

## Plan du code JS (orinoco)

### Index.js

Sur l'index.js donc sur la page d'accueil, on doit être capable d'appeler l'url de l'api qu'on souhaite récupérer (<http://localhost:3000/api/cameras>). C'est pour cela que dans un premier temps on stocke dans une variable cette url pour la garder dans tout notre code.

```
// URL de l'api
const url = "http://localhost:3000/api/cameras";
```

### La fonction getAllCams

Ensuite, on va vouloir créer une fonction pour récupérer tout le contenu des produits... c'est pour cela que la fonction getAllCams est créée, pour pouvoir utiliser la méthode fetch et être sûr qu'on nous retourne toutes les informations avant de les afficher... on stocke la réponse dans une constante pour pouvoir la réutiliser dans la fonction. On y attend du format JSON(format textuel où on retrouve toutes les informations sous forme d'objet similaire au javascript), la propriété async va nous servir pour pouvoir gérer ce code asynchrone, donc d'exécuter le code en même temps que tous les autres et attendre une promesse de réponse avec la propriété await.

```
// Récupère toutes les caméras
const getAllCams = async (url) => {
  const response = await fetch(url);
  return await response.json();
};
```

### La fonction displayProducts

Dès qu'on récupère notre api puis appeler les informations sous format json. On va vouloir afficher tous les produits dans la page d'accueil, c'est pour cela qu'on intègre dans une variable products la réponse de getAllCams, et on met en œuvre les informations qui nous intéressent c'est à dire... le nom de la caméra, l'image, le prix et l'ID pour le faire ressortir après pour le bon de commande... on les met dans les paramètres de notre fonction products.forEach => renderProduct pour pouvoir créer les balises puis le html avec ces paramètres, c'est là que la fonction renderProduct va servir

```
// Affiche tous les produits
const displayProducts = async () => {
  const products = await getAllCams(url);
  products.forEach((product) => {
    renderProduct(product.name, product._id, product.imageUrl, product.price);
  });
};
```

## La fonction renderProduct

Après vouloir afficher tous les produits, on va pouvoir créer les balises html enfin d'intégrer au mieux nos informations récupérés... cette fonction prend 4 paramètres comme signalé dans la fonction displayProducts... on veut le nom, l'id, l'image et le prix... avec toutes ces informations on récupère dans une variable l'élément html avec pour ID #products qui mettra tous nos produits dans notre page d'accueil. Puis, on crée des articles et dans ces articles on appelle la propriété innerHTML pour pouvoir créer des balises et intégrer tous les produits avec leur informations. Les articles on les insère en tant qu'enfant à l'ID html #products.

Tout cela fait, on oublie pas d'appeler la fonction displayProducts() pour que tout ce code fonctionne

```
// Fourni l'affichage d'un produit
function renderProduct(productName, productId, productImg, productPrice) {
  const products = document.querySelector("#products"); // Récupère la div qui contiendra les différents articles
  const article = document.createElement("article");
  article.innerHTML = `
  <button class="product-link" type="button"><a href="product.html?id=${productId}">Voir le produit</a></button>
  <p class="product-title">${productName}</p>
  <p class="price">${productPrice / 1000}</p>
  `;
  products.appendChild(article);
}

displayProducts();
```

## Product.js

Dès que la page d'accueil est terminée, on peut s'attaquer à la page produit pour la personnalisation des lentilles et pouvoir l'ajouter au panier par la suite.

Tout d'abord on récupère encore une fois l'url de l'api pour l'avoir dans tout notre code.

