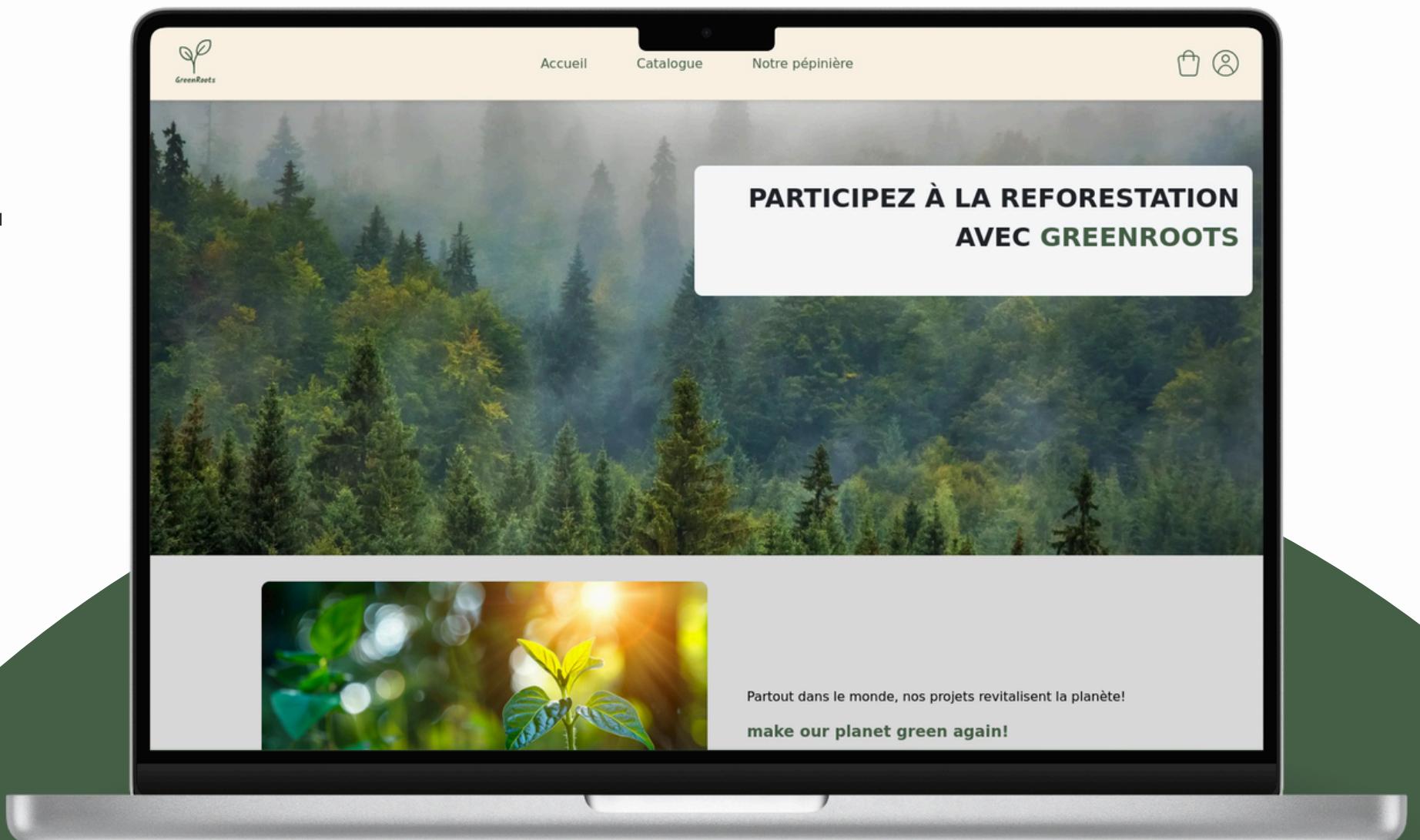




Présentation au titre **Concepteur Développeur d'Applications**

Chloé Garcia



Mon parcours

- diplôme d'Etat d'architecte et HMONP
- **architecte chef de projet** en agence durant une dizaine d'années
- **professeur des écoles**
- **graphiste / web designer** freelance

Le métier de développeur web me permet de:

- réunir mes différentes compétences, à la fois techniques et créatives
- répondre à des besoins concrets
- proposer des outils sur mesure

... Et finalement, il y a beaucoup de ressemblance entre le métier d'architecte et de développeur



Sommaire

▶ Présentation du projet	01
▶ Contexte technique	02
▶ Méthodologie de projet	03
▶ Outils utilisés	04
▶ Conception	05
▶ Tunnel d'achat	06
▶ Sécurité	07
▶ Tests	08
▶ Déploiement	09
▶ RGPD/ SEO/ Accessibilité	10
▶ Conclusion	11



OBJECTIFS

permettre aux utilisateurs de
découvrir l'entreprise GreenRoots

découvrir, via un **catalogue**, les arbres
liés aux projets de reforestation

contribuer financièrement via l'**achat**
d'arbres à la reforestation

1

2

3

PRÉSENTATION GREENROOTS

CONTEXTE:

plateforme e-commerce dédiée à la vente d'arbres dans le cadre de projets de reforestation

BESOIN:

participer à la reforestation par l'achat d'arbres dans des écosystèmes adaptés

SOLUTION:

GreenRoots met à disposition les arbres et les zones de plantation.

