28/11/2022
Seydou TOLOBA
Boubacar DOUCOURE

Réalisation d’une application REACT JS et MySQL pour la gestion d’un restaurant

Rapport sur le deroulement de l’activité et du foctionnement de l’application

TECHNOLAB-ISTA

Sommaire

I. Présentation

A. Contexte:

C'est dans le cadre de validation d'examin en REACT JS et Design Patterns que nous réalisons cet projet qui consiste en la réalisation d'une application web pour un restaurant Dénommé Dou Ka Fa.

B. Comprehension cahier de charge (CDC):

Dou ka fa est un restaurant offrant une varité de menu à prix très concourant à ses clients. Pour faire face aux concurrents elle nous demande de mettre en place une application mono-user(un seul utilisateur) qui permettra de faire les activités suivantes:

1. Gérer les menus(La division des menus en catégorie);
2. Gérer les ventes/commandes(Statistiques);
3. Gérer les clients;
4. Gérer les points de fidélité(des points de fidélités sont atribué lorsqu'un client fait un achaht et donne son numéro de téléphone, un seuil de fidelité etc…);
5. La livraison des commandes et l'authentification.

Notre client attend les fonctionalités suivantes:

1. lister les statistiques de ventes/commandes susmentionnés ;
2. ajouter, modifier, supprimer, lister les menus ;
3. ajouter, modifier, supprimer, lister les ventes/commandes ;
4. ajouter, modifier, supprimer, lister les clients ;
5. modifier le compte utilisateur.

II. Analyse et conception:

A. Analyse

01. Identification des acteurs:

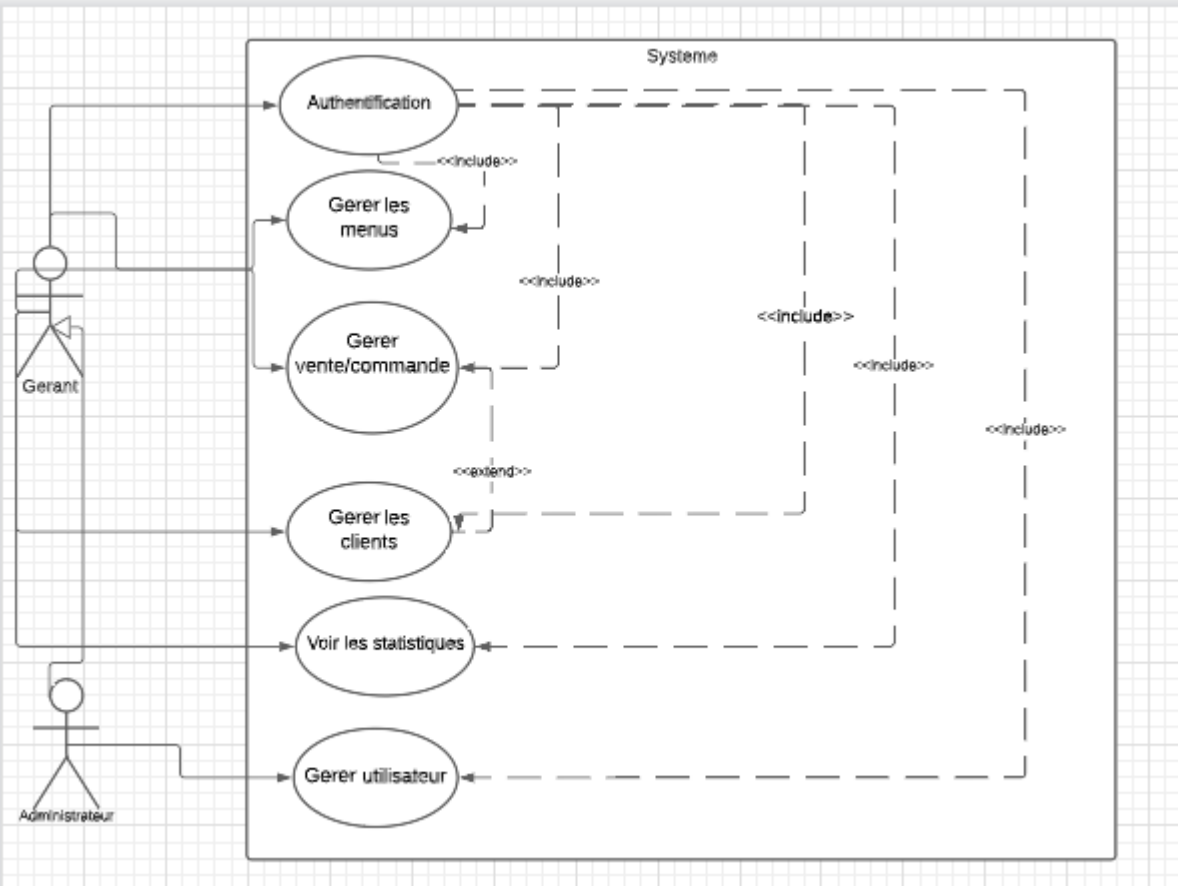
Dans le cahier de charge on nous indique que l'applicatio sera mono-user(utilisateur unique ou seul) donc seulement le personnel qualifier du restaurant pour l'utilisé.

