

Rapport de Projet

Table des matières

- 1. Contexte**
- 2. Obligation**
- 3. Documentation d'utilisation**
- 4. Documentation technique**

1. Contexte

La demande :

La ligue sportive d'auvergne est une association sportive qui souhaite effectuer la location de matériel sportif à leurs adhérents. Le concept est simple : le site affiche les listes des produits disponibles à la location (ballons, chasuble).

Les utilisateurs peuvent s'inscrire pour être adhérents. Les adhérents sont représentés avec un nom, prénom, téléphone, courriel, mot de passe. Une fois inscrit, le nouvel adhérent peut se connecter au site et ajouter les produits nécessaires dans un « panier ». Si le « panier » est confirmé, les quantités des produits sont retirées.

Projet à réaliser :

Réaliser le site web présenter ci-dessus. Il faudra prendre en compte, qu'un administrateur est également présent, il peut gérer les adhérents (CRUD) et les produits (CRUD).

Le site doit louer des produits qui sont dans le thème d'un sport : par exemple, la location des produits de foot uniquement (ballon, chasuble, maillot arbitre), ou une location de produits de natation (bonnet, ballon water-polo, lunettes de plongé).

2. Obligation

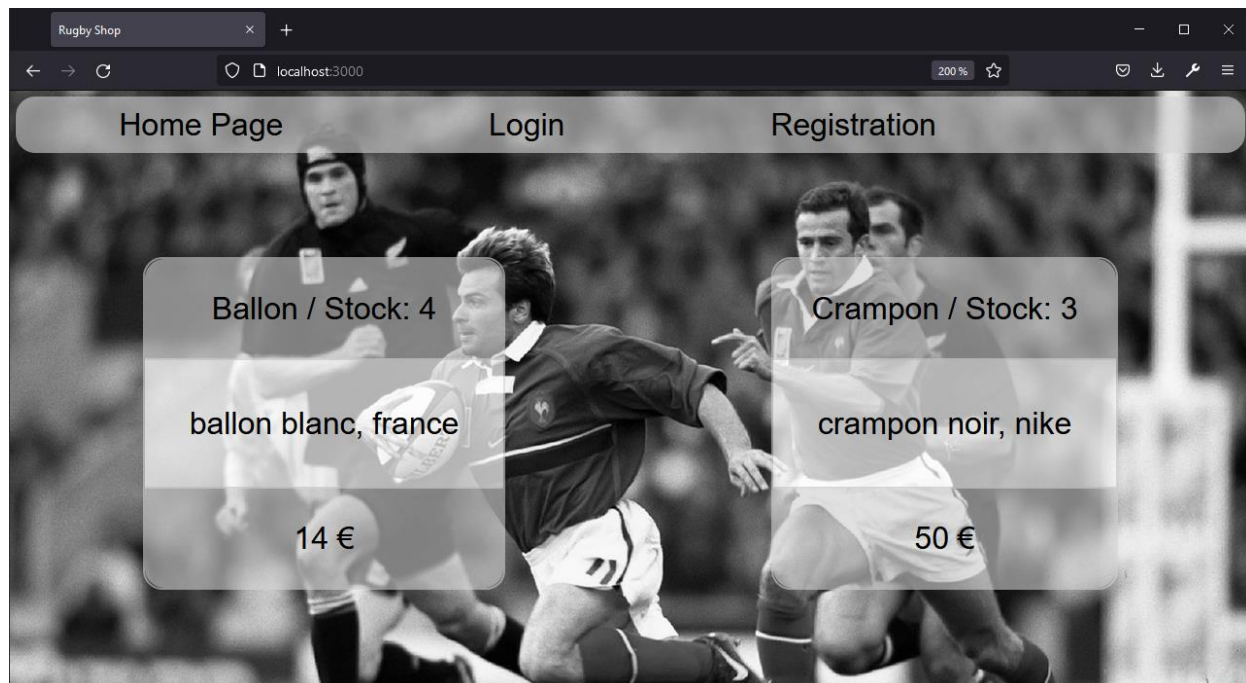
Langages et outils obligatoires :

Une base de données MySQL/MariaDB.

Un backend en NodeJS, et une API (utiliser la librairie Express).

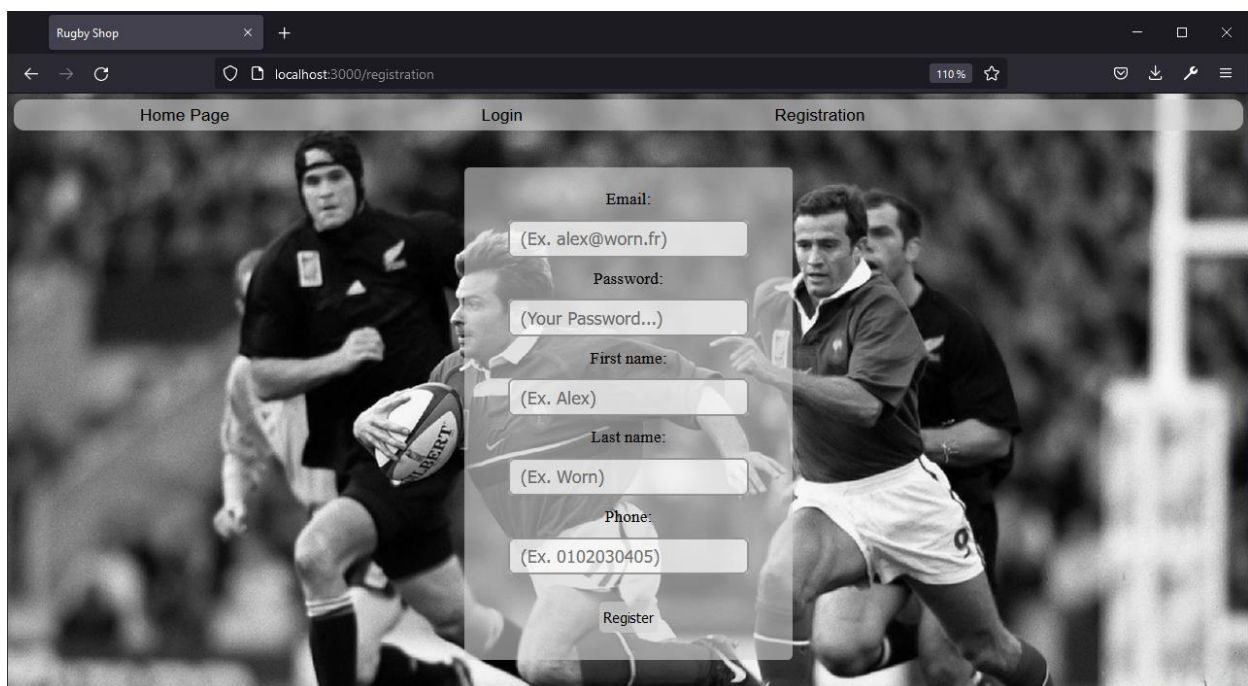
Un Front-End avec React ou tout autre framework Javascript Front.

3. Documentation d'utilisation



Lors de l'arrivée sur le site web tout le monde peut voir les produits disponibles et leurs stocks.