Ensuite, on récupère les paramètres de l'url, l'id et l'article dans lequel on veut afficher le produit sélectionné.

```
// URL de l'api
const url = "http://localhost:3000/api/cameras/";
// Récupère les paramètres de l'url
const params = new URLSearchParams(window.location.search);
const id = params.get("id");

const article = document.querySelector("article");
```

### La fonction displayProduct

On recrée la fonction displayProduct pour pouvoir afficher l'article, l'image, on stocke la réponse de la caméra dans une constante (data), on appelle dans cette fonction renderCams, customizeYourCamera et addToCart, qui sera créé juste après avec les propriétés attendus. Tout ceci se fera en asynchrone, en parallèle avec l'exécution d'autres codes JS, pour que le serveur charge ces données et récupère les informations.

```
// Affiche le produit
const displayProduct = async () => {
  const data = await getOneCams(url, id);
  renderCams(data);
  customizeYourCamera(article, data.lenses);
  addToCart(article, data);
};
```

### La fonction getOneCams

Cette fonction, on va attendre d'elle une réponse et nous retourner le résultat de la réponse pour récupérer la caméra sélectionnée, avec la méthode fetch encore une fois on attend du json, puis cette fonction sera appelée dans la fonction displayProduct pour être affiché par la suite.

```
// Récupère une caméra
const getOneCams = async (productUrl, productId) => {
  const response = await fetch(productUrl + productId);
  return await response.json();
};
```

### La fonction renderCams

Cette fonction prend en paramètre (productData), pour signaler le fait de la création de la structure html pour ce produit dans l'article qui lui est dédiée.

Dans l'article on crée la structure du innerHTML en mettant son image, son nom, une div qui contiendra un h2, 2 paragraphes en mettant le prix puis le nom de la caméra, et la description.

```
// Fourni l'affichage selon les données du produit
const renderCams = (productData) => {
  article.innerHTML = `
    <div class="product">
      
      <div class="product-information">
        <h2 class="product-title">${productData.name}</h2>
        <p class="price">${productData.price / 1000}</p>
        <p class="description">Toutes nos caméras sont de qualité, et vous garantissent une longévité extrême !</p>
      </div>
    </div>`;
};
```

### La fonction customizeYourCamera

Dans cette fonction on s'occupe de créer les options pour les lentilles à choisir avec 2 paramètres (parentElt, productLenses). Tout d'abord on crée dans 2 variables les balises suivantes : label et select.

Dans le label on rajoute des attributs comme : lenses-list (la liste des lentilles pour créer une liste déroulante de choix de lentilles).

Ensuite, on rajoute du texte en décrivant ce qu'il y aura dans ces options de listes (Lentilles disponibles : )

Et dans select on récupère la lenses-list pour faire la sélection. On ajoute par la suite label et select comme enfant à parentElt. Dans productLenses on crée des options pour chaque choix de lentilles dans une balise « option » ; avec tout cela, pour finir on souhaite écouter un événement quand l'utilisateur choisi une option, on garde ce choix avec la propriété « target.value », et ensuite on la sauvegarde dans la console.

```
// Personnalise le produit
const customizeYourCamera = (parentElt, productLenses) => {
  // Crée liste déroulante
  const label = document.createElement("label");
  const select = document.createElement("select");

  label.setAttribute("for", "lenses-list");
  label.textContent = "Lentilles disponibles : ";
  select.id = "lenses-list";

  parentElt.appendChild(label);
  parentElt.appendChild(select);
  // Crée une balise option pour chaque lentille
  productLenses.forEach((productLense) => {
    const option = document.createElement("option");
    option.value = productLense;
    option.textContent = productLense.toUpperCase();
    select.appendChild(option);
  });
  // Récupère la lentille choisie dans la console
  select.addEventListener("change", (e) => {
    lenseChosen = e.target.value.toLowerCase();
    console.log(lenseChosen);
  });
};
```

### La fonction addToCart

Il ne reste plus qu'à ajouter le produit au panier, après avoir récupéré le produit, l'afficher et personnaliser le produit.

Pour faire cela, on va faire passer en paramètres le parentElt et le productData auparavant évoqué. Pour par la suite prendre en compte ces informations dans le panier.