02. Identification, réalisation des cas d'utilisation:

Les différents cas d'utilisation que nous avons ressencé sont:

- L'authentification;
- Gestion menus;
- Gestion ventes/commandes;
- Gestion clients;
- Gestion paiements;
- Gestion Livraison;

03. description textuelle et graphique des cas d'utilisations



- **L’authentification:** C’est autrement dit se connexion en mettant le login et le mot de passe.
  Ainsi il pourra accéder à l’application et pourra faire les opérations donc l’activité nécessite.
- **Gérer menus**: dans le cas d’utilisation gérer les menus consiste à l’ajout, la modification et la suppression des articles qui forment notre menu; et à la constitution d’un menu du jour

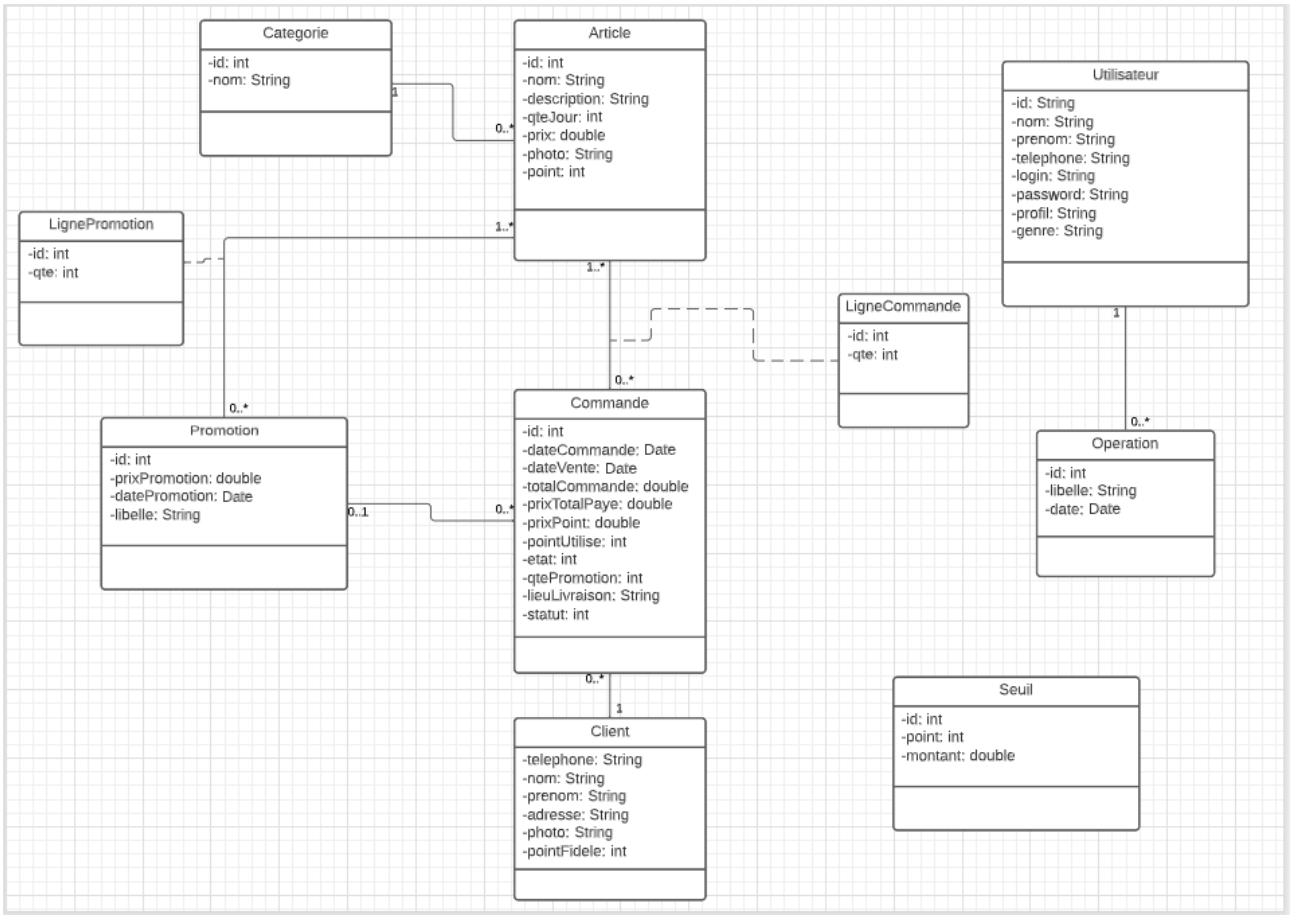
- **Gérer vente/commande:** Il s’agit d’enregistrer les commandes, gérer leur état et finaliser les commandes ou les annuler. Une commande enregistrer comme vente ne peut être supprimée que par un administrateur.
  Une commande ne nécessite pas forcément un client et on peut enregistrer directement un client lors de la commande. Après la vente, les points de fidélité du client sont incrémentés en fonction des points prévus pour les articles de sa commande.

- **Gérer client:** Les actions à mener sont l’ajout, la modification et la suppression.
- **Voir les statistiques:** Il s’agit d’avoir des informations concernant les ventes/commandes journalier ou périodique, nombre de clients, total vente, menus vendus etc…
  Nous aurons donc une idée sur le déroulement de l’activité et déciderons en conséquence.
- **Gérer utilisateur:** On peut ajouter un utilisateur, le modifier(mon, mot de passe, login et autre) ou le supprimer

NB: Les cas d’utilisations gérer menus, gérer vente/commande, gérer client, voir statistique, gérer utilisateur incluent tous préalablement une authentification de la part d’un utilisateur.