Si vous n'êtes pas inscrit cliquer sur « Registration » cela vous affichera cette page ci-dessous :

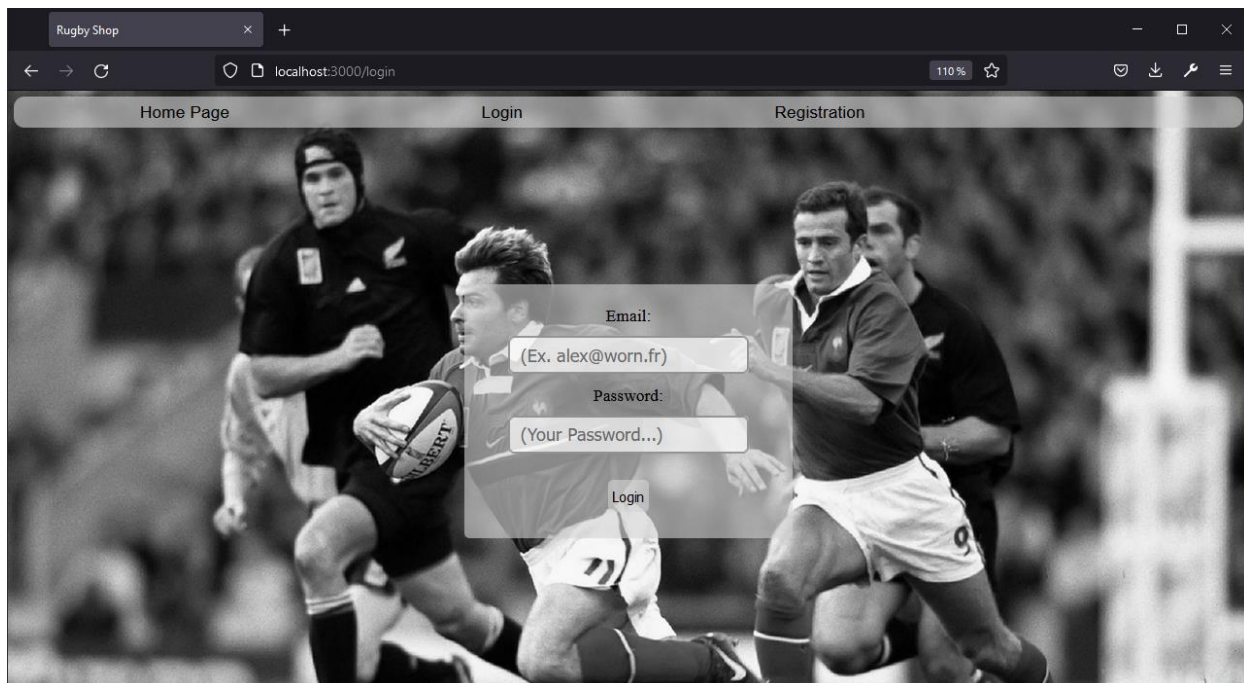


The screenshot shows a web browser window with the title 'Rugby Shop'. The address bar shows 'localhost:3000/registration'. The page has a navigation bar with three links: 'Home Page', 'Login', and 'Registration'. The background is a black and white photograph of rugby players in action. A semi-transparent registration form is centered on the page, containing the following fields and labels:

- Email: (Ex. alex@worn.fr)
- Password: (Your Password...)
- First name: (Ex. Alex)
- Last name: (Ex. Worn)
- Phone: (Ex. 0102030405)
- Register

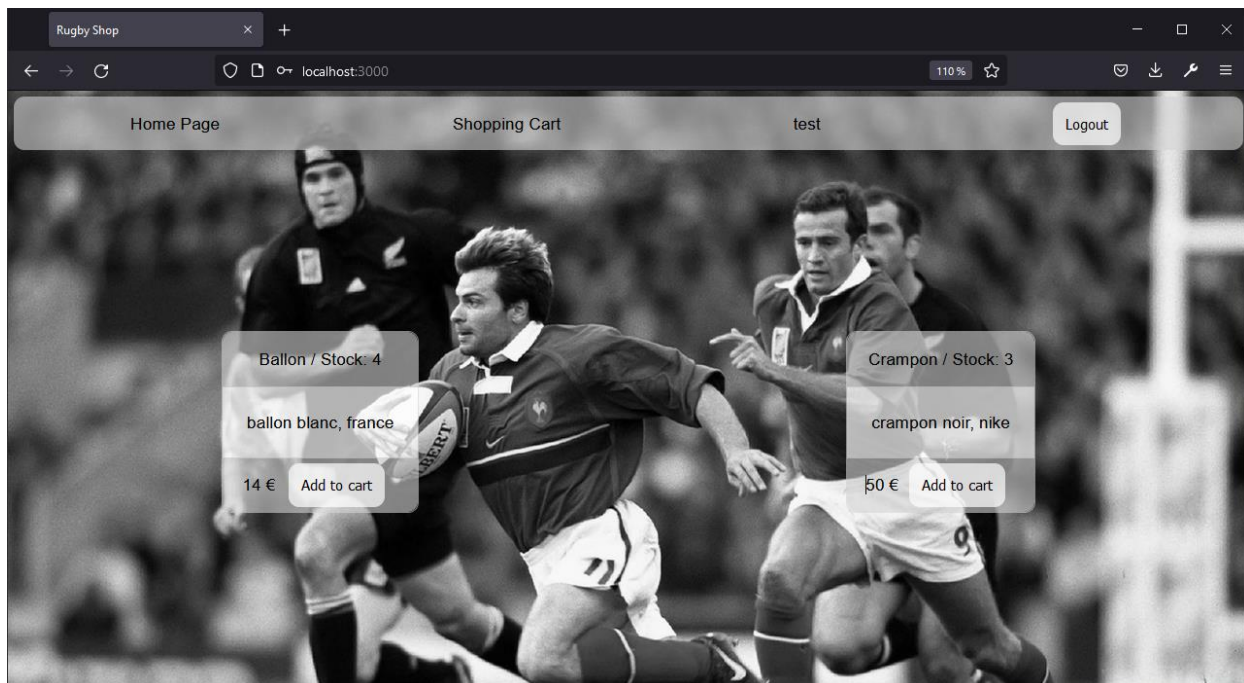
Saisir ces informations personnelles, une fois toutes les informations rentrer cliquer sur « Register ».

Une fois cela fait, nous serons redirigés sur la page de connexion :



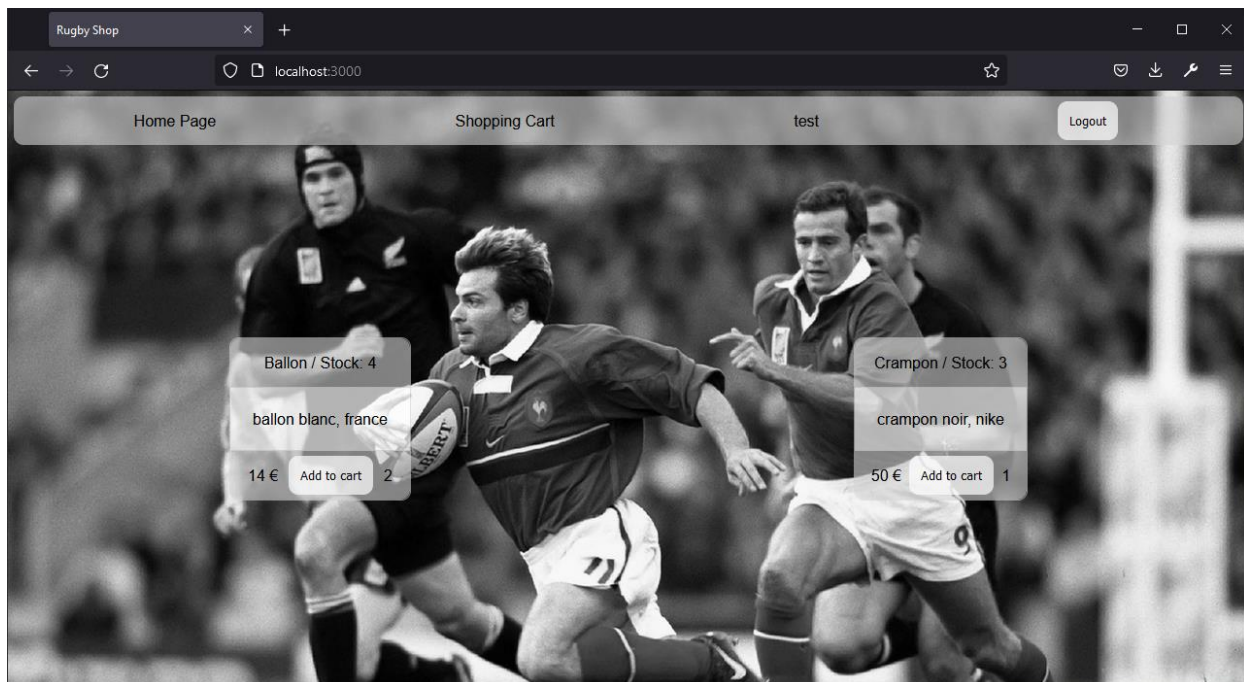
Entrez son courriel et son mot de passe défini lors de votre inscription (ou les informations données par votre administrateur) et cliquer sur « Login ».

