2024

# Rapport – TP Noté

Lola AUSTIN
Sophie NOIR

# Table des matières

# Introduction

Dans le cadre de notre projet, nous avons conçu une application web. L'objectif principal était de créer une plateforme en ligne affichant une liste de produits et qui permettait de gérer un panier dynamiquement. Ce projet fait partie intégrante de notre cours de développement web, dans le but de renforcer nos compétences en programmation JavaScript, en gestion de bases de données, ainsi qu'en utilisation de Framework et de bibliothèques couramment utilisés.

Nous avons suivi les instructions et utilisé le Framework Vue.js pour la création de l'interface utilisateur. Pour le design et la mise en forme, nous avons utilisé Bootstrap et du CSS pour avoir une interface dynamique. Dans la deuxième partie, nous avons utilisé Express.js ainsi qu'une base de données pour stocker et gérer les produits efficacement.

Dans ce rapport, nous détaillerons les différentes étapes de conceptions et de développement, en définissant les fonctionnalités principales, l'intégration d'une base de données, etc.

Pour finir, ce projet nous a permis de mettre en pratique les concepts théoriques appris en cours et de nous familiariser avec les outils et méthodes de développements avec le langage Javascript.

# Description de l'application

## Fonctionnalités principales

Dans ce projet, il fallait que nous développions une application web pour afficher une liste de produit et gérer un panier d'achats. Les principales fonctionnalités sont :

- **L'affichage de la liste des produits** : chacun des produits sont représenté par leur nom, description et leur prix.
- **La gestion du panier** : les utilisateurs peuvent ajouter un ou plusieurs produits et les supprimer du panier. Le contenu du panier est mis à jour en temps réel sans que l'utilisateur n'aille besoin d'actualiser la page.
- **Affichage du panier** : sur une partie de la page, un récapitulatif du panier sera affiché ainsi que le total.

## Technologies Utilisées

Nous avons utilisé plusieurs technologies durant ce projet :

- **Vue.js** : ce Framework Javascript permet de construit des interface utilisateur. Il est très simple d'utilisation et très flexible.
- **Bootstrap** : c'est une bibliothèque CSS qui permet de créer facilement des designs dynamiques.
- **Express.js** : ce Framework issu de Node.js est utilisé pour gérer les routes et les interactions avec la base de données.
- **Base de données** : nous avons utilisés MySQL pour gérer le stockage des données persistantes comme les produits.

## Composants principaux

Suivant les instructions du sujet, nous avons plusieurs composants principaux :

- **IndexPage.vue** : ce composant nous permet d'afficher la liste des produits et de gérer le panier. Il interagit avec les classes définies dans le fichier *manager.js*. Cet interaction nous permet d'ajouter et de retirer des produits du paniers et de mettre à jour l'affichage en temp réel.
- **Menu.vue** : dans ce fichier, nous retrouvons le code pour la bannière de l'application.
- **manager.js** : Ce fichier contient les classes du projet, comme *Product*, *Stock* ou *Cart*. La classe *Product* représente les produits avec leurs attributs (id, nom, description, prix). La classe *Stock* gérer le nombre de produit disponible. La classe *Cart*, elle, gère les opérations sur le panier : l'ajout, la suppression, le calcul du total, etc.)

# Développement

## Prérequis pour le projet

Pour pouvoir commencer à développer le projet, il faut installer Vue.js et initialiser un nouveau projet. Tout d'abord, il faut se rendre dans le dossier du projet avec le terminal, puis suivre ses étapes et entrer ses instructions dans le cmd :