Pour se faire, on crée des balises, button et div afin de pouvoir cliquer sur « ajouter au panier », on ajoute ces 2 balises comme enfant à parentElt qui a été créé avant. Ensuite, ce qu'on a envie de faire c'est de sauvegarder le choix de l'utilisateur dans le navigateur pour cela on va faire un tableau réunissant les propriétés qu'on souhaite garder pour le localStorage (l'id, le nom, l'image et le prix)... on va écouter un événement quand on clique sur « ajouter au panier » au bouton, on va vouloir transmettre au localStorage toutes les informations choisies par l'utilisateur en les gardant et utilisant le localStorage. Quand tout ceci sera fait, l'important c'est de signaler que le produit a été bien ajouté au panier.

```
// Ajoute le produit au panier
const addToCart = (parentElt, productData) => {
  // Crée le bouton d'envoi du produit
  const btn = document.createElement("button");
  const div = document.createElement("div");
  btn.textContent = "Ajouter au panier";
  div.classList.add("add-to-cart");
  parentElt.appendChild(div);
  parentElt.appendChild(btn);

  // Assigne valeur à envoyer à localStorage
  const product = [
    productData._id,
    productData.name,
    productData.price,
    productData.imageUrl,
  ];

  // Envoie valeur à localStorage après un clique
  btn.addEventListener("click", () => {
    localStorage.setItem(productData.name, JSON.stringify(product));
    btn.classList.add("invisible");
    div.textContent = "Le produit a été ajouté au panier !";
  });
};

displayProduct();
```

## Cart.js

Après avoir créer les 2 fichiers Js de la page d'accueil et de la personnalisation du produit, bien évidemment, on s'attaque à la page « panier ».

Tout d'abord on récupère la section du panier #cart, on récupère le h3 créé pour afficher le prix total .cart-total, puis le formulaire .form pour par la suite donner les informations liés aux coordonnées de l'utilisateur.

On stocke le prix initial à 0 pour pouvoir l'utiliser par la suite, et on crée un objet cartInformation pour stocker les contacts puis les informations de produits

```
const cart = document.querySelector("#cart"); // Récupère la section du panier
const cartTotal = document.getElementById("cart-total"); //Récupère le h3 pour le prix total
const form = document.querySelector("form"); // Récupère le formulaire

const cartInformation = {
  contact: {},
  products: [],
};
/* Stock le prix total */
let totalPrice = 0;
```

## La fonction displayCart

On va faire en sorte que cette fonction représente du code asynchrone en parallèle avec les autres exécutions de code. L'exécution du code est la suivante : il y a une condition IF/else qui met en œuvre le fait que si il y a un article sélectionné on récupère son nom, ces informations, le prix et reste dans le panier jusqu'à suppression du produit, sinon on affiche que le panier est vide et on rend invisible le formulaire. Puis on récupère l'élément dans le localStorage.

```
// Affiche le/les produit(s) du panier.
const displayCart = async () => {
  if (localStorage.length > 0) {
    for (let i = 0; i < localStorage.length; i++) {
      // Pour chaque article du panier
      const product = await getCart(i); // Récupère les informations du produit
      const camId = product[0]; // Stocke l'id du produit
      const camName = product[1]; // Stocke le nom du produit
      const camPrice = product[2] / 100; // Stocke le prix du produit
      const camImg = product[3]; // Stocke l'image du produit
      cartInformation.products.push(camId); // Envoie l'id du produit au tableau products de cartInformation

      renderCart(camName, camPrice, camImg); // Fourni l'affichage du/des produits du panier

      const remove = document.querySelectorAll(".remove")[i];
      const article = document.querySelectorAll("article")[i];

      deleteCart(remove, article, camName);
    }
  } else {
    cart.textContent = "Votre panier est vide.";
    form.classList.add("invisible");
  }
};

// Récupère élément dans localStorage
const getCart = async (index) => {
  return await JSON.parse(localStorage.getItem(localStorage.key(index)));
};
```