Nous serons redirigées sur la page d'accueil :

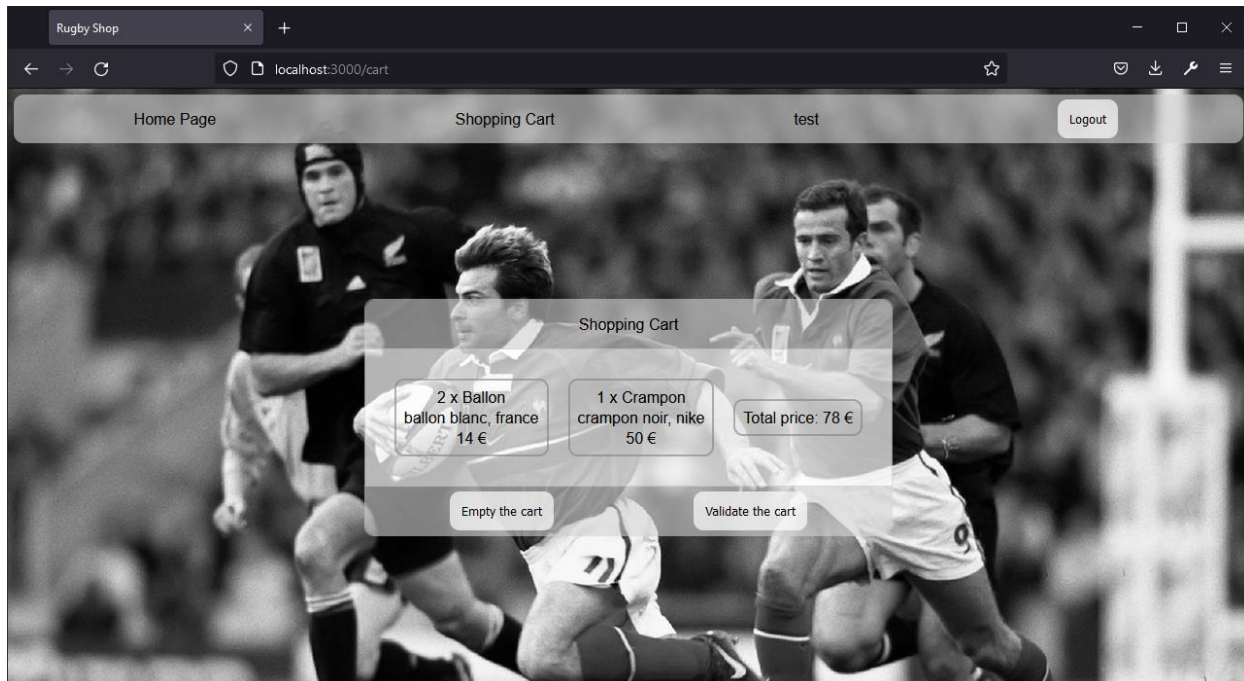


De nouvelles choses apparaissent, nous pouvons désormais ajouter des produits à son panier en cliquant sur le bouton « Add to cart » de l'article.

Une fois cliquer un chiffre apparaitra signifiant la quantité d'article ajouter au panier.



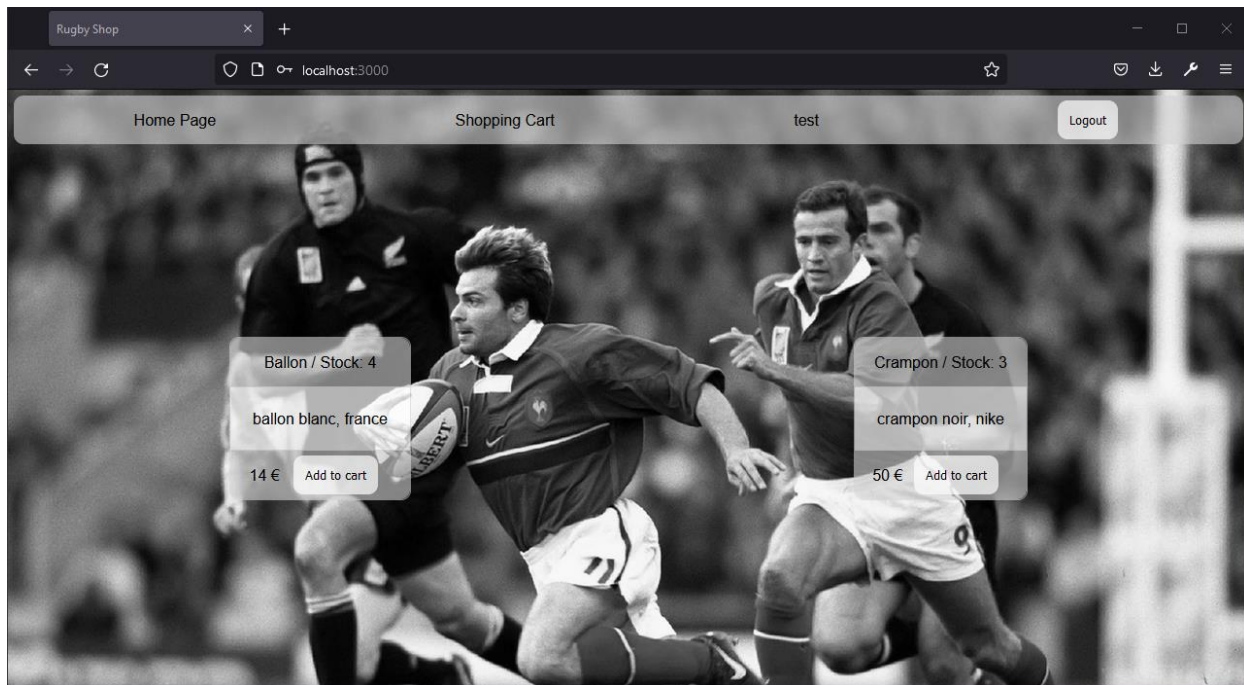
Une fois tous les articles ajoutés voulu au panier cliquer sur « Shopping Cart », nous serons redirigés sur la page du panier :