MVP

FONCTIONNALITÉS DE BASE

CÔTÉ CLIENT

- 
- 01 PRÉSENTATION ET DÉCOUVERTE
 - informer les visiteurs sur la mission de GreenRoots
 - présenter l'impact social et environnemental
 - 02 CATALOGUE ET ACHAT
 - consulter le catalogue
 - consulter la fiche d'un arbre
 - acheter un ou plusieurs arbres
 - 03 GESTION DE COMPTE
 - création d'un compte utilisateur
 - connexion et déconnexion
 - suivi de commandes
 - 04 SUPPORT
 - contacter l'équipe de GreenRoots par email

MVP

FONCTIONNALITÉS DE BASE

CÔTÉ ADMINISTRATEUR:
GESTION DU CATALOGUE

- 
- 01 Création de nouvelle fiches d'arbres
 - 02 Edition des arbres existants
 - 03 Suppression d'arbres du catalogue

EVOLUTIONS POSSIBLES POST MVP

TRACABILITÉ SUIVI

- suivi des arbres achetés
- carte interactive
- notifications

ACHAT

- achat en lots
- réductions
- événements et campagnes

GESTION UTILISATEURS

- gestion des comptes utilisateurs
- analyse du comportement utilisateurs

FONCTIONNALITÉS AVANCÉES

- version multilingue
- gamification

ARCHITECTURE API REST + MPA

MVC (Models, View, Controller)

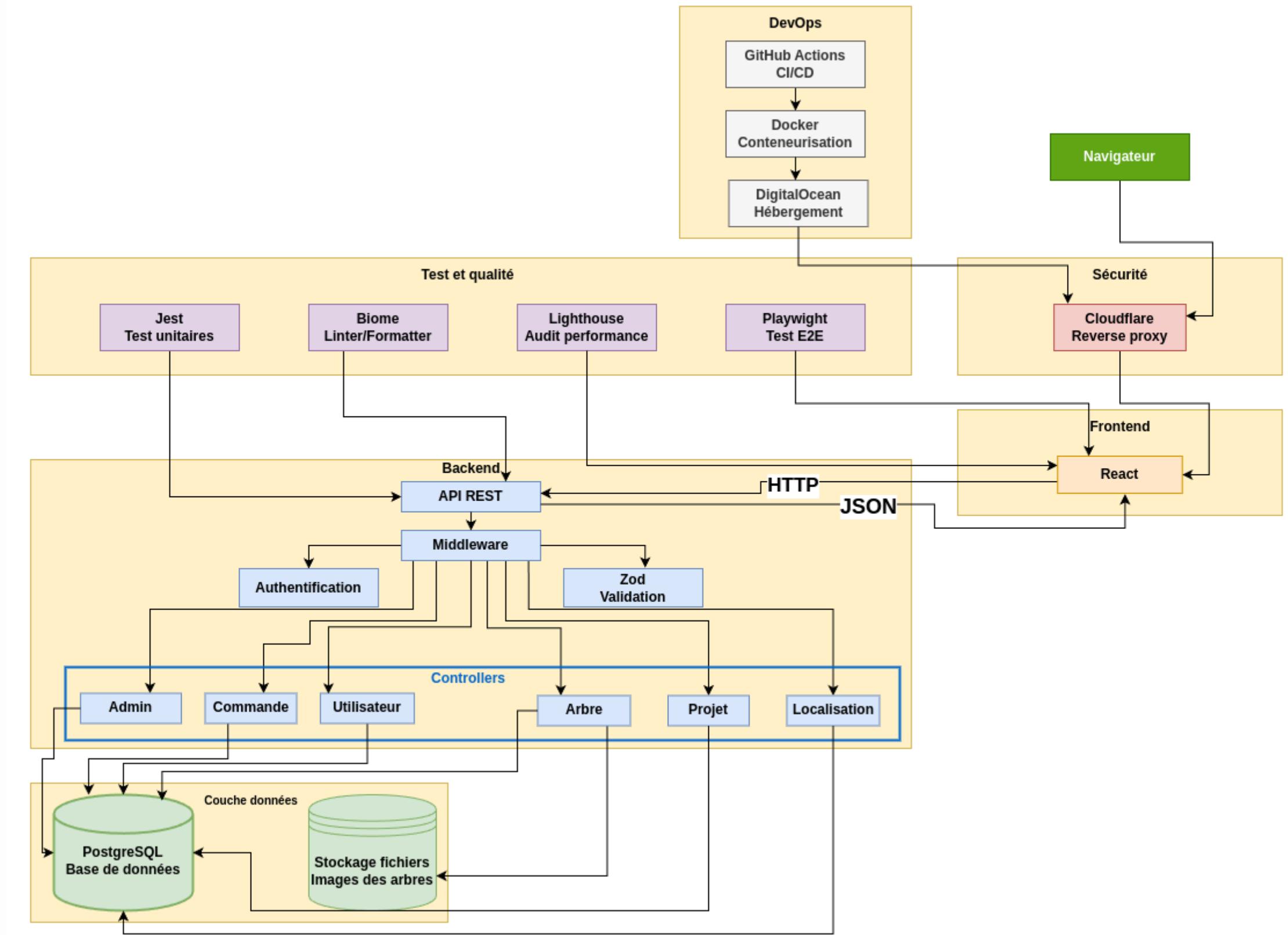
partiellement découplé:

BASE DE DONNÉES / BACKEND (models et controllers)

- **models**: définissent la logique métier et dialoguent avec la base de données
- **controllers**: intermédiaires entre les models et la requête HTTP
- **routers**: définissent les endpoints et lient une URL à une méthode du controller