## La fonction renderCart

Cette fonction met en œuvre la création de l'article dans le panier, par le innerHTML, on crée les balises img, div puis paragraphe pour ainsi insérer le nom, le prix et l'image du produit sélectionné... on remarque qu'on ajoute le bouton « supprimer » pour pouvoir supprimer le produit du panier, on crée le contenu du prix total en additionnant tous les produits du panier, et automatiquement calculer le prix. Puis, quand un utilisateur clique sur « supprimer » on fait en parallèle une suppression dans le localStorage aussi en même temps, et actualise automatiquement la page.

```
// Fourni l'affichage du/des produits du panier
const renderCart = (productName, productPrice, imgUrl) => {
  /* Affiche article(s) du panier */
  const article = document.createElement("article");
  article.innerHTML = `
    
    <div class="product-information">
      <p class="product-title">${productName}</p>
      <p class="price">${productPrice}</p>
    </div>
    <p class="remove ">supprimer</p>`;
  cart.insertBefore(article, cartTotal); // Insère article avant cartTotal

  totalPrice += productPrice; /* Implémente prix */
  cartTotal.textContent = `Total : ${totalPrice}€`; /* Affiche le prix total */
};

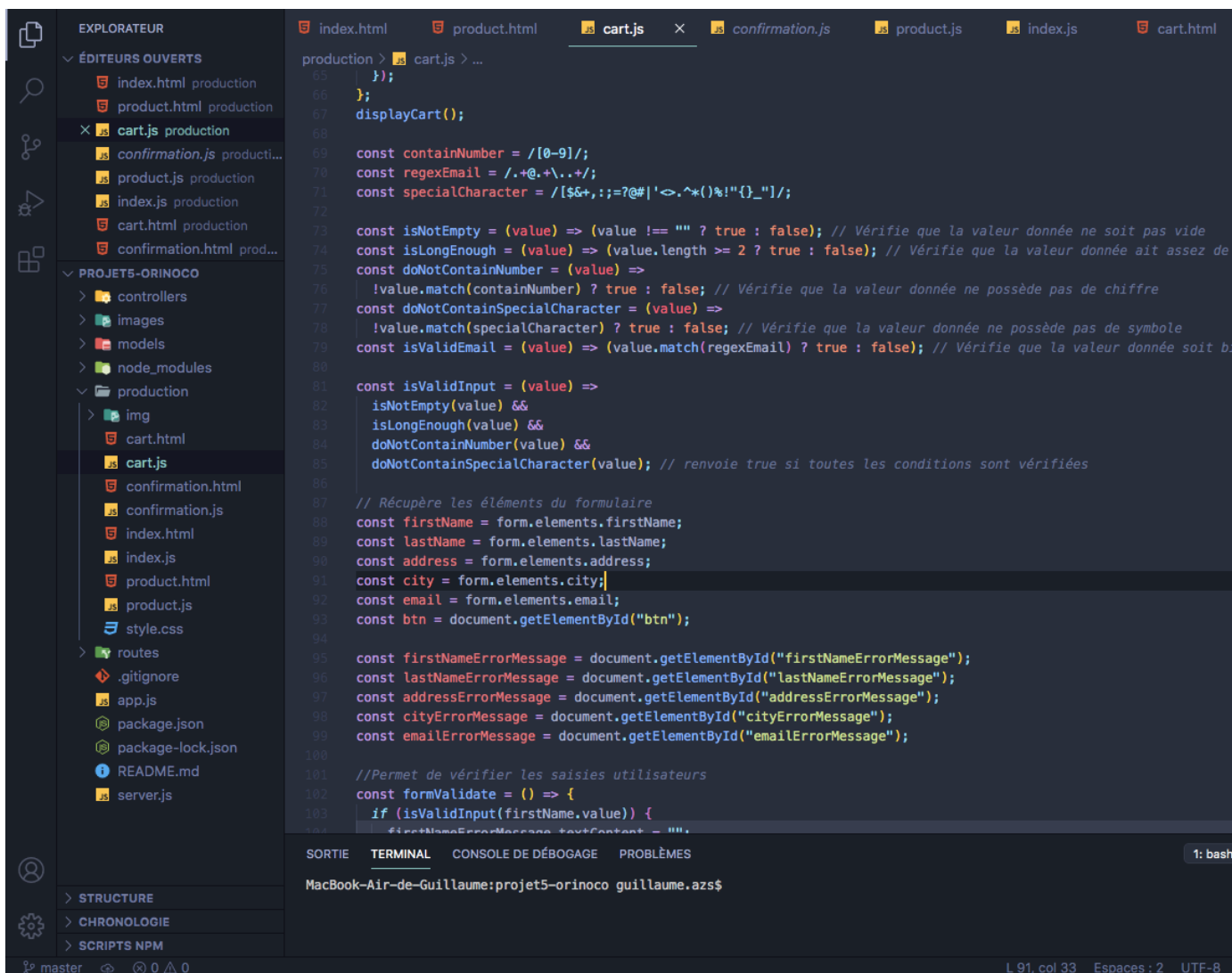
/* Supprime élément du panier sur un clique*/
const deleteCart = (removeElt, container, productName) => {
  removeElt.addEventListener("click", async () => {
    /* Gestionnaire d'évènement sur clique */
    await localStorage.removeItem(
      productName
    ); /* Supprime item du localStorage */
    container.remove(); /* Supprime item du DOM */
    location.reload(true); /* Actualise la page dynamiquement */
  });
};
```