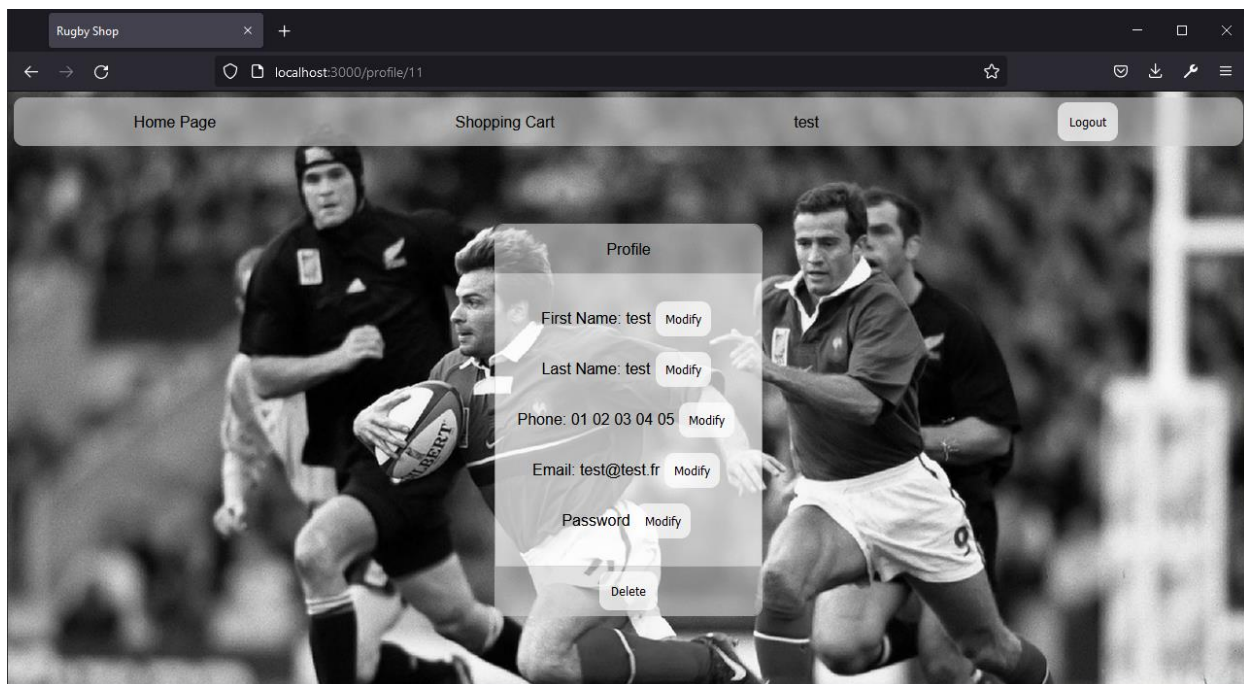
Nous pouvons voir le nombre d'articles ajouter au panier et le prix total de tout les article ajouter au panier.

Pour vider le panier cliquer sur « Empty the cart », se qui nous redirigera sur le page de tous les articles disponibles.

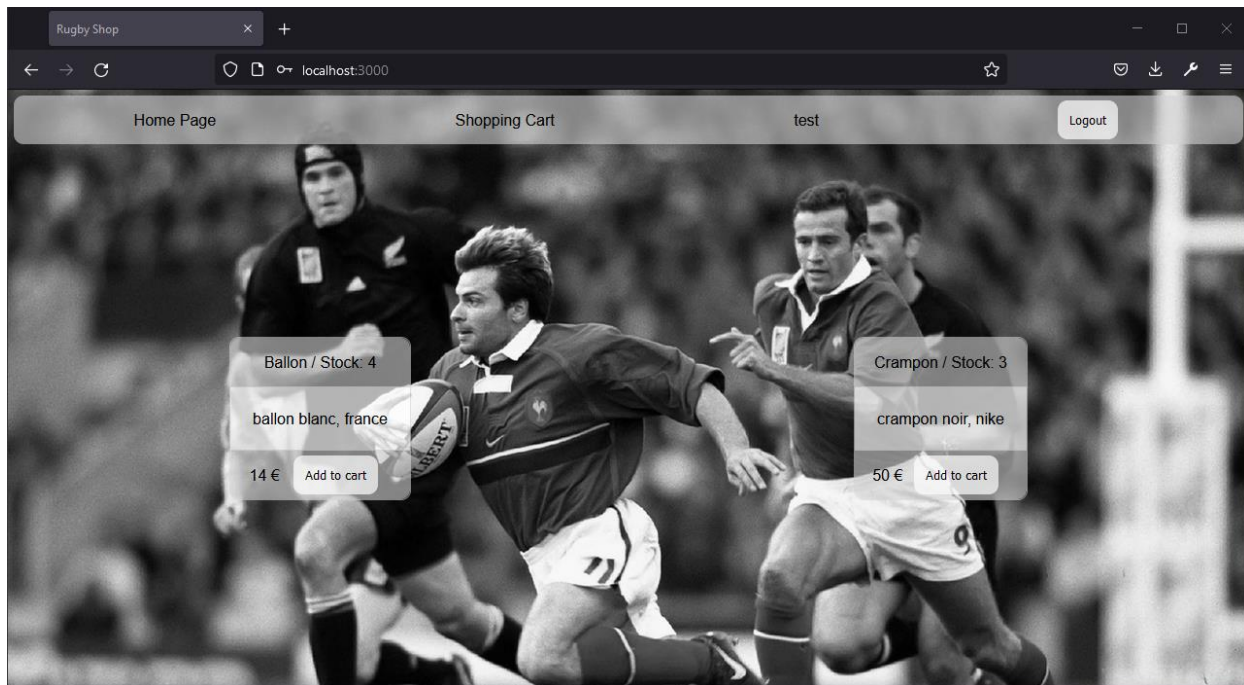
Pour valider le panier cliquer sur « Validate the cart », cela déduira le nombre d'article en stock et nous redirigera sur le page de tous les articles disponibles.



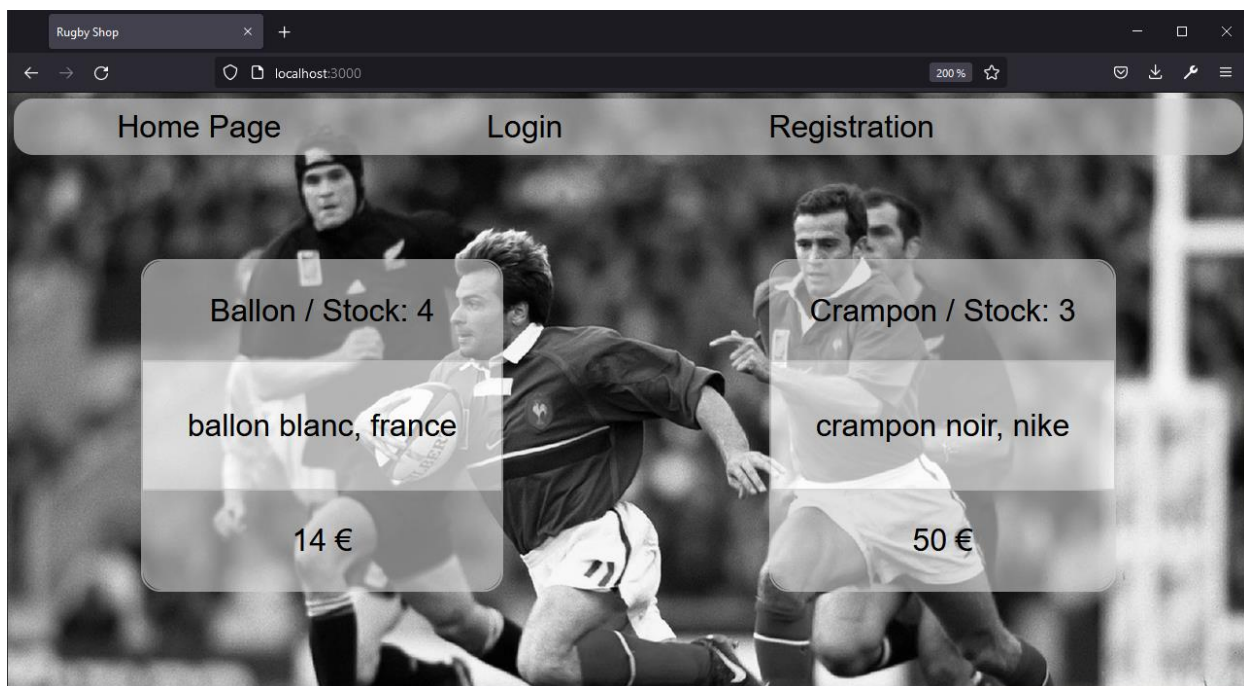
Cliquer sur votre prénom (« test ») pour accéder à votre profil.