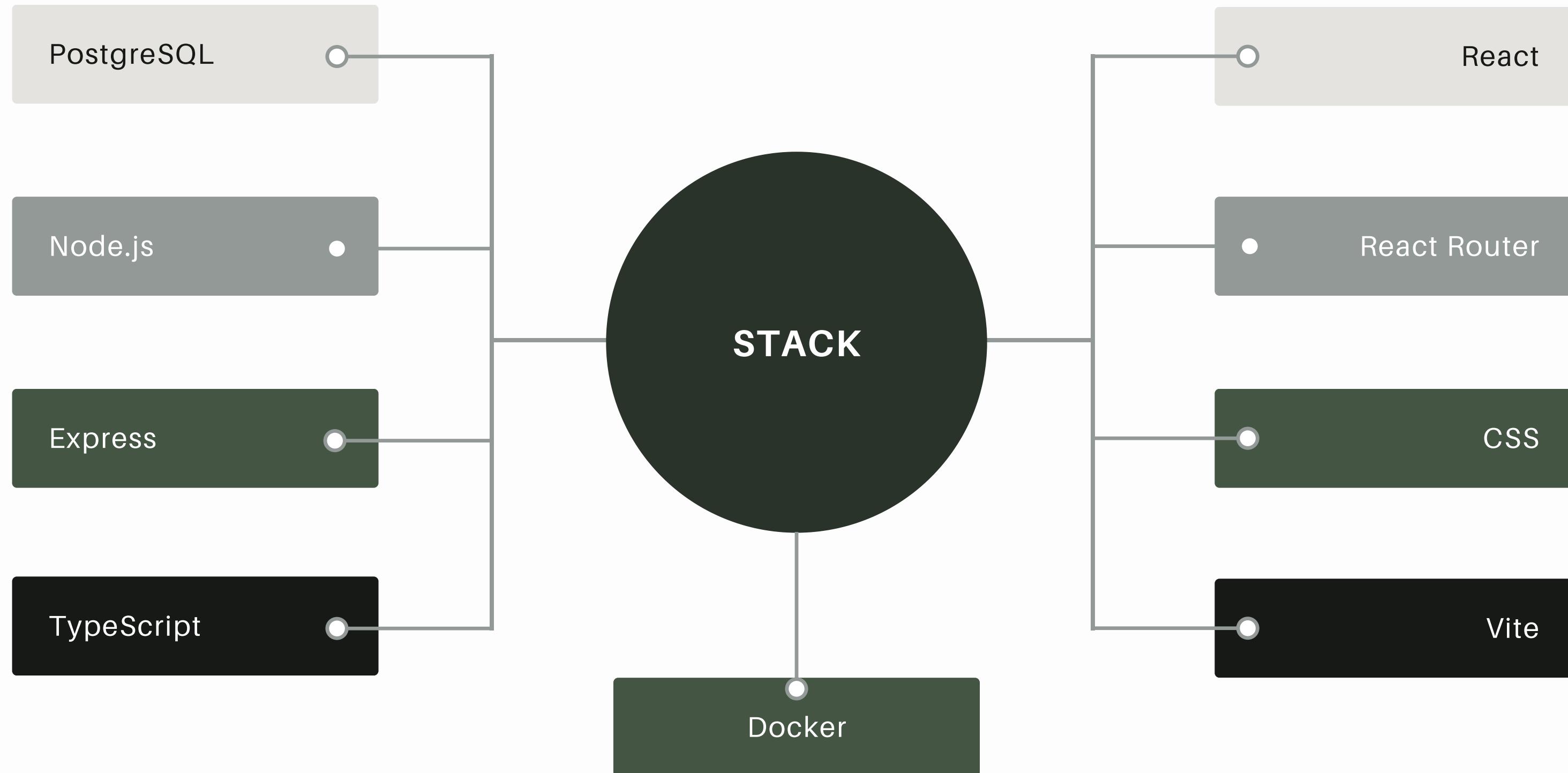
FRONTEND (view)

- **consomme l'API REST**
- **affiche l'interface** visible par le client



PRINCIPALES TECHNOLOGIES

ENVIRONNEMENT TECHNIQUE



NOTRE **EQUIPE**

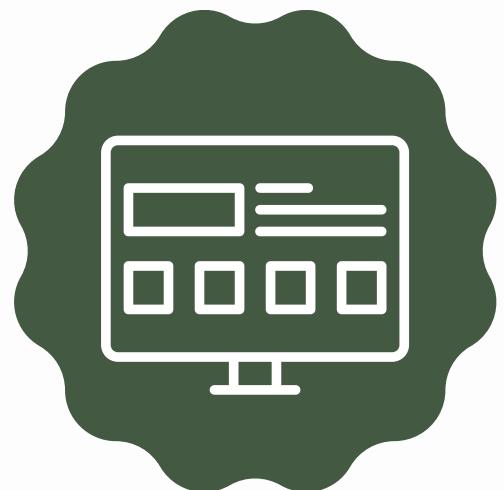
5 rôles clés:



CHEF DE PROJET

Gestion:

- délais
- équipe
- vision globale



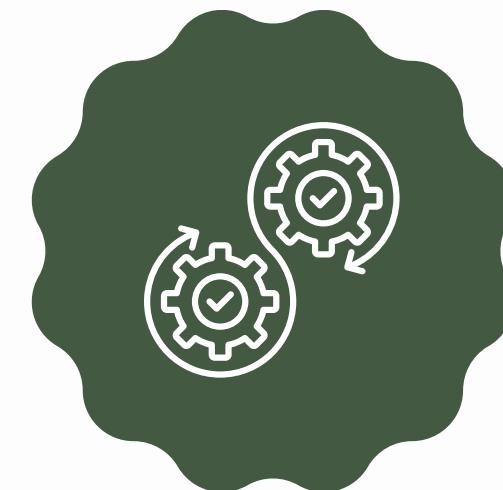
UI / UX

- parcours utilisateur
- wireframes
- maquettes



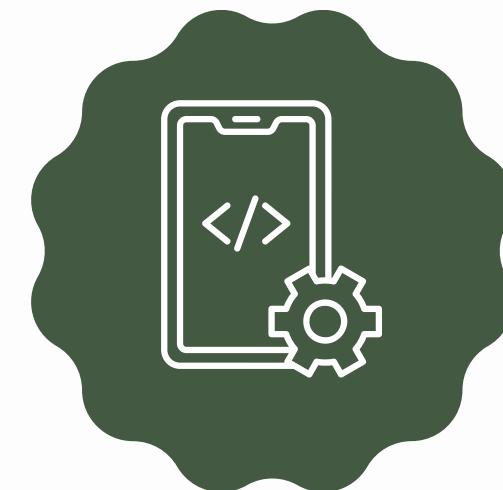
TESTEUR

- responsable des tests



DEVOPS

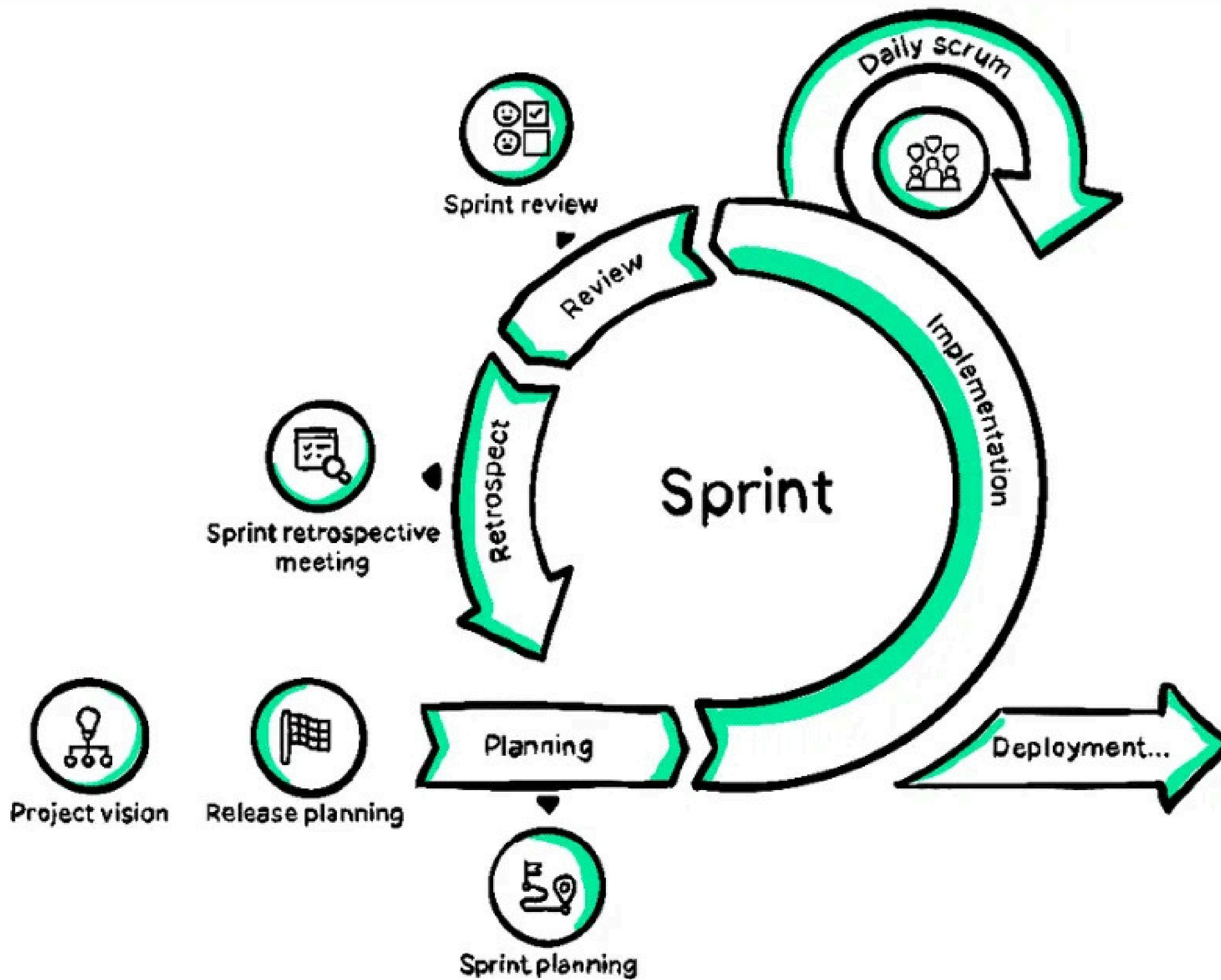
- processus de déploiement
- configuration du serveur



DEVELOPPEUR

- code frontend
- code backend

METHODOLOGIE AGILE SCRUM



ORGANISATION EN SPRINTS:

3 sprints:

- 1.conception
2. models, controllers, endpoints, tests / composants, login, ajout au panier
3. inscription, mise en place de stripe/ github actions et déploiement

DAILY chaque matin

REVIEW à chaque fin de sprint:

- présenter les fonctionnalités
- ajuster le planning si nécessaire

RETROSPECTIVE pour trouver des axes d'amélioration

COLLABORATION OUTILS

VERSIONNING:

- GitHub

PRIORISATION / SUIVI DES TACHES:

outil project de GitHub

- status board
- backlog

COMMUNICATION:

- Slack
- Discord

CODE:

- outil liveshare de VSC (pair programming)

DESIGN:

- Figma (maquette)

Title	Priority	Status	Assignees	Sprint	Reac
1 Création des items sprint 0 sur Github Projects #1	P0	Done	adam-benye...	Sprint 0	CP4
2 Conception des User Stories #4	P0	Done	adam-benye...	Sprint 0	CPS
3 Valider les technologies à utiliser #5	P0	Done	adam-benye...	Sprint 0	CP6
4 Définition de l'organisation sur git (branche par feature, protéger la... #6	P0	Done	adam-benye...	Sprint 0	CP4
5 Définition des livrables #7	P0	Done	cedric-famib...	Sprint 0	CP4
6 Création des tâches et du github project (format Kanban) #8	P0	Done	adam-benye...	Sprint 0	CP4
7 Recherche éco-conception compatible React / SEO #12	P0	Done	cedric-famib...	Sprint 0	CP6
8 Compléter le cahier des charges #20	P0	Done	cedric-famib...	Sprint 0	CP4
9 Définir une convention à suivre pour les commits #21	P0	Done	adam-benye...	Sprint 0	CP4
10 Définir une convention à suivre pour les Pull Request (PR) #22	P0	Done	adam-benye...	Sprint 0	CP4
11 Faire concorder les tâches du Kanban avec le Rec sprint 0 #24	P0	Done	adam-benye...	Sprint 0	CP4
12 Reprendre la présentation du projet #25	P0	Done	VincentVautier	Sprint 0	CP4
13 US001 - Accéder à la landing page #45	P0	Done	ChloeGarcia...	Sprint 1	CP2 CP3 CP7 CP8
14 US002 - Consulter le catalogue des produits #46	P0	Done	ChloeGarcia...	Sprint 1	CP2 CP3 CP7 CP8
15 US003 - Consulter un produit individuel #48	P0	Done	ChloeGarcia...	Sprint 1	CP2 CP3 CP7 CP8
16 Crée un dictionnaire de données #93	P0	Done	VincentVautier	Sprint 0	CP7