B. Conception

Diagramme de classes



- **Catégorie**
  - **Attributs: id, nom**
  - **Détails:** Catégories de plat.
- **Classe article**
  - **Attributs:** Id, nom, qteJour, photo, point.
  - **Détails:** la classe article ou plat concerne les article ou produit disponible ou vendus dans le restaurant
- **Classe commande**
  - **Attributs:** id, dateCommande, dateVente, totalCommande, prixTotalPayé, prix point, pointUtilisé, etat, qtePromotion
  - **Détails:** cette classe fait ressortir les informations nécessaire sur la commande.
- **Classe ligneCommande**
  - **Attributs:** id, qte.
  - **Détails:** Elle contiendra les articles associés à une commande et leur quantité

- **Promotion**
  - **Attributs:** id, prixPromotion, datePromotion, libelleé
  - **Détails:** Il s'agit dans le cas du restaurant, il s'agit du menu du jour car elle bénéficie d'une promotion sur le prix total.
- **Ligne Promotion**
  - **Atttributs:** id, qte
  - **Détails:** Elle contiendra les plats dont une promotion fait allusion et leur quantité
- **Utilisateur**
  - **Attributs:** id, nom, prenom, téléphone, login, password, profil, genre
  - **Détails:** Elle regroupe les information sur l'utilisateur
- **Operation:**
  - **Attributs:** id, libelle, date
  - **Détails:** La classe Opération est considéré comme le log, elle a pour vocation de tracer l'ensemble des opération effectuées par les utilisateurs
- **Seuil**
  - **Attributs:** id, point, montant
  - **Détails:** Il s'agit du seuil, d'un plafonnement de point. Si le client atteint ce seuil, un montant lui est attribuer

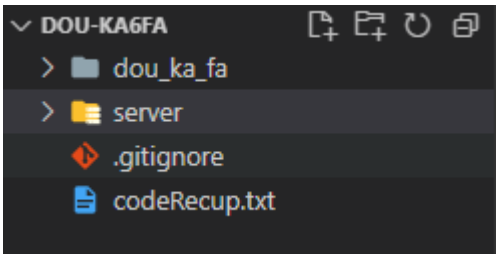
Rélations:

les relations existantes entre différents classes:

1. **Catégorie-article:** Dans une catégorie on peut avoir plusieur article, mais dans un article ne peut appartenir qu'à un quel catégorie
2. **Article-promotion:** un article peut faire l'objet de zéro(0) ou plusieurs promotion et une promotion peut faire allusion à un ou plusieurs articles. Entre les deux nous avons une classe associative **LignePromotion** qui prendra la quantité.
3. **Article-commande:** un article peut faire l'objet de zéro(0) ou plusieurs commandes et une commande peut inclure 1 ou plusieurs articles. Entre les deux nous avons une classe associative **LigneCommande** qui prendra la quantité
4. **Commande-client:** Une commande peut être passer par un et un seul client, un client peut passer zéro(0) ou plusieurs commandes.
5. **Utilisateur-operation:** un utilisateur peut effectuer zéro(0) ou plusieurs opérations mais une même opération ne peut être effectué que par un et un seul utilisateur à la fois.

III. Module2: Implementation

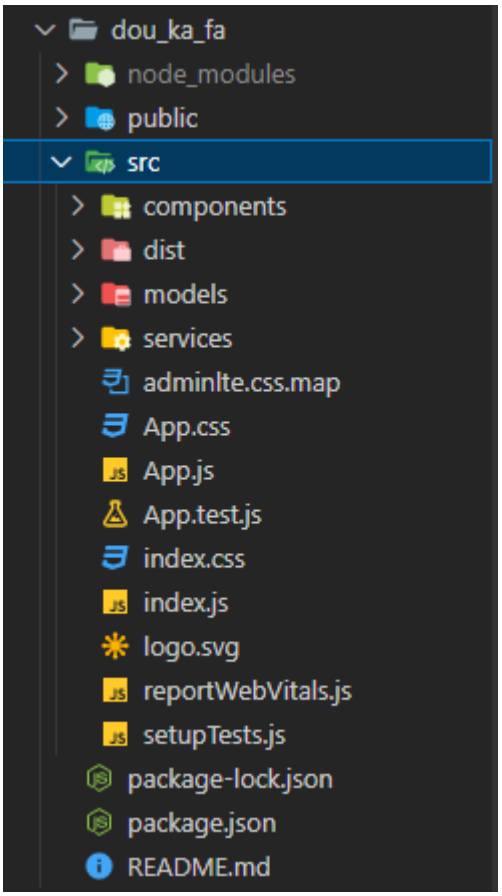
A. Structure de projet



Le projet est divisé en deux parties

1. Le front-end en REACTJS: **dou_ka_fa**
2. Le back-end avec REACTJS et une base de donnée MySQL: **server**

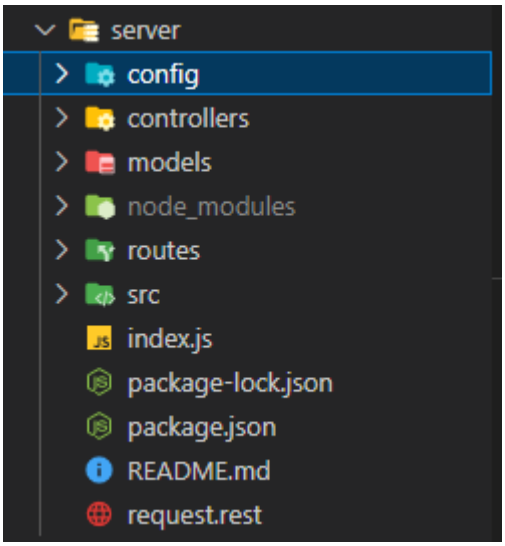
1) Dou_ka_fa



01. **Node_module:** Contient toutes les librairies nécessaires au développement du projet il est créé par node à partir du fichier **package.json**.
02. **Public:** contient tout les ressources public ou partager dans le projet
03. **Src:** Est le dossier qui va plus nous intéresser car elle contient les pages, les modèles et les services de notre application.
04. **Components:** Il contient les vues que nous voulons montrées à l'utilisateur.