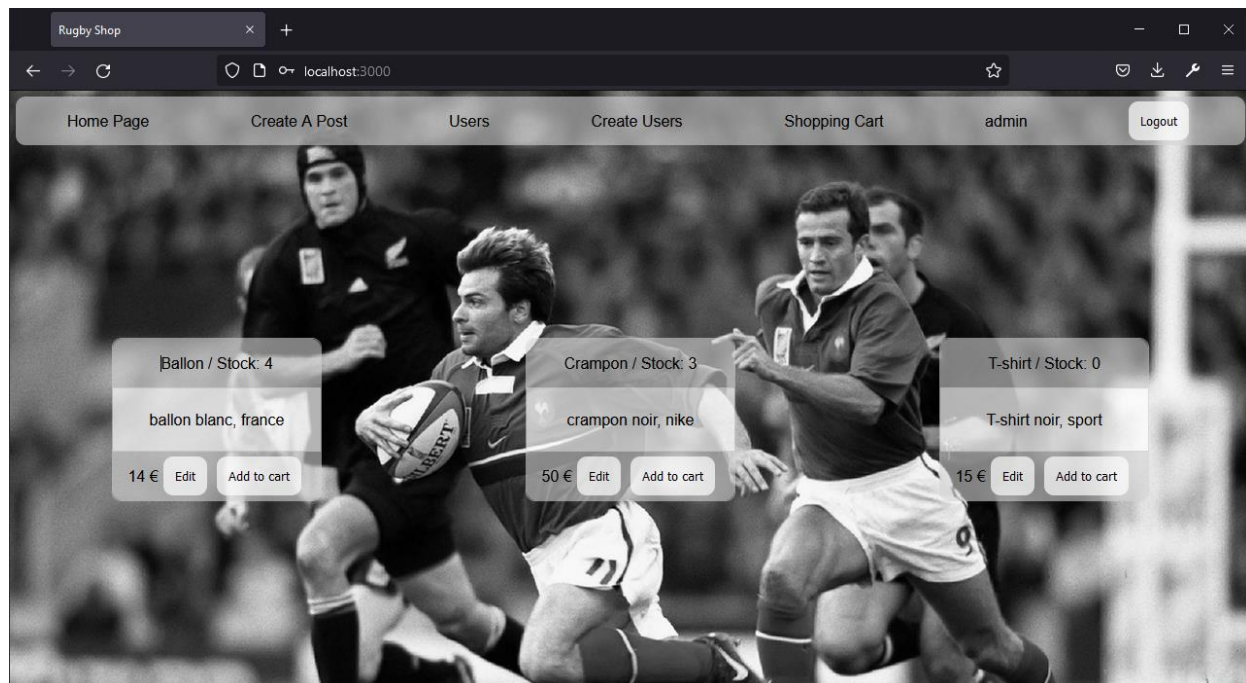
Nous pourrions donc modifier nos informations personnelles et supprimer notre compte en cliquant sur le bouton « Delete ».



Pour se déconnecter cliquer sur le bouton « Logout », nous serons redirigés sur la page de tous les articles disponibles sans pouvoir les ajouter à son panier.

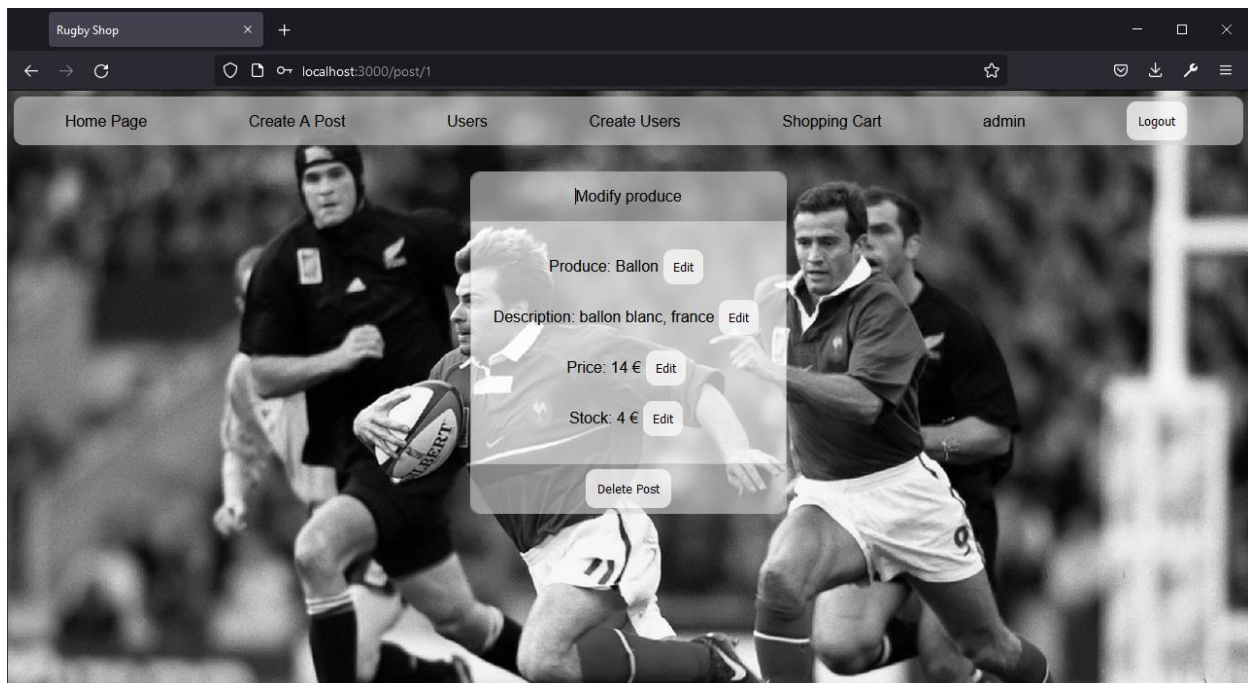


Pour les administrateurs d'autres fonctionnalités sont disponibles.