Iteration	Completed	In progress	In review	Done
No Iteration	Show empty values	This item hasn't been started	This is actively being worked on	This has been completed
Completed	84	greenroots #57 US011 - Accéder au site en anglais/français au choix	greenroots #63 US017 - Réinitialiser mon mot de passe	greenroots #1 Création des items sprint 0 sur Github P
In progress	20 / 3	greenroots #58 US012 - Naviguer sur le site au clavier	greenroots #64 US018 - Supprimer mon compte	greenroots #2 Recherches graphiques (inspirations, et palette graphique)
In review	0 / 5	greenroots #59 US013 - Suivre le site avec des outils d'accessibilité	greenroots #66 US020 - Modifier mes informations personnelles	greenroots #3 Recherches sur les questions en suspen
Ready	0 / 0	greenroots #67 US021 - Consulter le statut d'une de mes commandes	greenroots #68	greenroots #4 Conception des User Stories
Backlog	20 / 3	greenroots #69	+ Add item	greenroots #5 Valider les technologies à utiliser

CONCEPTION USER STORIES

Elles décrivent le besoin du point de vue d'un utilisateur.

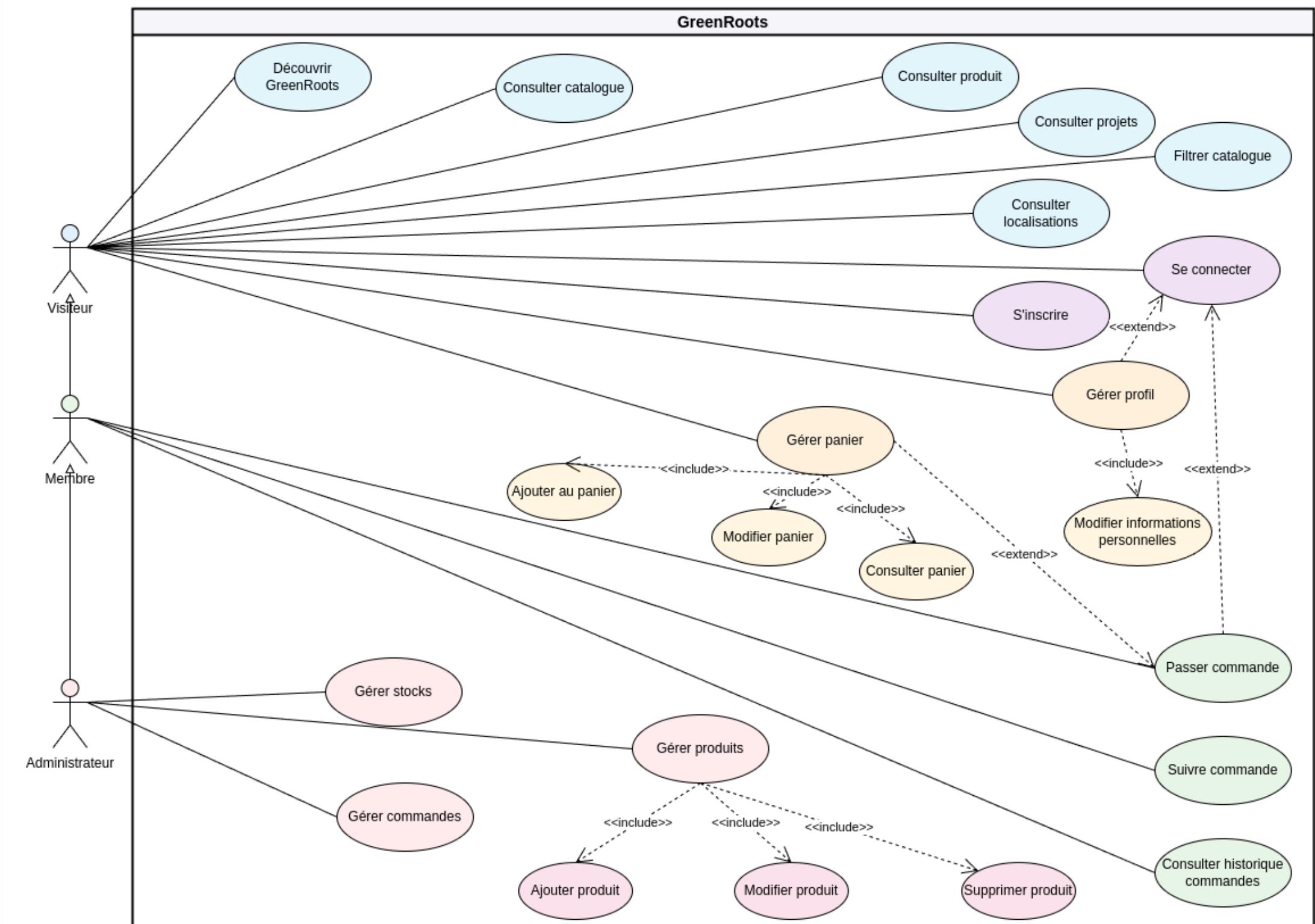
3 rôles définis:

- **Visiteur** (utilisateur non connecté)
- **Membre** (utilisateur connecté)
- **Administrateur**

En tant que	Je veux	Afin de	Sprint
Visiteur	accéder à la landing page	découvrir GreenRoots	1
Visiteur	consulter le catalogue des produits	voir les produits proposés	1
Visiteur	consulter un produit individuel	connaître les détails du produit	1
Visiteur	filtrer le catalogue par localisation	affiner ma recherche	1
Visiteur	contacter l'équipe du site par mail	transmettre ou demander une information	1
Visiteur	ajouter un ou plusieurs produits au panier	préparer ma commande	2
Visiteur	consulter mon panier	supprimer / modifier / valider mon panier	2
Visiteur	filtrer le catalogue par localisation	affiner ma recherche	2
Visiteur	m'inscrire sur le site	bénéficier des avantages membre	2