05. **Dist:** Contient le css et autres fichiers concernant le template utilisé dans le projet.
06. **Models:** Contient les classes.
07. **Services:** Contient les fichiers qui contiennent les méthodes d'accès au backend pour les traitements dans la base de données **.**
08. **App.js:** C'est la première page appelée par le fichier index.js lors du lancement de l'application. Nous l'avons utilisé comme un conteneur qui vas nous afficher la page de Login.js Si on a pas d'utilisateur logger ou la page Base.js si on a un utilisateur connecté.
09. **Index.js:** c'est le fichier qui s'exécute au lancement du projet
10. **Package.json:** Elle contient le nom des librairies et leurs versions installées
11. **Package-lock.json:** peut être considéré comme le log et est générer après le lancement de la commande **npm i.**

2) Server

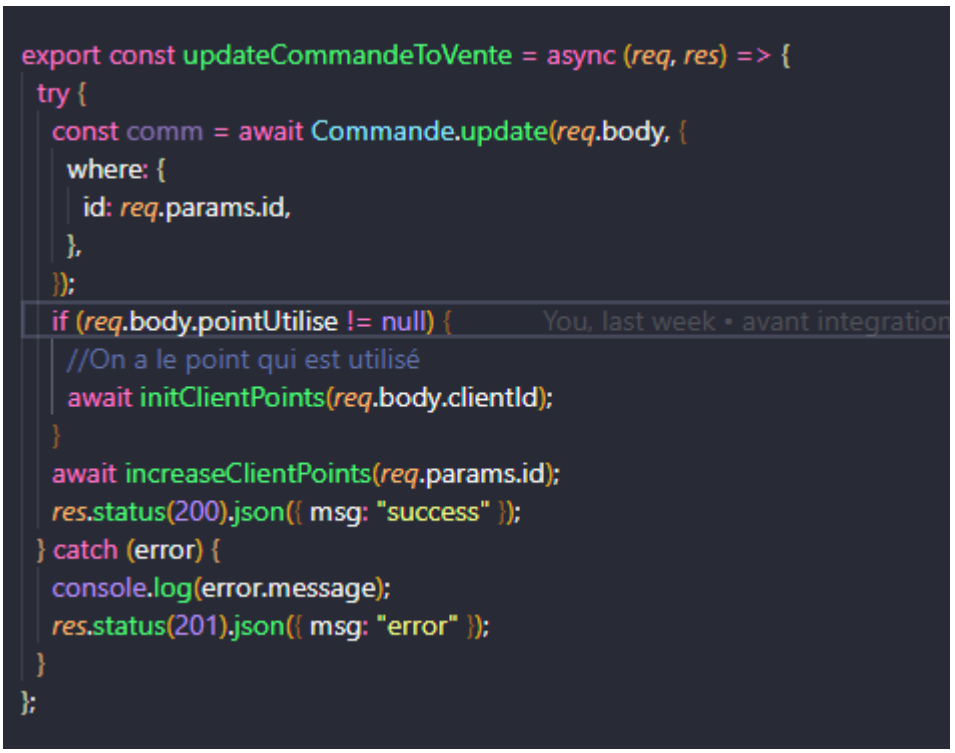


01. **Config:** Continent les configurations de la base de données
02. **Controller:** Contient les méthodes qui vont agir sur les données
03. **Models:** Contient les fichiers contenant les instructions de création de nos tables dans la base de données et qui nous permettent aussi de faire les traitements sur les données des dits tables qui sont créées.

04. **Routes:** Définis les routes et les méthodes dans les controllers qui seront utilisés pour effectuer le traitement demandé..
05. **Index.js:** nous permet d'utiliser nos routes

B. Portions de code clés

Méthode Utilisé lors de la transformation d'une commande en vente



```
export const updateCommandeToVente = async (req, res) => {
  try {
    const comm = await Commande.update(req.body, {
      where: {
        id: req.params.id,
      },
    });
    if (req.body.pointUtilise != null) {          You, last week • avant integration
      //On a le point qui est utilisé
      await initClientPoints(req.body.clientId);
    }
    await increaseClientPoints(req.params.id);
    res.status(200).json({ msg: "success" });
  } catch (error) {
    console.log(error.message);
    res.status(201).json({ msg: "error" });
  }
};
```

Methode d’ajout des points des articles dans la commande au point de fidelité du client

```
async function increaseClientPoints(commandeId) {
  /*
   Methode permettant de'ajouter les point de fidelité prevu par les articles
   d'une commande apès la validation de la commande
  */
  try {
    await Commande.findOne({
      where: {
        id: commandeId,
      },
    }).then(async (commande) => {
      await Client.findOne({
        where: {
          id: commande.clientId,
        },
      }).then(async (client) => {
        if (client != null)
          await LigneCommande.findAll({
            where: {
              commandeId: commandeId,
            },
          }).then(async (listLigneCommande) => {
            listLigneCommande.forEach(async (ligne) => {
              await Article.findOne({
                where: {
                  id: ligne.articleId,
                },
              }).then(async (article) => {
                await Client.update(
                  {
                    point:
                      Number(client.point) +
                      Number(article.point) * Number(ligne.qte),
                  },
                  {
                    where: {
                      id: client.id,
                    },
                  }
                ).then(async (response) => {
                  await Client.findOne({
                    where: {
                      id: commande.clientId,
                    },
                  }).then(async (client2) => {
                    await Seuil.findAll().then(async (seuils) => {
                      if (seuils.length > 0) {
                        if (client2.point >= seuils[0].point) {
                          await Client.update(
                            {          You, last week • avant integrationde la modération
                              point: seuils[0].point,
                            },
                            {
                              where: {
                                id: client2.id,
                              },
                            }
                          );
                        }
                      } else {
                        //on met l'ancien point du client
                        await Client.update(
                          {
                            point: client.point,
                          },
                          {
                            where: {
                              id: client2.id,
                            },
                          }
                        );
                      }
                    });
                  });
                });
              });
            });
          });
      });
    });
  } catch (error) {
    console.log(error);
  }
}
```