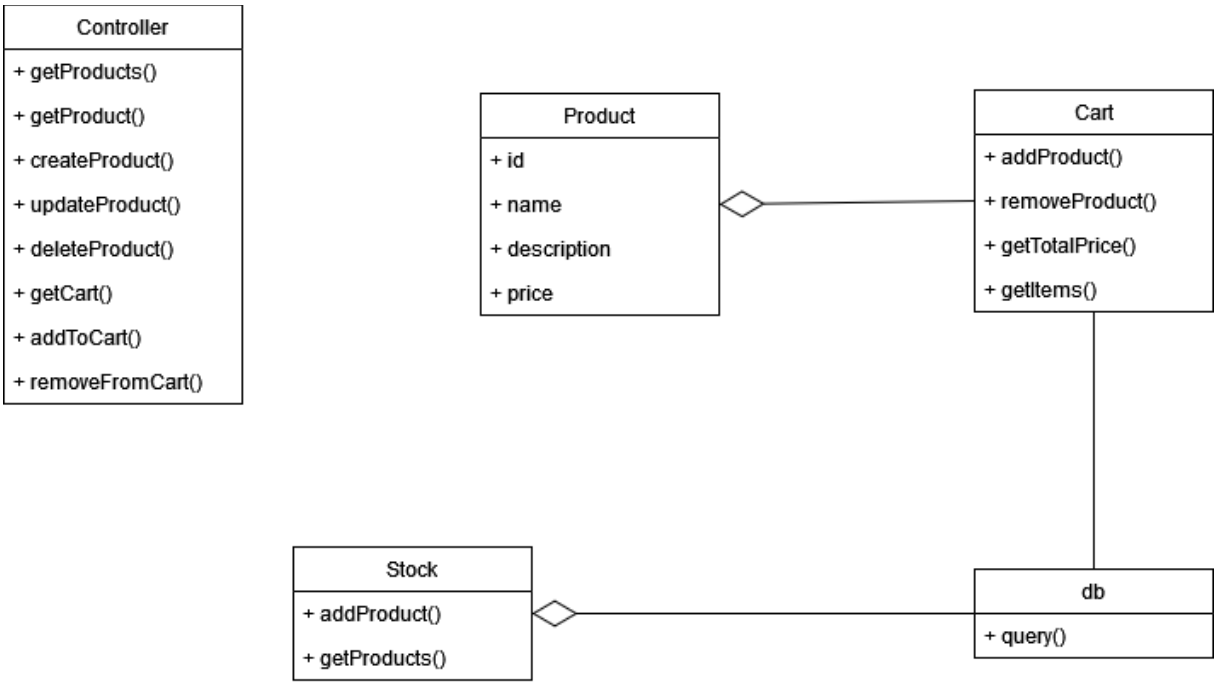
1. Installer Vue.js : *npm install -g @vue/cli*
2. Créer le projet : *vue create webshop*
3. Se rendre dans le dossier du projet : *cd webshop*

Une fois le projet crée, il faut installer Bootstrap ainsi que Bootstrap-vue .

4. Installation de Bootstrap et Bootstrap-vue : n*pm install bootstrap bootstrap-vue*

Une fois toutes ses étapes réalisées, nous pouvons commencer à écrire du code.

## Diagramme de Classe du Projet



## Code des classes principales

Nous définissons les classes *Product*, *Stock* et *Cart* dans le fichier *manager.js*.

### Classe Product :

```
class Product {
    constructor(id = "", name = "", desc = "", price = 0) {
      this.id = id;
      this.name = name;
      this.desc = desc;
      this.price = price;
    }
  }
```

### Classe Stock

```
class Stock {
  constructor() {
    this.list_product = [];
    this.init();
  }

  get_list_product() {
    return this.list_product;
  }

  get_prod_by_id(id) {
    for (var i = 0; i < this.list_product.length; i++) {
      if (this.list_product[i].id == id) {
        return this.list_product[i];
      }
    }
    return null;
  }

  init() {
    this.list_product.push(new Product(1, "Germinal 1", "description germinal 1", 10));
    this.list_product.push(new Product(2, "Germinal 2", "description germinal 2", 20));
    this.list_product.push(new Product(3, "Germinal 3", "description germinal 3", 30));
    this.list_product.push(new Product(3, "Germinal 4", "description germinal 4", 40));
    this.list_product.push(new Product(5, "Germinal 5", "description germinal 5", 50));
    this.list_product.push(new Product(6, "Germinal 6", "description germinal 6", 60));
    this.list_product.push(new Product(7, "Germinal 7", "description germinal 7", 70));
    this.list_product.push(new Product(8, "Germinal 8", "description germinal 8", 80));
    this.list_product.push(new Product(9, "Germinal 9", "description germinal 9", 90));
    this.list_product.push(new Product(10, "Germinal 10", "description germinal 10", 100));
    this.list_product.push(new Product(11, "Germinal 11", "description germinal 11", 110));
  }
}
```

Nous initialisations aussi la liste des produits dans cette classe.