## Les constantes pour vérification du formulaire

Avant tout les constantes sont créés pour vérifier les caractères spéciaux du formulaire, que ce soit pour l'adresse, prénom etc... afin de s'assurer que l'utilisateur rentre bien ce qu'on souhaite.

On vérifie si le champ n'est pas vide. Que la valeur ait assez de caractères... la valeur ne possède pas de chiffres, pas de symboles, que ce soit bien en format mail pour les mails. On crée les constantes `firstname`, `lastname`, `address`, `city` et `email` pour par la suite les récupérer. Et ensuite les même constantes pour gérer les erreurs de saisis par l'utilisateur.



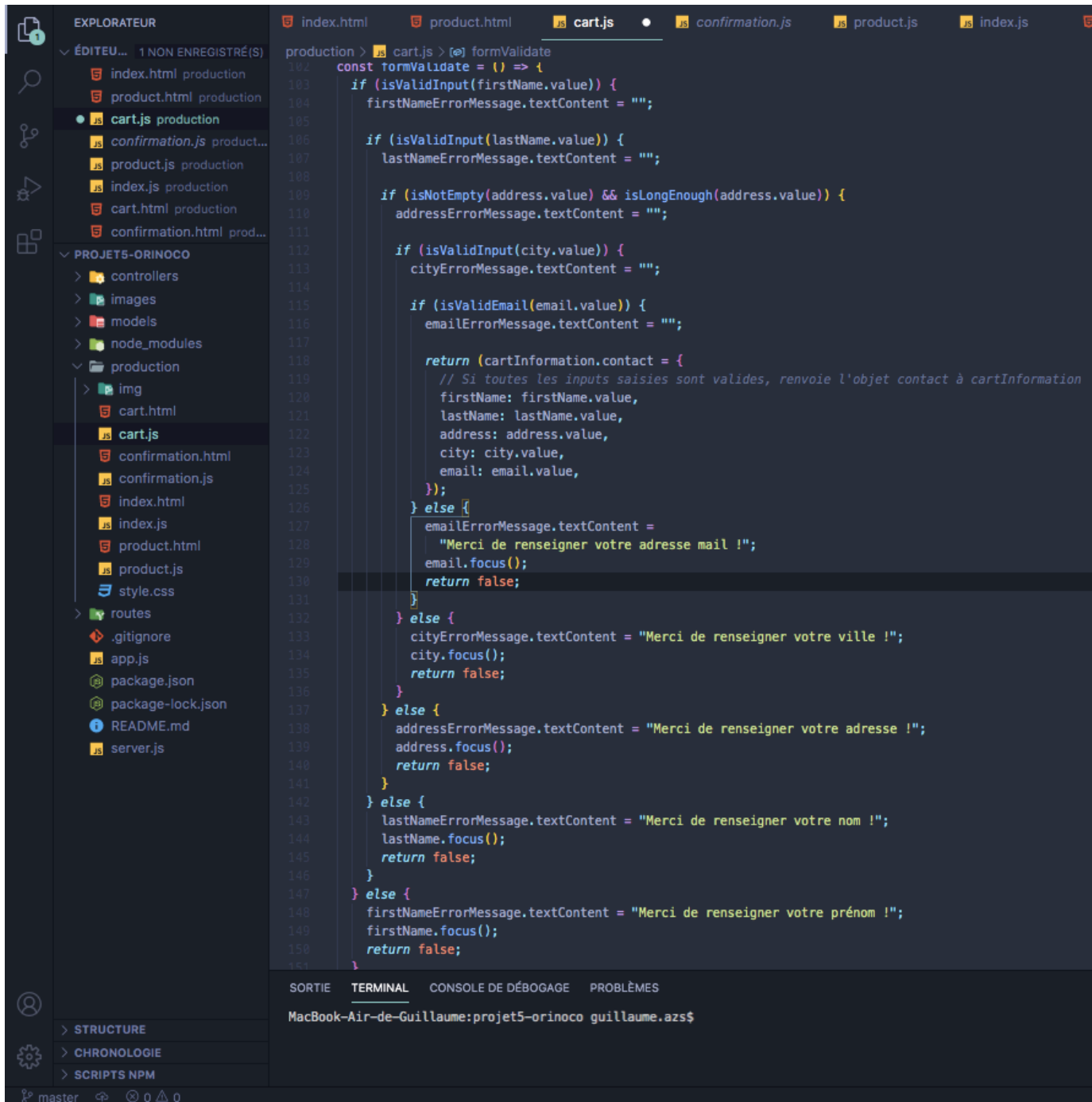
```
production > cart.js > ...
65 | });
66 | };
67 | displayCart();
68 |
69 | const containNumber = /[0-9]/;
70 | const regexEmail = /.+@.+\...+/;
71 | const specialCharacter = /[$&+,:;=?@#|'<>.^*()!'"{}_~\|/;
72 |
73 | const isEmpty = (value) => (value !== "" ? true : false); // Vérifie que la valeur donnée ne soit pas vide
74 | const isLongEnough = (value) => (value.length >= 2 ? true : false); // Vérifie que la valeur donnée ait assez de
75 | const doNotContainNumber = (value) =>
76 |   !value.match(containNumber) ? true : false; // Vérifie que la valeur donnée ne possède pas de chiffre
77 | const doNotContainSpecialCharacter = (value) =>
78 |   !value.match(specialCharacter) ? true : false; // Vérifie que la valeur donnée ne possède pas de symbole
79 | const isValidEmail = (value) => (value.match(regexEmail) ? true : false); // Vérifie que la valeur donnée soit b
80 |
81 | const isValidInput = (value) =>
82 |   isEmpty(value) &&