Méthode qui initialise les points de fidélités dun client après utilisation des points lors d'une commande

```
async function initClientPoints(clientId) {          You, last week • avant integrationde la modération ...
  /*
    Methode qui permet d'initialiser les points d'un client après l'utilisation des points dans  une commande
  */
  await Client.update(
    {
      point: 0,
    },
    {
      where: {
        id: clientId,
      },
    }
  );
}
```

Méthode pour mettre à jour (modifier) un menu du jour

```
84
85    export const updateMenuJour = async (req, res) => {
86      console.log(req.params);
87      try {
88        await Promotion.update(req.body.menu, {
89          where: {
90            id: req.params.id,
91          },
92        }).then(async (reussi) => {
93          LignePromotion.destroy({
94            where: {
95              promotionId: req.params.id,
96            },
97          }).then((li) => {
98            req.body.lignes.forEach(async (ligne) => {
99              ligne.promotionId = req.params.id;
100             await LignePromotion.create(ligne);
101           });
102         });
103
104         res.status(201).json({ msg: "success" });
105         // await LignePromotion.create
106       });
107     } catch (error) {
108       console.log(error.message);
109       res.status(201).json({ msg: "Failed to create Promotion" });
110     }
111   };          You, 7 days ago • après debut du dashboard
```

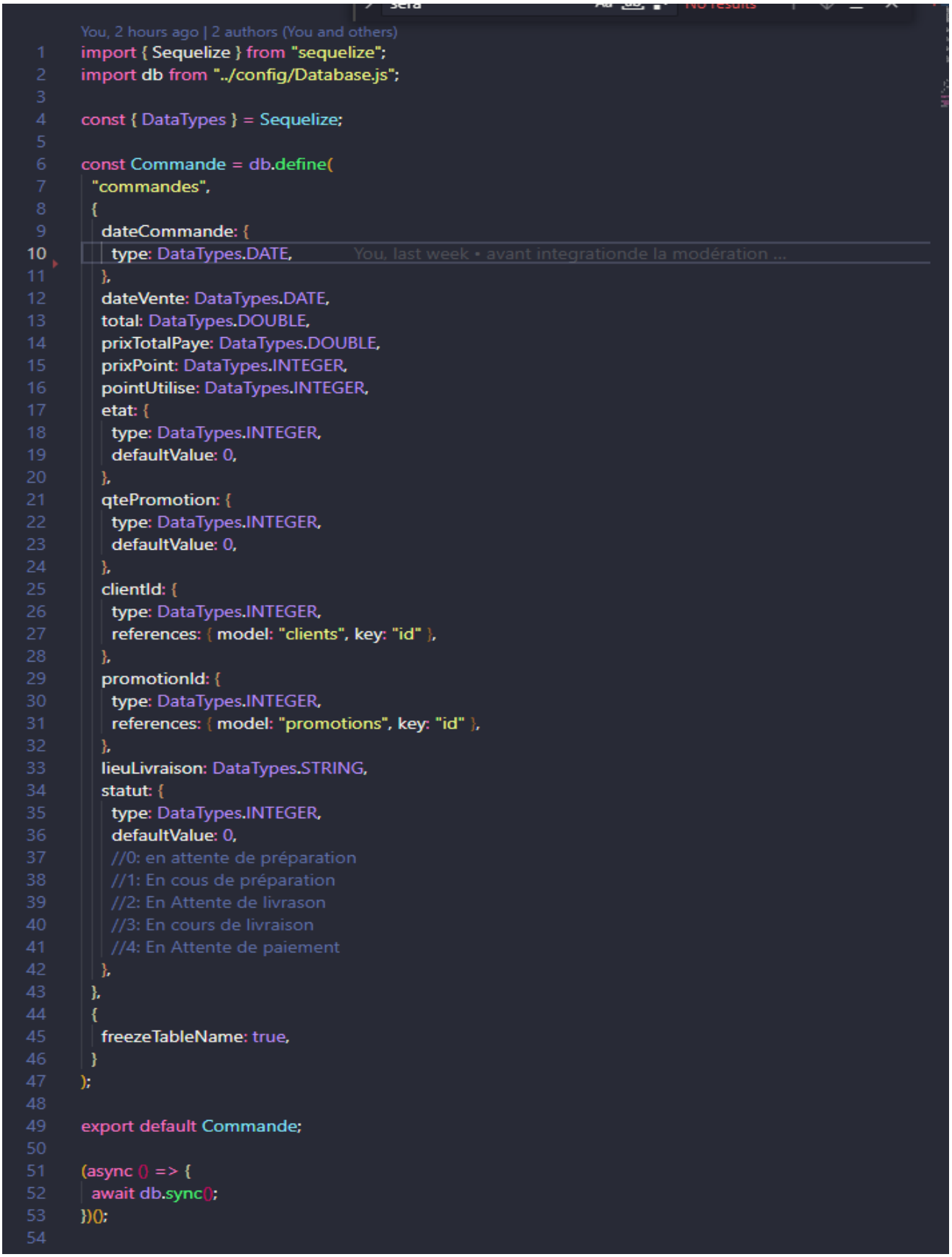
Methode utilisé pour recupérer le menu du jour de la journée

```
export const getTodayMenu = async (req, res) => {          You, 7 days ago • après debut du das
  /*
    Methode qui recupère  le menu du jour d'aujourdh'ui
  */
  try {
    let dateMin = new Date();
    dateMin.setMilliseconds(0);
    dateMin.setSeconds(0);
    dateMin.setMinutes(0);
    dateMin.setHours(0);

    let dateMax = new Date();
    dateMax.setMilliseconds(999);
    dateMax.setSeconds(59);
    dateMax.setMinutes(59);
    dateMax.setHours(23);
    await Promotion.findOne({
      where: {
        datePromotion: {
          [Op.gt]: dateMin,
          [Op.lt]: dateMax,
        },
      },
    }).then(async (menuJour) => {
      if (menuJour) {
        await LignePromotion.findAll({
          where: {
            promotionId: menuJour.id,
          },
        }).then(async (lignes) => {
          await Commande.findAll({
            where: {
              promotionId: menuJour.id,
            },
          }).then((list) => {
            console.log("succes");
            list.length > 0
              ? res
                  .status(200)
                  .json({ menuJour: menuJour, lignes: lignes, isUsed: true })
              : res
                  .status(200)
                  .json({ menuJour: menuJour, lignes: lignes, isUsed: false });
          });
        });
      } else {
        res.status(200).json({ menuJour: null, lignes: [], isUsed: false });
      }
    });
  } catch (error) {
    console.log(error.message);
  }
};
```

