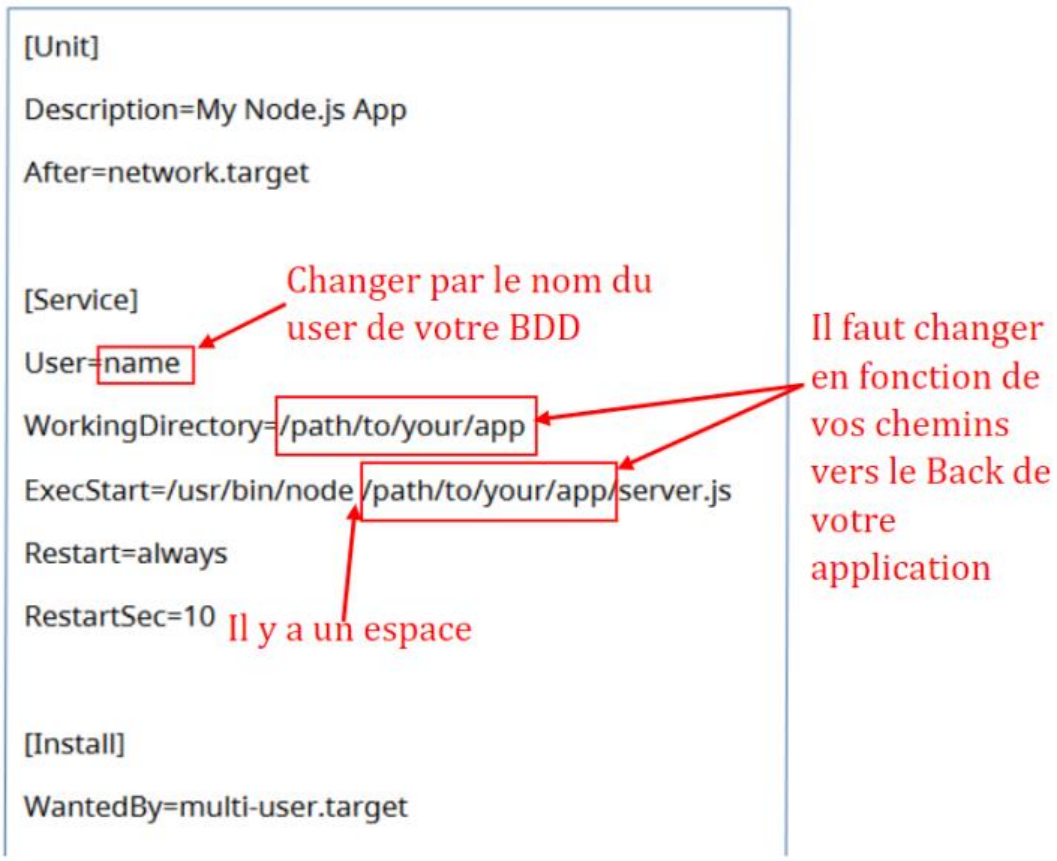
5. Déployer votre BDD sur le serveur mysql d'ubuntu, puis créer un
utilisateur qui aura accès à votre Base de données.
● sudo mysql -u root -p
● CREATE DATABASE nomDatabase;
● use nomDatabase;
● sudo mysql -u root -p nomDatabase < chemin/votreDatabase.sql
(modificable en fonction des commandes situé dans le fichier .sql).
● GRANT ALL PRIVILEGES ON votreDatabase.* TO user@localhost
identified by «votreMotDePasse»;
● FLUSH PRIVILEGES ;

6. Créer le fichier .env dans le dossier de votre projet et donner la
configuration nécessaire
sudo nano /opt/votreServeurNode/.env

7. Essayer de lancer le serveur et d'y accéder via l'ip
http://adresseIpVm:port/path (pour voir l'ip de la machine on peut faire
ip a)

8. Maintenant il faut mettre en place le fichier permettant de lancer le
serveur au démarrage, pour cela il faut créer un fichier
serveurnode.service dans le dossier etc/systemd/system/
sudo nano /etc/systemd/system/serveurnode.service

9. Dans ce fichier, il faut modifier la config suivant pour qu'elle soit conforme pour
votre machine :



10. Désormais il faut actualiser les unités systemsd puis il faut activer le service qu'on vient de créer
- sudo systemctl daemon-reload
- sudo systemctlenable serveurnode.service
- sudo systemctl start serveurnode.service

11. Redémarrer la VM et le backend se lance automatiquement !

3) Fonctionnement

Le back-end est localisé sur une machine virtuelle et elle communique avec le front qui est localisé sur le pc directement. Le backend se lance automatiquement au lancement de la VM et pour lancer le front il faut juste faire un npm start directement dans le dossier ou se trouve le front. Pour toute la partie admin je sécurise mes Routes avec le localstorage pour voir si je suis connecté et si j'ai le status 1 (le status 1 indique que je suis admin).