83 |   isLongEnough(value) &&
84 |   doNotContainNumber(value) &&
85 |   doNotContainSpecialCharacter(value); // renvoie true si toutes les conditions sont vérifiées
86 |
87 | // Récupère les éléments du formulaire
88 | const firstName = form.elements.firstName;
89 | const lastName = form.elements.lastName;
90 | const address = form.elements.address;
91 | const city = form.elements.city;
92 | const email = form.elements.email;
93 | const btn = document.getElementById("btn");
94 |
95 | const firstNameErrorMessage = document.getElementById("firstNameErrorMessage");
96 | const lastNameErrorMessage = document.getElementById("lastNameErrorMessage");
97 | const addressErrorMessage = document.getElementById("addressErrorMessage");
98 | const cityErrorMessage = document.getElementById("cityErrorMessage");
99 | const emailErrorMessage = document.getElementById("emailErrorMessage");
100 |
101 | //Permet de vérifier les saisies utilisateurs
102 | const formValidate = () => {
103 |   if (isValidInput(firstName.value)) {
104 |     firstNameErrorMessage.textContent = "";
105 |   }
106 |   if (isValidInput(lastName.value)) {
107 |     lastNameErrorMessage.textContent = "";
108 |   }
109 |   if (isValidInput(address.value)) {
110 |     addressErrorMessage.textContent = "";
111 |   }
112 |   if (isValidInput(city.value)) {
113 |     cityErrorMessage.textContent = "";
114 |   }
115 |   if (isValidInput(email.value)) {
116 |     emailErrorMessage.textContent = "";
117 |   }
118 |   btn.disabled = false;
119 | }
120 |
121 | // Vérifie si le bouton est cliqué
122 | btn.addEventListener("click", formValidate);
```