Méthode de recherche dans la liste des articles

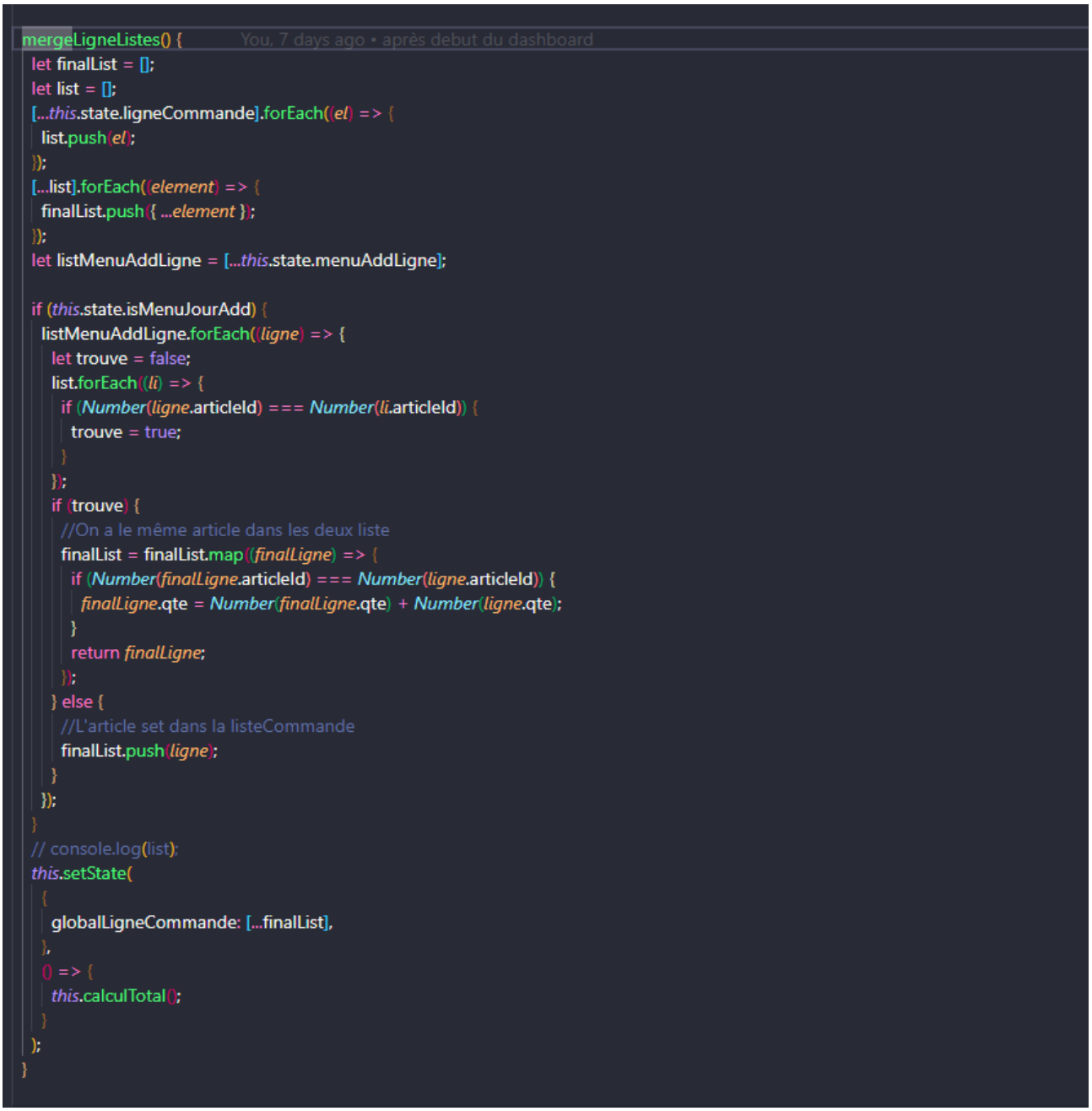
```
search = () => {          You, 8 hours ago • après un peu de style et les filtres des tables
  if (this.state.searchWord.length > 0) {
    if (this.state.searchWord.split("#").length > 1) {
      if (this.state.searchWord.split("#")[0].toLowerCase() === "c") {
        this.setState({
          tableRows: [...this.state.listArticle].filter((art) =>
            art.categorieName.includes(this.state.searchWord.split("#")[1])
          ),
        });
      } else if (this.state.searchWord.split("#")[0].toLowerCase() === "n") {
        this.setState({
          tableRows: [...this.state.listArticle].filter((art) =>
            art.nom.includes(this.state.searchWord.split("#")[1])
          ),
        });
      }
    } else {
      this.setState({
        tableRows: [...this.state.listArticle].filter(
          (art) =>
            art.nom.includes(this.state.searchWord) ||
            art.description.includes(this.state.searchWord) ||
            art.qteJour.toString().includes(this.state.searchWord) ||
            art.prix.toString().includes(this.state.searchWord) ||
            art.point.toString().includes(this.state.searchWord) ||
            art.categorieName.includes(this.state.searchWord)
        ),
      });
    }
  } else {
    this.setState({
      tableRows: [...this.state.listArticle],
    });
  }
};
```