Classe Cart

```
class Cart {
  constructor() {
    this.list_cart = {};
  }

  get_list_cart() {
    return this.list_cart;
  }

  addInCart(id) {
    let elemId = null;
    for (const el in this.list_cart) {
      if (el == id) {
        elemId = id;
      }
    }
    if (elemId !== null) {
      this.addExistedElem(elemId);
    } else {
      this.addNew(id);
    }
  }

  removeFromCart(id) {
    let elemId = null;
    for (const el in this.list_cart) {
      if (el == id) {
        elemId = id;
      }
    }
    if (elemId !== null) {
      if (this.list_cart[id] == 1) {
        delete this.list_cart[id];
      } else {
        this.subExistedElem(id);
      }
    }
  }
```

```
  addNew(id) {
    this.list_cart[id] = 1;
  }

  addExistedElem(id) {
    let val = this.list_cart[id];
    this.list_cart[id] = ++val;
  }

  subExistedElem(id) {
    let val = this.list_cart[id];
    this.list_cart[id] = --val;
  }

  get_nbr_product() {
    let total = 0;
    for (const el in this.list_cart) {
      total = total + this.list_cart[el];
    }
    return total;
  }

  get_total_price(stk) {
    let total = 0;
    for (const el in this.list_cart) {
      let prd = stk.get_prod_by_id(el);
      total = total + (this.list_cart[el] * prd.price);
    }
    return total;
  }
}
```

## Composants Vue.js

Après avoir défini toutes nos classes, nous occupons des composants Vue.js

IndexPage.vue

```vue

<template>
  <div class="IndexPage container">
    <br>
    <div class="row">
      <!-- Product list -->
      <div class="col-md-9">
        <div class="row gx-4 gy-4 row-cols-3">
          <div class="col" v-for="product in list_products" :key="product.id">
            <div class="p-3 border bg-light">
              <h5>{{ product.name }}</h5>
              <p>{{ product.description }}</p>
              <p>$ {{ product.price }} </p>
              <button type="button" class="btn btn-success btn-sm" @click="add_to_cart(product.id)">Ajouter au panier</button>
            </div>
          </div>
        </div>
      </div>

      <!-- Cart -->
      <div class="col-6 col-md-3">
        <div class="col">
          <div class="p-3 border bg-success">
            <h5>Cart</h5>
          </div>
        </div>
        <div class="row gx-4 row-cols-1">
          <div class="col" v-for="item in cartItems" :key="item.cart.id">
            <div class="p-3 border bg-light">
              <h5>{{ item.product.name }}</h5>
              <h6>Quantité: {{ item.quantity }}</h6>
              <button type="button" class="btn btn-success btn-sm" @click="remove_from_cart(item.cart.id)">Supprimer</button>
            </div>
          </div>

```

```
              <!-- show total -->
              <div class="col">
                <div class="p-3 border bg-success">
                  <h5>Total: {{ total_price }} $</h5>
                  <h6>Nombre de produits: {{ total_quantity }}</h6>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </template>

    <script>
    export default {
      name: 'IndexPage',
      data() {
        return {
          list_products: [],
          cartItems: [],
          total_price: 0,
          total_quantity: 0
        }
      },
      created() {
        this.fetchProducts();
        this.fetchCart();
      },
      methods: {
        fetchProducts() {
          fetch('http://localhost:8080/api/products')
            .then(response => response.json())
            .then(data => {
              this.list_products = data.result;
            })
            .catch(error => console.error('Error fetching products:', error));
        },
```

```
        fetchCart() {
          const userId = 1; // Assuming a static user ID for simplicity
          fetch(`http://localhost:8080/api/cart/${userId}`)
            .then(response => response.json())
            .then(data => {
              if (data.result) {
                this.cartItems = this.calculateCartItems(data.result);
                this.updateTotal();
              } else {
                console.error('Cart data format is incorrect', data);
              }
            })
            .catch(error => console.error('Error fetching cart:', error));
        },
        add_to_cart(productId) {
          const userId = 1; // Assuming a static user ID for simplicity
          fetch('http://localhost:8080/api/cart', {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json'
            },
            body: JSON.stringify({ user_id: userId, product_id: productId })
          })
            .then(() => {
              this.fetchCart();
            })
            .catch(error => console.error('Error adding to cart:', error));
        },
        remove_from_cart(cartId) {
          fetch(`http://localhost:8080/api/cart/${cartId}`, {
            method: 'DELETE'
          })
            .then(() => {
              this.fetchCart();
            })
            .catch(error => console.error('Error removing from cart:', error));
        },
```

```
        calculateCartItems(cartData) {
          const cartItemsMap = new Map();
          cartData.forEach(cartItem => {
            const { id, product_id } = cartItem;
            const product = this.list_products.find(p => p.id === product_id);
            if (product) {
              if (cartItemsMap.has(product_id)) {
                cartItemsMap.get(product_id).quantity += 1;
              } else {
                cartItemsMap.set(product_id, { cart: { id }, product, quantity: 1 });
              }
            }
          });
          return Array.from(cartItemsMap.values());
        },
        updateTotal() {
          this.total_price = this.cartItems.reduce((total, item) => total + item.product.price * item.quantity, 0);
          this.total_quantity = this.cartItems.reduce((total, item) => total + item.quantity, 0);
        }
      }
    }
    </script>

    <style scoped>
    .IndexPage {
      margin-top: 20px;
    }
    </style>