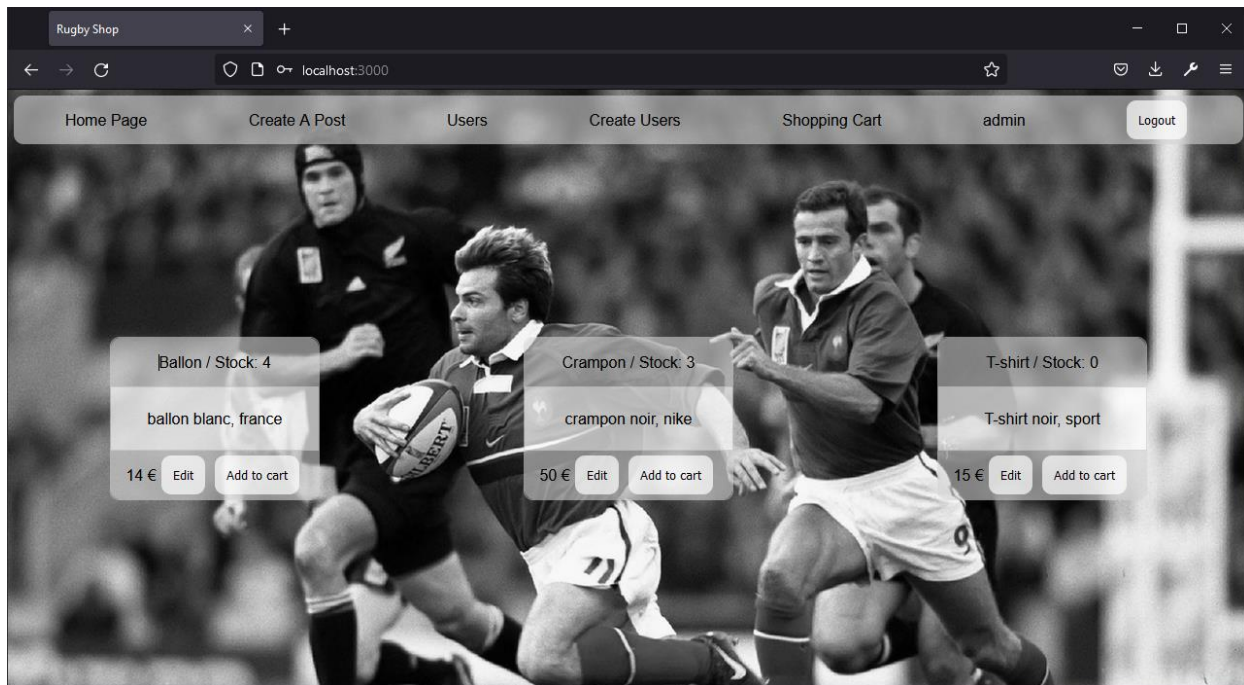


Ils peuvent voir les articles indisponibles dont le stock est de 0 et éditer les articles en cliquant sur le bouton « Edit » de l'article voulu.

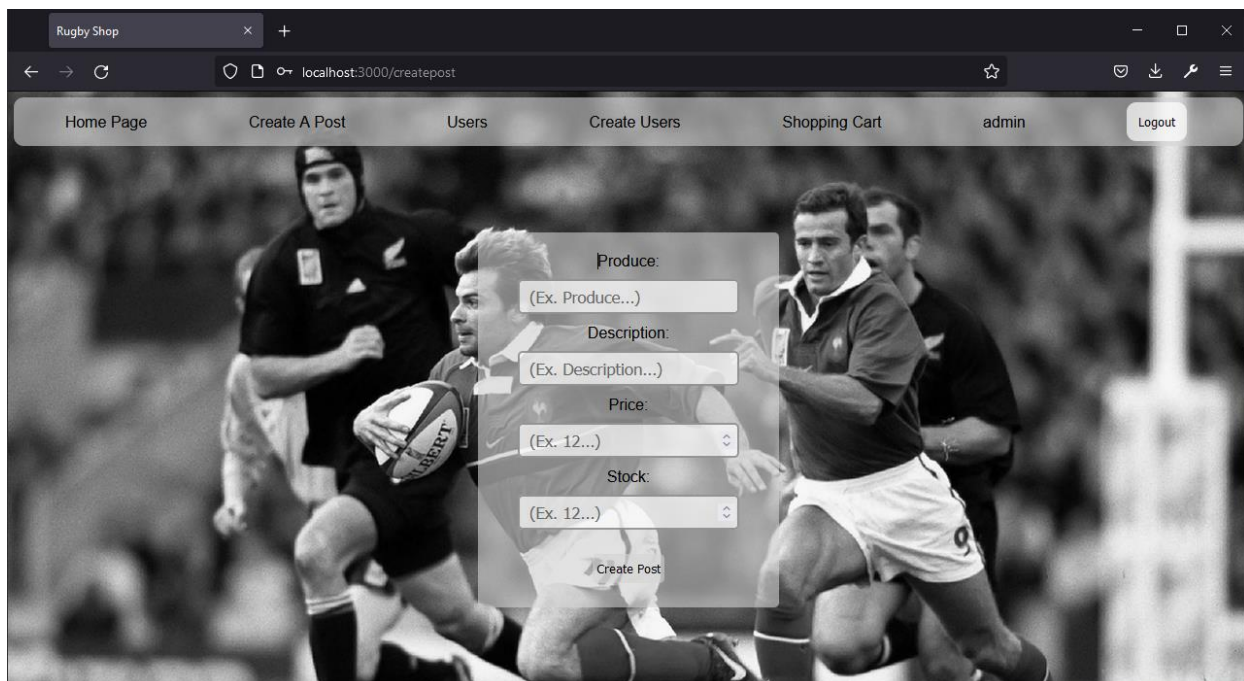
Nous serons redirigés vers la page de modification du produit voulu :



Nous pourrions modifier toutes les informations de l'article et le supprimer en cliquant sur le bouton « Delete ».

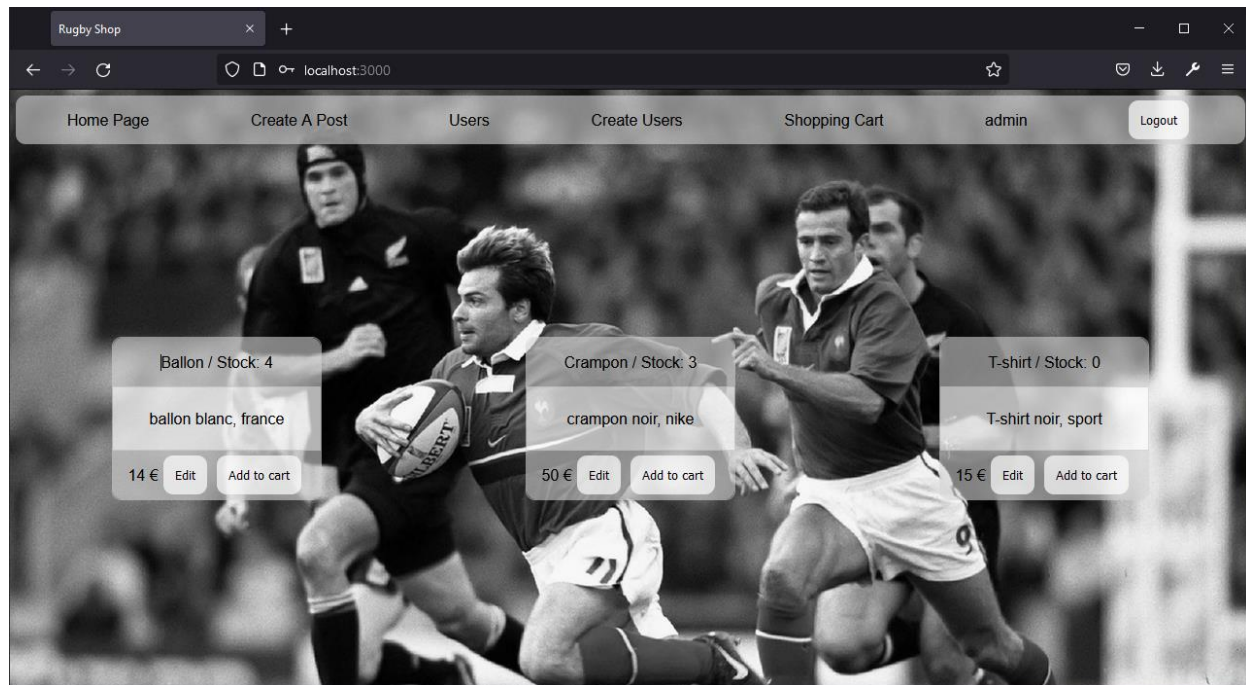


En cliquant sur « Create A Post », nous serons redirigés sur la page de création d'article :