Méthode de création de la table commande dans la base de donnée avec le model commande du ‘**server’**



```
You, 2 hours ago | 2 authors (You and others)
import { Sequelize } from "sequelize";
import db from "../config/Database.js";

const { DataTypes } = Sequelize;

const Commande = db.define(
  "commandes",
  {
    dateCommande: {
      type: DataTypes.DATE,          You, last week • avant integrationde la modération ...
    },
    dateVente: DataTypes.DATE,
    total: DataTypes.DOUBLE,
    prixTotalPaye: DataTypes.DOUBLE,
    prixPoint: DataTypes.INTEGER,
    pointUtilise: DataTypes.INTEGER,
    etat: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    qtePromotion: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
    },
    clientId: {
      type: DataTypes.INTEGER,
      references: { model: "clients", key: "id" },
    },
    promotionId: {
      type: DataTypes.INTEGER,
      references: { model: "promotions", key: "id" },
    },
    lieuLivraison: DataTypes.STRING,
    statut: {
      type: DataTypes.INTEGER,
      defaultValue: 0,
      //0: en attente de préparation
      //1: En cous de préparation
      //2: En Attente de livrason
      //3: En cours de livraison
      //4: En Attente de paiement
    },
  },
  {
    freezeTableName: true,
  }
);

export default Commande;

(async () => {
  await db.sync();
})();
```

Méthode pour melanger la liste des articles dans la promotion et des articles ajouter avec la ligne lors de la création d'une commande

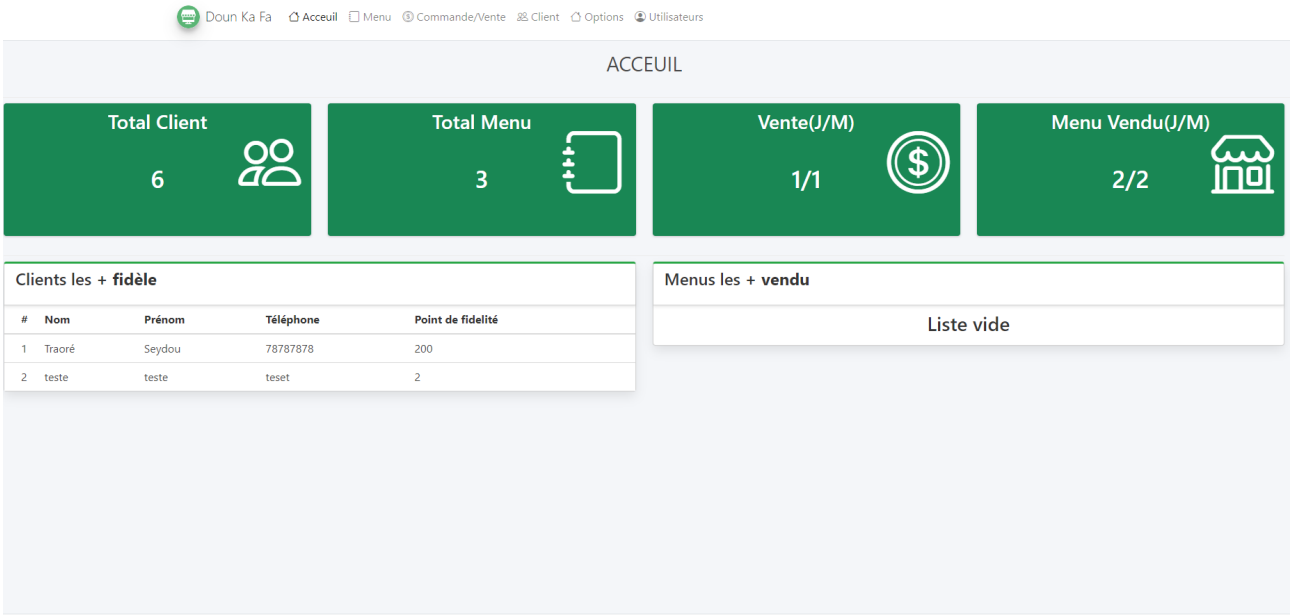


```
mergeLigneListes() {          You, 7 days ago • après debut du dashboard
  let finalList = [];
  let list = [];
  [...this.state.ligneCommande].forEach((el) => {
    list.push(el);
  });
  [...list].forEach((element) => {
    finalList.push({ ...element });
  });
  let listMenuAddLigne = [...this.state.menuAddLigne];

  if (this.state.isMenuJourAdd) {
    listMenuAddLigne.forEach((ligne) => {
      let trouve = false;
      list.forEach((li) => {
        if (Number(ligne.articleId) === Number(li.articleId)) {
          trouve = true;
        }
      });
      if (trouve) {
        //On a le même article dans les deux liste
        finalList = finalList.map((finalLigne) => {
          if (Number(finalLigne.articleId) === Number(ligne.articleId)) {
            finalLigne.qte = Number(finalLigne.qte) + Number(ligne.qte);
          }
          return finalLigne;
        });
      } else {
        //L'article set dans la listeCommande
        finalList.push(ligne);
      }
    });
  }
  // console.log(list);
  this.setState(
    {
      globalLigneCommande: [...finalList],
    },
    () => {
      this.calculTotal();
    }
  );
}
```