```

## Menu.vue

```
<template>
  <nav class="navbar navbar-expand-lg navbar-light bg-success">
    <div class="container">
      <a class="navbar-brand text-white" href="#">Web Shop</a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
          <li class="nav-item">
            <a class="nav-link active text-white" aria-current="page" href="#">Accueil</a>
          </li>
          <li class="nav-item">
            <a class="nav-link text-white" href="#">Produits</a>
          </li>
          <li class="nav-item">
            <a class="nav-link text-white" href="#">Contact</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>
</template>
```

```
<script>
import axios from 'axios';

export default {
  name: 'AppMenu',
  created() {
    axios.get('http://localhost:8080/api/products/')
      .then(response => {
        console.log(response.data);
      })
      .catch(error => {
        console.error('Error fetching menu data:', error);
      });
  }
};
</script>


<style scoped>
.navbar-brand {
  font-weight: bold;
}
</style>
```

## Main.js

```
import Vue from 'vue'
import App from './App.vue'
import BootstrapVue from 'bootstrap-vue'

import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'

Vue.config.productionTip = false
Vue.use(BootstrapVue)

new Vue({
  render: h => h(App),
}).$mount('#app')
```

## App.vue

```vue
<template>
  <div id="app">
    <Menu />
    <IndexPage />
  </div>
</template>

<script>
import Menu from './components/Menu.vue';
import IndexPage from './components/IndexPage.vue';

export default {
  name: 'App',
  components: {
    Menu,
    IndexPage
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

## Gestion des routes

Pour gérer des routes, nous utilisons le Framework Express.

### Route.js

Nous répertorions les routes dans ce fichier. L'application se base dessus pour rediriger l'utilisateur sur les différentes pages et gérer ses différentes actions.

```
const express = require('express');
const router = express.Router();

const {
    getProducts,
    getProduct,
    createProduct,
    updateProduct,
    deleteProduct,
    getCart,
    addToCart,
    removeFromCart
} = require('./controller');

router.get('/products', getProducts);
router.get('/products/:productID', getProduct);
router.post('/products', createProduct);
router.put('/products/:productID', updateProduct);
router.delete('/products/:productID', deleteProduct);

router.get('/cart/:userID', getCart);
router.post('/cart', addToCart);
router.delete('/cart/:cartID', removeFromCart);

module.exports = router;
```

## Gestion de la base de données

Afin de gérer les données via la base de données, nous utilisons des Controller pour gérer les interactions entre le site web et la base de données.

## Connexion à la base de données : bdd.js

```
const mysql = require('mysql2');

const db = mysql.createConnection({
    host: '127.0.0.1',
    user: 'root',
    password: '',
    database: 'webshop'
});

db.connect((err) => {
    if (err) {
        console.log('Database connection failed:', err);
    } else {
        console.log('Connected to MySQL database');
    }
});

module.exports = db;
```

Dans ce fichier, nous retrouvons les modalités de connexion à la base de données. Ici notre base de données est une base MySQL.

## Gestion des produits dans la base de données : product.js

```
const db = require('./bdd');

class Product {
    static getAll(callback) {
        db.query('SELECT * FROM products', (err, results) => {
            if (err) {
                callback(err, null);
            } else {
                callback(null, results);
            }
        });
    }

    static getById(id, callback) {
        db.query('SELECT * FROM products WHERE id = ?', [id], (err, results) => {
            if (err) {
                callback(err, null);
            } else if (results.length === 0) {
                callback(null, null);
            } else {
                callback(null, results[0]);
            }
        });
    }

    static create(productData, callback) {
        db.query('INSERT INTO products SET ?', productData, (err, results) => {
            if (err) {
                callback(err, null);
            } else {
                callback(null, results.insertId);
            }
        });
    }

```

```
36      static update(id, productData, callback) {
37          db.query('UPDATE products SET ? WHERE id = ?', [productData, id], (err, results) => {
38              if (err) {
39                  callback(err, null);
40              } else {
41                  callback(null, results.affectedRows);
42              }
43          });
44      }
45
46      static delete(id, callback) {
47          db.query('DELETE FROM products WHERE id = ?', [id], (err, results) => {
48              if (err) {
49                  callback(err, null);
50              } else {
51                  callback(null, results.affectedRows);
52              }
53          });
54      }
55  }
56
57  module.exports = Product;
```

## Gérer toutes les opérations : controller.js