MacBook-Air-de-Guillaume:projet5-orinoco guillaume.azs\$



## La fonction formValide

On crée une condition, si l'utilisateur rentre bien un prénom, un nom, une adresse correcte, une ville et un email correcte alors les informations sont envoyées par l'objet cartInformation, sinon on affiche des messages d'erreurs à chaque fois que les règles ne sont pas respectés... et par la suite envoie une api par la méthode fetch avec post dans l'application.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'production', 'controllers', 'images', 'models', 'node\_modules', 'production', 'routes', and files like 'index.html', 'product.html', 'cart.js', 'confirmation.js', 'product.js', 'index.js', 'cart.html', 'confirmation.html', 'confirmation.js', 'index.html', 'index.js', 'product.html', 'product.js', 'style.css', 'routes', '.gitignore', 'app.js', 'package.json', 'package-lock.json', 'README.md', 'server.js'. The code editor shows the 'cart.js' file with the following code:

```
production > cart.js > formValidate
102 const formValidate = () => {
103   if (isValidInput(firstName.value)) {
104     firstNameErrorMessage.textContent = "";
105
106   if (isValidInput(lastName.value)) {
107     lastNameErrorMessage.textContent = "";
108
109   if (isNotEmpty(address.value) && isLongEnough(address.value)) {
110     addressErrorMessage.textContent = "";
111
112   if (isValidInput(city.value)) {
113     cityErrorMessage.textContent = "";
114
115   if (isValidEmail(email.value)) {
116     emailErrorMessage.textContent = "";
117
118   return {cartInformation.contact = {
119     // Si toutes les inputs saisies sont valides, renvoie l'objet contact à cartInformation
120     firstName: firstName.value,
121     lastName: lastName.value,
122     address: address.value,
123     city: city.value,
124     email: email.value,
125   }};
126 } else {
127   emailErrorMessage.textContent =
128     "Merci de renseigner votre adresse mail !";
129   email.focus();
130   return false;
131 }
132 } else {
133   cityErrorMessage.textContent = "Merci de renseigner votre ville !";
134   city.focus();
135   return false;
136 }
137 } else {
138   addressErrorMessage.textContent = "Merci de renseigner votre adresse !";
139   address.focus();
140   return false;
141 }
142 } else {
143   lastNameErrorMessage.textContent = "Merci de renseigner votre nom !";
144   lastName.focus();
145   return false;
146 }
147 } else {
148   firstNameErrorMessage.textContent = "Merci de renseigner votre prénom !";
149   firstName.focus();
150   return false;
151 }
```

The bottom of the editor shows the 'TERMINAL' tab with the command prompt: `MacBook-Air-de-Guillaume:projet5-orinoco guillaume.azs$`

## Envoie des données et gérer l'événement

Dès que l'utilisateur clique sur la validation du formulaire on envoie les données saisis au serveur, et une redirection vers une page de confirmation s'affiche.



## Confirmation.js

La dernière page s'affiche quand le panier et le formulaire a été validé. On met en œuvre des constantes dans lesquelles on stocke le prix total, le prénom qu'on récupère grâce au formulaire, l'id de la commande qu'on récupère aussi puis on remercie pour la commande en ajoutant du `textContent` dans les classes créées dans le html.