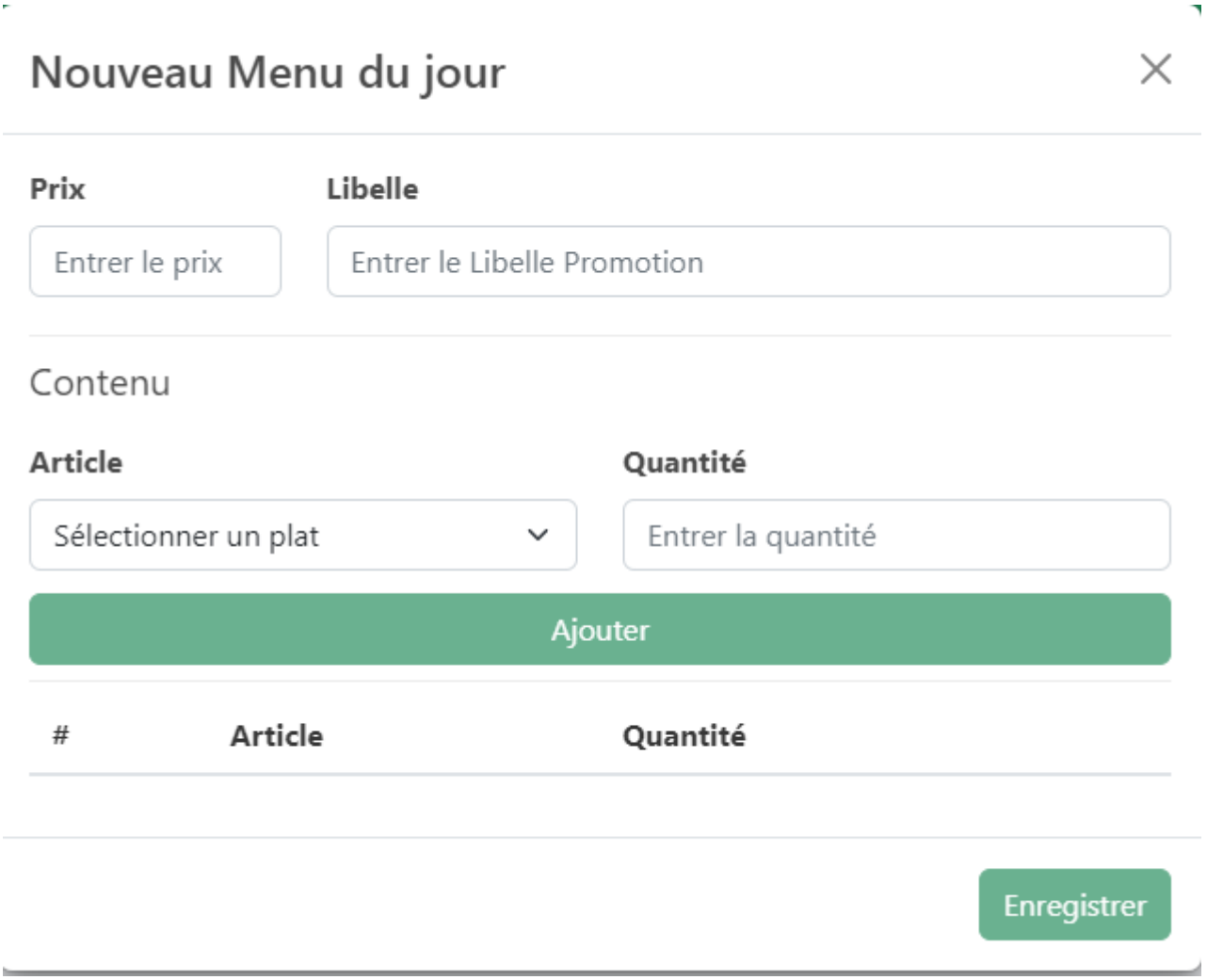
C. Capture d’écran de l’application

Ecran de login

Ecran d’acceuil



Ecran de création du menu du jour



Ecran de création de la commande

Ecran de validation d'annulation de de gestion des états d'une commande.

IV. Deploiement

A. Deploiement sur Heroko

Le deploiement sur Heroku n'a pu s'effectuer car nous avons besoin d'un ***Add-on*** pour le coté BDD, pour ce faire une vérification de compte est néccessaire et celle-ci passe par le renseigement d'une carte de crédit, carte que nous ne disposons pas actuellement.

B. Lien vers le Dépôt git

Pour accéder au dépot git voici le lien:
https://github.com/doucoure05/Dou-ka6fa.git

C. Lien vers l'application

Application non déployer

Nous vous rémercions pour votre attention