```
const db = require('./bdd');
const Product = require('./product');

const getProducts = (req, res) => {
    Product.getAll((err, results) => {
        if (err) {
            return res.status(500).json({ msg: err });
        }
        res.status(200).json({ result: results });
    });
};

const getProduct = (req, res) => {
    Product.getById(req.params.productID, (err, result) => {
        if (err) {
            return res.status(500).json({ msg: err });
        } else if (!result) {
            return res.status(404).json({ msg: 'Product not found' });
        }
        res.status(200).json({ result });
    });
};

const createProduct = (req, res) => {
    Product.create(req.body, (err, result) => {
        if (err) {
            return res.status(500).json({ msg: err });
        }
        res.status(201).json({ id: result });
    });
};
```

```
const updateProduct = (req, res) => {
    Product.update(req.params.productID, req.body, (err, result) => {
        if (err) {
            return res.status(500).json({ msg: err });
        } else if (!result) {
            return res.status(404).json({ msg: 'Product not found' });
        }
        res.status(200).json({ msg: 'Product updated' });
    });
};

const deleteProduct = (req, res) => {
    Product.delete(req.params.productID, (err, result) => {
        if (err) {
            return res.status(500).json({ msg: err });
        } else if (!result) {
            return res.status(404).json({ msg: 'Product not found' });
        }
        res.status(200).json({ msg: 'Product deleted' });
    });
};

const getCart = (req, res) => {
    db.query('SELECT * FROM carts WHERE user_id = ?', [req.params.userID], (err, results) => {
        if (err) {
            return res.status(500).json({ msg: err });
        }
        res.status(200).json({ result: results });
    });
};

const addToCart = (req, res) => {
    const { user_id, product_id } = req.body;
    db.query('INSERT INTO carts (user_id, product_id) VALUES (?, ?)', [user_id, product_id], (err, results) => {
        if (err) {
            return res.status(500).json({ msg: err });
        }
        res.status(201).json({ id: results.insertId });
    });
```

```
73
74  const removeFromCart = (req, res) => {
75      db.query('DELETE FROM carts WHERE id = ?', [req.params.cartID], (err, results) => {
76          if (err) {
77              return res.status(500).json({ msg: err });
78          } else if (results.affectedRows === 0) {
79              return res.status(404).json({ msg: 'Item not found' });
80          }
81          res.status(200).json({ msg: 'Item removed from cart' });
82      });
83  };
84
85  module.exports = {
86      getProducts,
87      getProduct,
88      createProduct,
89      updateProduct,
90      deleteProduct,
91      getCart,
92      addToCart,
93      removeFromCart
94  };
```

Dans ce fichier, nous retrouvons des fonctions pour gérer les produits ainsi que la panier dans le site web ainsi que dans la base de données.

### Interactivité et mise à jour en temps réel de la page

Grace au Framework Vue.js, notre site web peut mettre à jour le contenu du panier sans que l'utilisateur n'aille à recharger sa page/

Les interactions avec le panier sont gérées par le fichier *IndexPage.vue* tout en s'assurant que l'interface utilisateur reste synchronisée.

### Déploiement de l'application

Pour pouvoir cloner le site, il faut suivre ses étapes dans le dossier voulu via le terminal.

1. Cloner le dépôt Git : *git clone* <URL>
2. Entrer dans le dossier : *cd webshop*
3. Installer les dépendances : *npm install*

Une fois ses étapes suivies, vous pouvez lancer le site.

4. Lancer l'application : *npm run serve*

## Conclusion

En créant une application web, nous avons pu explorer des technologies comme Vue.js, Bootstrap et Express.js.

Au cours du développement, nous avons réussi à implémenter les principales fonctionnalités comme l'affichage des produits, la gestion du panier et l'interaction avec le panier. L'utilisation de Vue.js a permit de gérer le côté dynamique du site, améliorant l'expérience utilisateur. De plus l'intégration de Bootstrap a facilité la création d'une interface attrayante.

La deuxième partie du projet, axé sur l'utilisation d'une base de données et la gestion des données. Nous avons pu découvrir la gestion des données via Express.js.

Ce projet nous a permis de développer des compétences techniques en développement web.

En conclusion, ce projet a été une expérience enrichissante et formatrice.