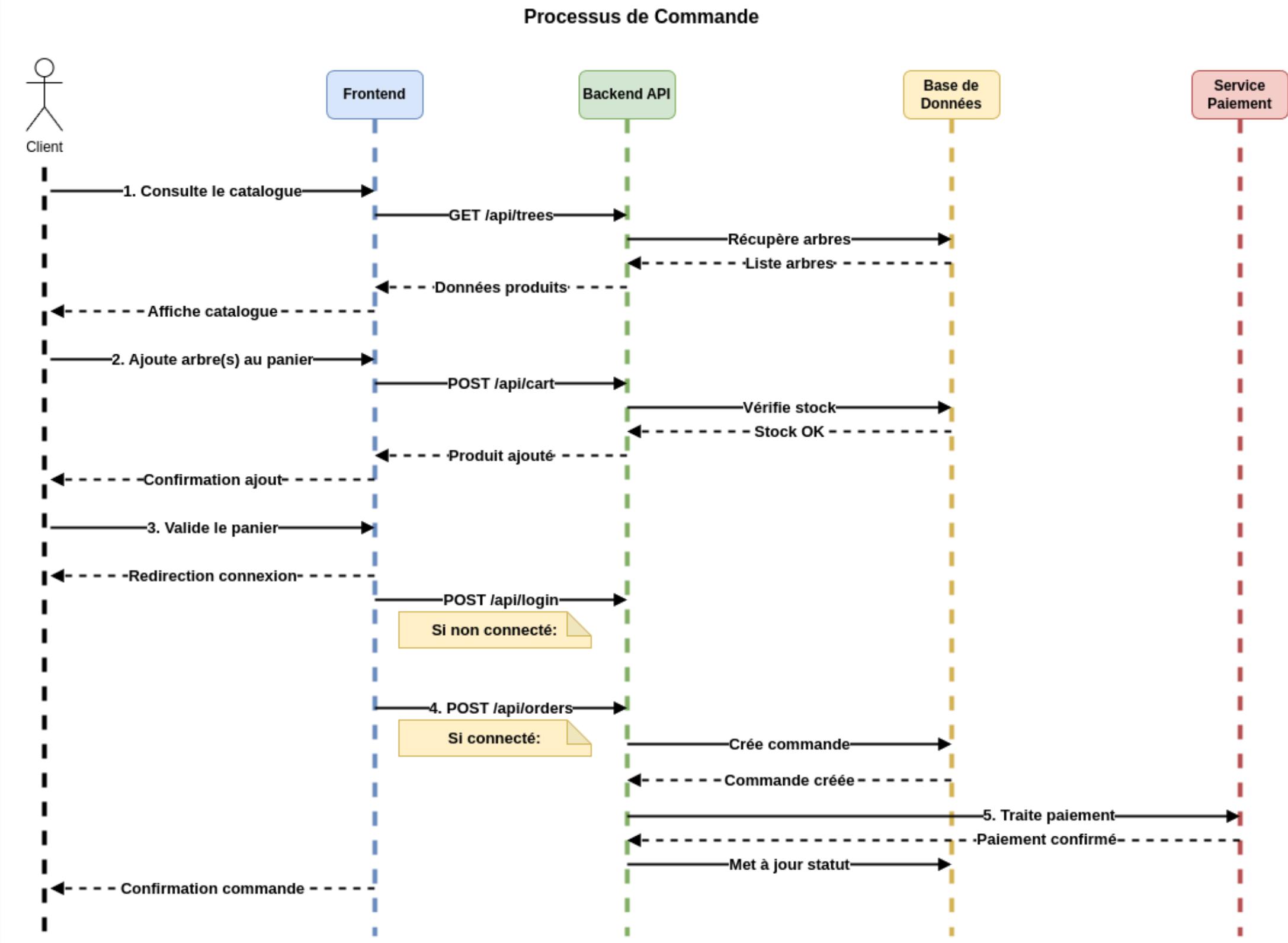
CONCEPTION USER CASES

Elles décrivent les interactions entre le système et acteurs.



CONCEPTION DIAGRAMME DE SEQUENCE

- décrit comment une fonctionnalité se déroule étape par étape
- fonctionnalité principale de GreenRoots = tunnel d'achat