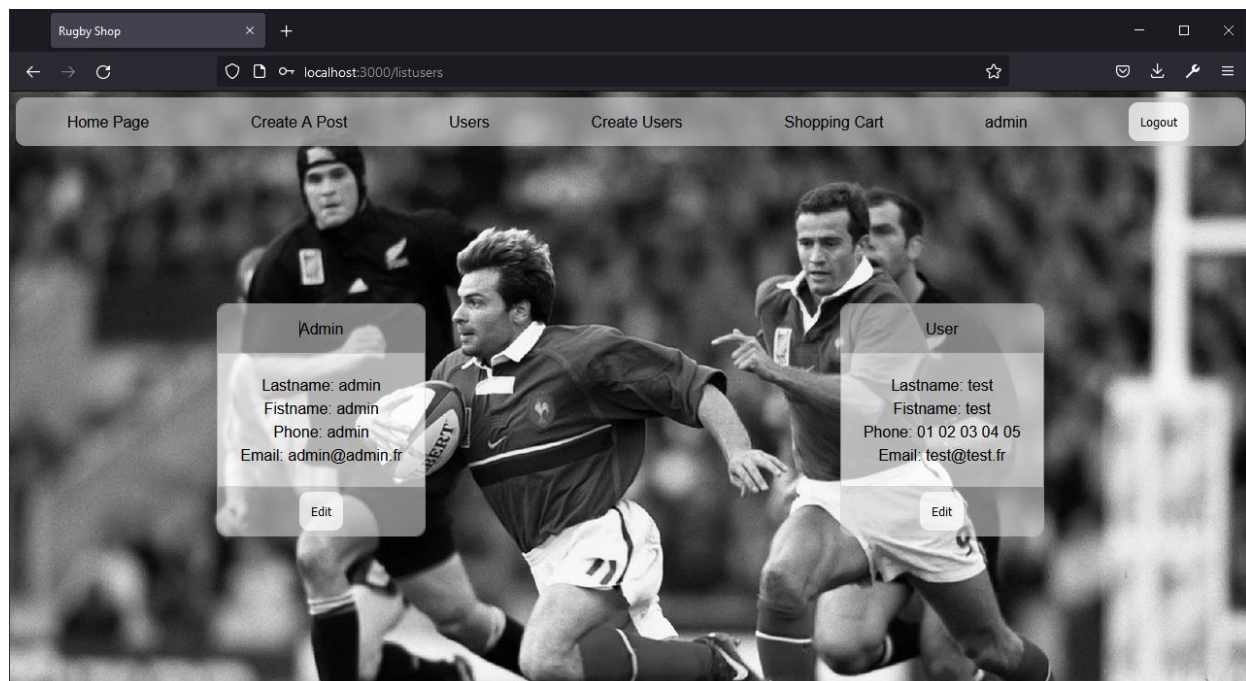


Rentrons les informations du produit puis cliquer sur le bouton « Create Post » pour créer l'article.

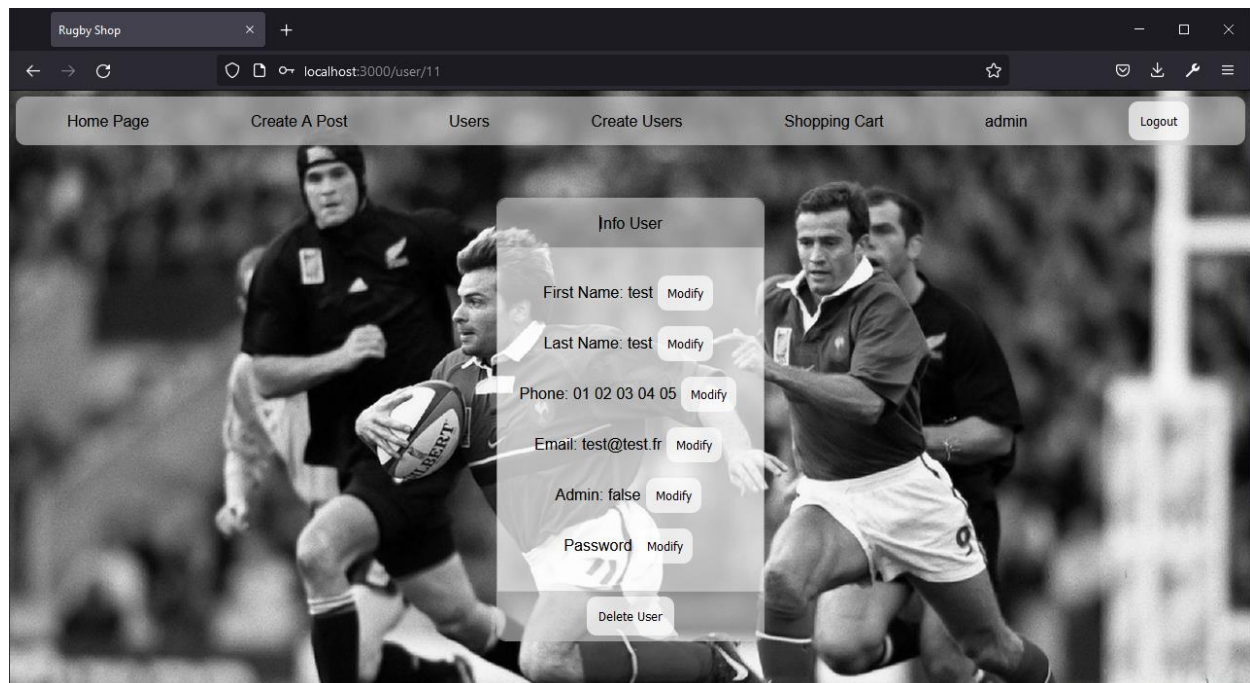
Nous serons redirigés sur la page de tous les articles.



Cliquer sur « Users » pour être redirigé sur la page contenant toutes les informations de tous les utilisateurs :



Pour se rendre sur la page de modification de l'utilisateur voulue, cliquer sur le bouton « Edit ».



Nous pourrions donc modifier toutes les informations de l'utilisateur, aussi donner ou enlever le rôle administrateur de l'utilisateur et supprimer le compte en cliquant sur le bouton « Delete ».

4. Documentation technique

API

Index.js

```
const express = require("express");
const app = express();
const cors = require("cors");

app.use(express.json());
app.use(cors());

const db = require("../models");

// Routers
const postRouter = require("../routes/Posts");
app.use("/posts", postRouter);
const userRouter = require("../routes/Users");
app.use("/auth", userRouter);

db.sequelize.sync().then(() => {
  app.listen(3001, () => {
    console.log("Server running on port 3001");
  });
});
```

Express : C'est une infrastructure d'applications Web Node.js minimaliste et flexible qui fournit un ensemble de fonctionnalités robuste pour les applications Web et mobiles.

Cors : Il sert à partager des ressources entre différents liens avec un seul et même nom de domaine

Sequelize et MySQL2 : Permet d'ajouter une connexion MySQL pour pouvoir récupérer les informations depuis une base de données et afficher ces informations dans l'application sous forme de liste simple.