1. Pour s'inscrire dans l'api il y a eu plusieurs mesures de sécurité pour éviter tout type de soucis. Premièrement il y a la méthode de hachage de mot de passe pour ma part j'ai utilisé bcrypt. Il y a aussi le statut donné au compte lors de sa création, on peut voir que chaque compte lors de sa création se voit affecter 0 par default. Ce statut lui donnera les droits d'un utilisateur classique.

```
// Requete pour la création d'un compte utilisateur

app.post('/addusers', async (req, res) => {
    let conn;
        console.log("poste")
    try {
    bcrypt.hash(req.body.password, 10)
        .then(async (hash) => {
        console.log("lancement de la connexion")
        conn = await pool.getConnection();
        console.log("lancement de la requete insert")
        console.log(req.body);
        let requete = 'INSERT INTO users (pseudo, email, password, statut) VALUES (?, ?, ?, 0);'
        let rows = await conn.query(requete, [req.body.pseudo, req.body.email, hash]);
        console.log(rows);
        res.status(200).json(rows.affectedRows)
        })
    }
    catch (err) {
        console.log(err);
    }

})
```

2. Lorsque l'utilisateur soumet le formulaire de connexion, la fonction handleSubmit est appelée. Cette fonction envoie une requête POST à l'API avec les informations d'identification de l'utilisateur. Si l'API renvoie une réponse réussie, alors la fonction stockera l'email et le statut du compte dans le localstorage et une alerte indiquant la réussite de la connexion apparaîtra sur votre écran. Cela indiquera que l'utilisateur est maintenant connecté et permet au composant parent de gérer l'état de connexion de l'utilisateur et d'afficher le contenu approprié en conséquence.

Connexion.js

```
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const ls = localStorage;

let navigate = useNavigate();

const handleSubmit = async (event) => {
event.preventDefault();
console.log('connexion')
const response = await axios.post('http://localhost:8000/Connexion',
  email,
  password,
}).then(res => {
    console.log(res.data)
    if(res.status === 200) {
        ls.setItem("email", res.data.email);
        ls.setItem("statut", res.data.statut);
        console.log(ls)
        const statut = ls.getItem("statut");
        alert("Connexion reussi");
        if (statut === "0") {
            navigate("/");
        }else if (statut === "1") {
            navigate("/ModifyArticles");
        }
    }
    else{
        alert("Erreur de connexion");
    }
})
```

3. Pour afficher les articles disponibles dans la page principale et côté admin j'ai dans mon api cette méthode qui fait appel à la base de données pour afficher les articles disponibles.

```
// afficher l'article en fonction de l'id

app.get('/articles/:id', async (req, res) => {
  const id = parseInt(req.params.id)
  let conn;
  try {
      console.log("lancement de la connexion")
      conn = await pool.getConnection();
      console.log("lancement de la requete select")
      const rows = await conn.query('SELECT * FROM articles WHERE id = ?', [id]);
      res.status(200).json(rows)
  }
  catch (err) {
      console.log(err);
  }
})
```

4. Pour supprimer un article j'utilise cette méthode de mon API qui fait appel à la fonction DELETE de SQL qui permet de supprimer une donnée en fonction de son id dans la table articles et sur mon front je fais appel à la fonction handleDelete qui utilise Axios pour envoyer une requête DELETE à l'API avec l'ID de l'article à supprimer.

```
// supprimer un article

app.delete('/articles/:id', async (req, res) => {
    const id = parseInt(req.params.id)
    let conn;
    try {
        console.log("lancement de la supression")
        conn = await pool.getConnection();
        console.log("supression en cours")
        const id = parseInt(req.params.id);
        const rows = await conn.query("DELETE FROM articles WHERE id = ?", [id]);
        res.status(200).json(rows.affectedRows)
    }
    catch (err) {
        console.log(err);
    }
})
```

6. Pour afficher un article en fonction de son ID pour ensuite le supprimer ou le modifier je fais appel à cette méthode de mon API qui fait appel à la fonction SELECT * de SQL qui permet d'afficher l'article en fonction de son ID.

```
// afficher les articles

app.get('/articles', async (req, res) => {
  let conn;
  try {
      conn = await pool.getConnection();
      const rows = await conn.query('SELECT * FROM articles');
      res.status(200).json(rows)
  }
  catch (err) {
      console.log(err);
  }
})

// afficher l'article en fonction de l'id

app.get('/articles/:id', async (req, res) => {
  const id = parseInt(req.params.id)
  let conn;
  try {
      console.log("lancement de la connexion")
      conn = await pool.getConnection();
      console.log("lancement de la requete select")
      const rows = await conn.query('SELECT * FROM articles WHERE id = ?', [id]);
      res.status(200).json(rows)
  }
  catch (err) {
      console.log(err);
  }
})
```

7. Après avoir sélectionné l'utilisateur je peux modifier les données d'un utilisateur avec cette méthode de mon api qui fait appel à la fonction UPDATE de SQL ce qui va mettre à jour les données d'un utilisateur précis en fonction de son ID qui est récupéré sur mon front.

```
// modifier un article

app.put('/articles/:id', async (req, res) => {
    const id = parseInt(req.params.id)
    let conn;
    try {
        conn = await pool.getConnection();
        let requete = 'UPDATE articles SET name = ?, prix = ?, image = ?, quantite = ? WHERE id = ?;'
        let rows = await conn.query(requete, [req.body.name, req.body.prix, req.body.image, req.body.quantite, id]);
        console.log(rows);
        res.status(200).json(rows.affectedRows)
    }
    catch (err) {
        console.log(err);
    }
})
```

8. Toutes les modifications qui peuvent être appliquées sur les articles peuvent aussi être appliquées sur les comptes utilisateurs à quelques détails près.