La constante « db » sert à se connecter à la base de données avec sequelize et MySQL2.

Grâce au fichier JavaScript ci-dessous :

```
'use strict';

const fs = require('fs');
const path = require('path');
const Sequelize = require('sequelize');
const basename = path.basename(__filename);
const env = process.env.NODE_ENV || 'development';
const config = require(__dirname + '/../config/config.json')[env];
const db = {};

let sequelize;
if (config.use_env_variable) {
  sequelize = new Sequelize(process.env[config.use_env_variable], config);
} else {
  sequelize = new Sequelize(config.database, config.username, config.password, config);
}

fs
  .readdirSync(__dirname)
  .filter(file => {
    return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js');
  })
  .forEach(file => {
    const model = require(path.join(__dirname, file))(sequelize, Sequelize.DataTypes);
    db[model.name] = model;
  });

Object.keys(db).forEach(modelName => {
  if (db[modelName].associate) {
    db[modelName].associate(db);
  }
});

db.sequelize = sequelize;
db.Sequelize = Sequelize;

module.exports = db;
```

Et au fichier JSON :

```
"development": {  
  "username": "root",  
  "password": "",  
  "database": "tutorialdb",  
  "host": "localhost",  
  "dialect": "mysql"  
}
```

Username = nom d'utilisateur de la base de données

Password = mot de passe de l'utilisateur

Database = nom de la base de données

Host = l'endroit où s'exécute la base de données

```
const postRouter = require(  
  "./routes/Posts");  
app.use("/posts", postRouter);  
const userRouter = require(  
  "./routes/Users");  
app.use("/auth", userRouter);
```

Permet d'importer les routes où se trouve nos requêtes à envoyer à la base de données.

Voici celle de la table Users partie 1 :

```
const express = require("express");
const router = express.Router();
const { Users } = require("../models");
const bcrypt = require("bcrypt");
const { validateToken } = require("../middleware/AuthMiddleware");
const { sign } = require("jsonwebtoken");

router.get("/", async (req, res) => {
  const listOfUsers = await Users.findAll();
  res.json(listOfUsers);
});

router.get("/byId/:id", async (req, res) => {
  const id = req.params.id;
  const user = await Users.findByPk(id);
  res.json(user);
});

router.post("/", async (req, res) => {
  const { email, password, first_name, last_name, phone } = req.body;
  if (email && password && first_name && last_name && phone) {
    bcrypt.hash(password, 10).then((hash) => {
      Users.create({
        email: email,
        password: hash,
        first_name: first_name,
        last_name: last_name,
        phone: phone,
      });
      res.json("SUCCESS");
    });
  }
});

router.post("/login", async (req, res) => {
  const { email, password } = req.body;
  if (email && password) {
    const user = await Users.findOne({ where: { email: email } });

    if (!user) res.json({ error: "Email Doesn't Exist" });

    const validPassword = await bcrypt.compare(password, user.password);

    if (validPassword === false) {
      res.json({ error: "Wrong email And Password Combination" });
    } else {
      if (validPassword === true) {
        const accessToken = sign(
          { email: user.email, id: user.id, first_name: user.first_name },
          "importantsecret"
        );

        res.json({
          token: accessToken,
          email: email,
          id: user.id,
          first_name: user.first_name,
          admin: user.admin,
        });
      }
    }
  }
});
```

Nous importons le model de la table Users :

```
module.exports = (sequelize, DataTypes) => {  
  const Users = sequelize.define("Users", {  
    email: {  
      type: DataTypes.STRING,  
      allowNull: false,  
    },  
    password: {  
      type: DataTypes.STRING,  
      allowNull: false,  
    },  
    first_name: {  
      type: DataTypes.STRING,  
      allowNull: false,  
    },  
    last_name: {  
      type: DataTypes.STRING,  
      allowNull: false,  
    },  
    phone: {  
      type: DataTypes.STRING,  
      allowNull: false,  
    },  
    admin: {  
      type: DataTypes.BOOLEAN,  
      allowNull: false,  
      defaultValue: false,  
    },  
  });  
  
  return Users;  
};
```

Sequelize.define(« le nom de la table de la base de données », « les champs de cette table avec leurs paramètres »)

Type : `Datatype.STRING` = dire que c'est un champ de caractère

`allowNull : false` = dire que ce champ ne peut pas être null

`defaultValue : false` = mettre la valeur de champ automatiquement false à sa création.

Bcrypt : Sert à mettre en place un hash

validateToken nous sert à vérifier si l'utilisateur est bien connecter avec les bonne information, on l'import du fichier :

```
const { verify } = require("jsonwebtoken");

const validateToken = (req, res, next) => {
  const accessToken = req.header("accessToken");

  if (!accessToken) return res.json({ error: "User not logged in !" });

  try {
    const validToken = verify(accessToken, "importantsecret");
    req.user = validToken;
    if (validToken) {
      return next();
    }
  } catch (err) {
    return res.json({ error: err });
  }
};

module.exports = { validateToken };
```

jsonwebtoken : Permet l'échange sécurisé de jetons (tokens) entre plusieurs parties.

On importe sign de la librairie jsonwebtoken : qui fonctionne comme une signature numérique pour vérifier l'identité

Les routes :

```
router.get("/", async (req, res) => {
  const listOfUsers = await Users.findAll();
  res.json(listOfUsers);
});
```

La route GET sert récupérer des données de la base de données.
Dans ce cas là nous récupérerons toutes les données de la table Users

Axios : C'est un client http basé sur des promesses pour envoyer ou récupérer des données.

La requête est envoyée quand le front-end envoie un GET avec axios dans un chemin de notre domaine :

```
axios
.get("http://localhost:3001/auth/auth", {
  headers: {
    accessToken: sessionStorage.getItem("accessToken"),
  },
})
.then((response) => {
  if (response.data.error) {
    setAuthState({ ...authState, status: false });
  } else {
    setAuthState({
      email: response.data.email,
      id: response.data.id,
      first_name: response.data.first_name,
      admin: response.data.admin,
      status: true,
    });
  }
});
```

Pour vérifier l'identité de l'utilisateur il récupère le Token écrit par bcrypt.
Et il met toute les données de la requête du backend dans la réponse de la promesse.

```
router.post("/", async (req, res) => {
  const { email, password, first_name, last_name, phone } = req.body;
  if (email && password && first_name && last_name && phone) {
    bcrypt.hash(password, 10).then((hash) => {
      Users.create({
        email: email,
        password: hash,
        first_name: first_name,
        last_name: last_name,
        phone: phone,
      });
      res.json("SUCCESS");
    });
  }
});
```

La route POST sert à envoyer des données dans la base de données. Dans ce cas nous envoyons les données lors de l'inscription d'un utilisateur pour que ces données soit stocker dans la base de données.

```
router.put("/firstName", async (req, res) => {
  const { newFirstName, id } = req.body;
  await Users.update({ first_name: newFirstName }, { where: { id: id } });
  res.json(newFirstName);
});
```

La route PUT sert à envoyer de modification dans un endroit cibler dans la base de données. Dans ce cas nous recherchons l'endroit où l'id correspond avec celle envoie par le front-end, et on remplace la valeur de first_name par la nouvelle valeur envoyer par le front-end.


```
router.delete("/:userId", validateToken, async (req, res) => {  
  const userId = req.params.userId;  
  await Users.destroy({  
    where: {  
      id: userId,  
    },  
  });  
});
```

La route DELETE sert à supprimer les données de la base de données en ciblant la ligne à supprimer.

Dans ce cas on cible la ligne à supprimer avec la valeur du front-end ou elle correspond dans la base de donnée pour supprimer toute la ligne.