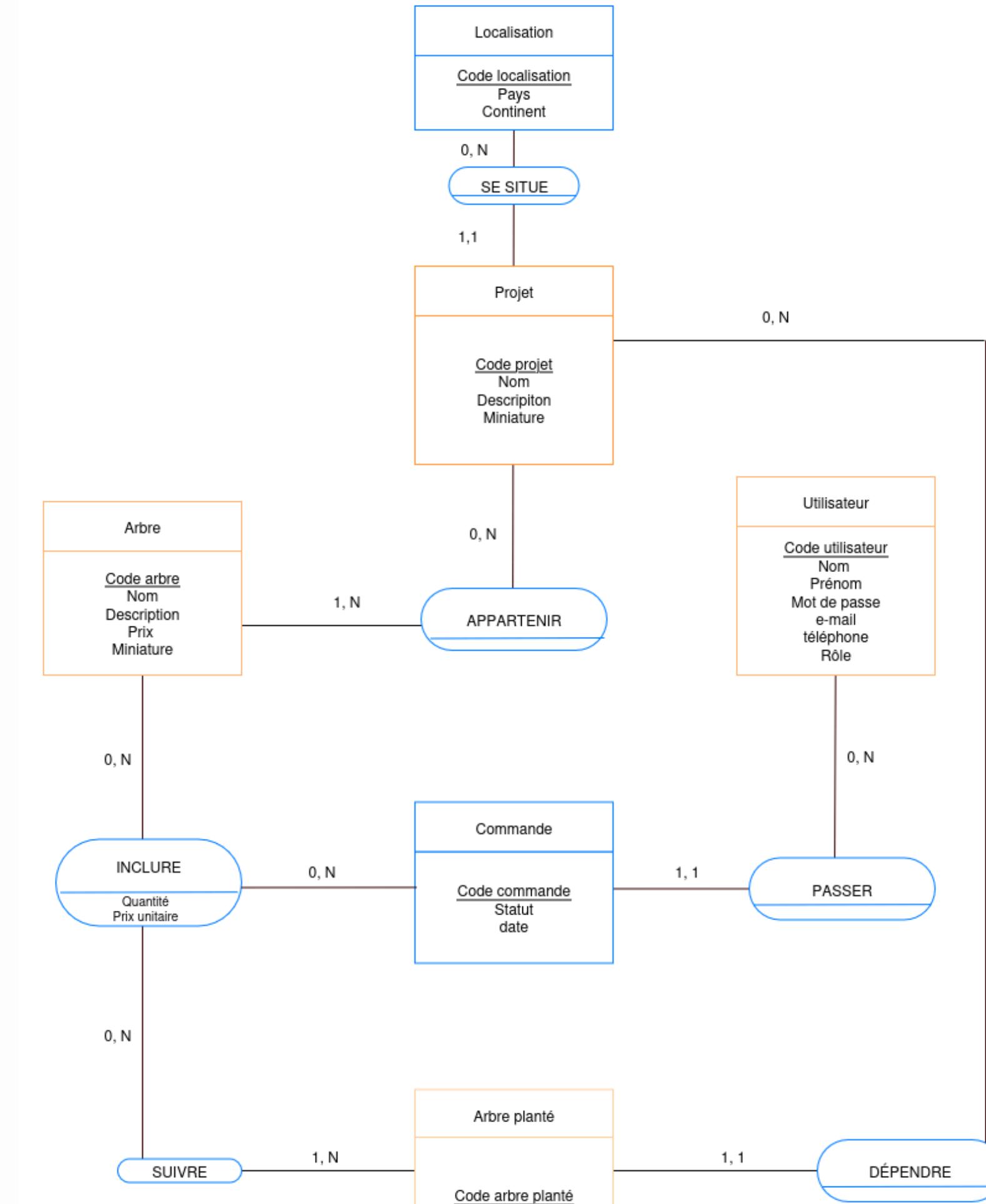


CONCEPTION MCD

Modèle Conceptuel de Données =
représentation visuelle de la base de
données avant sa création

Il définit:

- les **entités** avec leurs **attributs**
- les **relations** avec leurs **cardinalités**



CONCEPTION MLD

Modèle Logique de Données =traduit le MCD en structure de tables

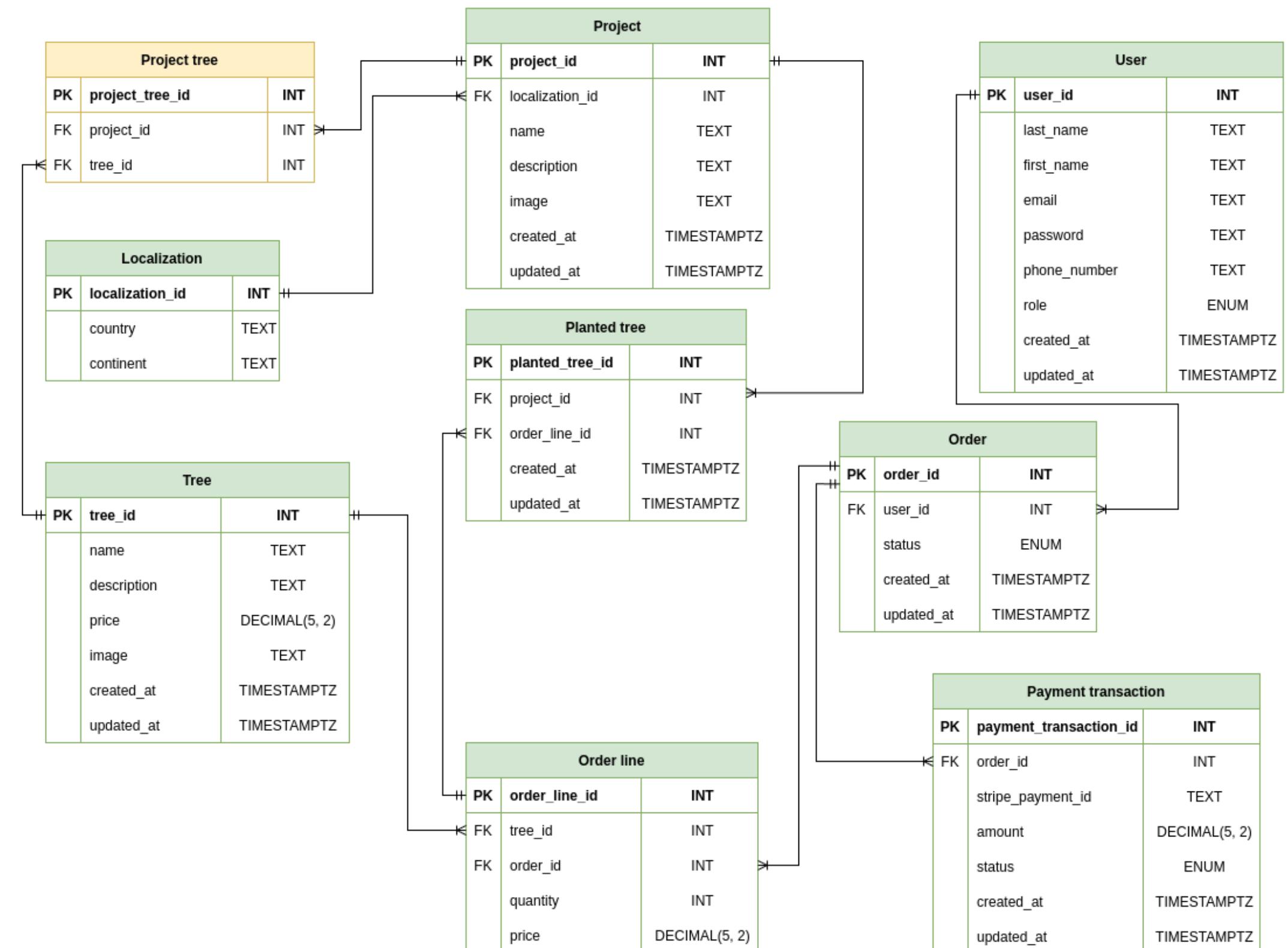
Il définit:

- les **tables et les colonnes**
- les **relations**
- les **clés primaires et étrangères**

```
1
2
3     Arbre (CodeArbre, Nom, Description ,Prix ,Miniature )
4
5     Utilisateur (CodeUtilisateur, Nom, Prénom, E-mail, Mot de passe, Téléphone, Rôle )
6
7     Localisation (CodeLocalisation, Pays, Continent )
8
9     Commande (CodeCommande, Statut, date)
10
11    Projet (CodeProjet, Nom, Description, Miniature)
12
13    Arbre Planté (CodeArbrePlanté, Date)
14
15    INCLUDE (#CodeArbre, #CodeCommande, Quantité, Prix unitaire)
16
17    APPARTENIR (#Codeprojet, #CodeArbre)
18
19    DÉPENDRE (#CodeProjet, #CodeArbrePlanté)
20
21    SUIVRE (#CodeCommande, #CodeArbrePlanté)
22
23    SE SITUER (#CodeLocalisation, #CodeProjet)
24
25    PASSER (#CodeCommande, #CodeUtilisateur)
26
```

CONCEPTION ERD

Entity Relationship Diagram =
représentation visuelle incluant les **clés**
étrangères et primaires



CONCEPTION DICTIONNAIRE DE DONNÉES

- **décrit les données utilisées**
- recense chaque champ, sa signification, son type, ses contraintes, ses liens

PERMET DE:

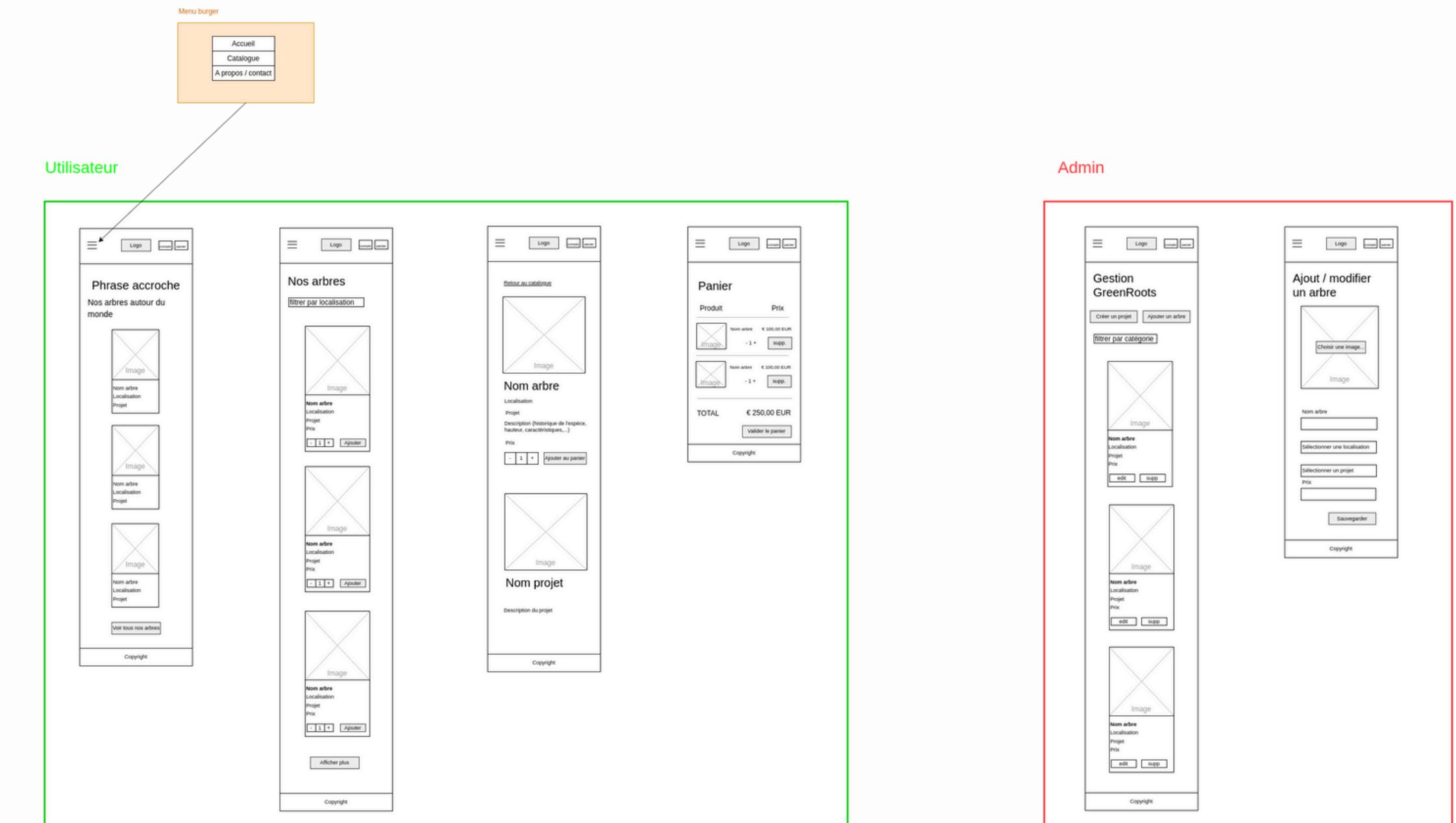
- clarifier le sens
- faciliter la conception et la maintenance
- assurer la cohésion des données
- documenter

Entité Tree

Nom de données	Description	Type de données	Format	Taille	Contraintes	Relation	Usage et contexte d'utilisation
Tree_ID	Identifiant unique de l'arbre	Numérique			Clé primaire	Project_ID (clé étrangère)	Identification de l'arbre dans la BDD
Name	Nom de l'arbre	Texte		50 caractères			Identification et affichage du nom de l'arbre
Description	Description	Texte		250 caractères			Décrire l'arbre
Price	Prix total TTC	Numérique					Calcul du prix total d'une future commande
Image	Illustration de l'arbre	Texte	Url				Identification et affichage l'image de l'arbre

CONCEPTION WIREFRAMES

- **représentation simplifiée** d'un écran
- permet de se concentrer sur
l'ergonomie
- approche **mobile first**



CONCEPTION ECO CONCEPTION

Élément central de GreenRoots:

- uniquement **frameworks et librairies nécessaires**
- utilisation des **polices par défaut**

UI - GreenRoots

Typo

Open Sans ou Arial ou Helvetica (polices standard préchargées)

Colors

Main



Dark Green
455E46



Dark Grey
212529



Black
#000000



Pearl
#FAF2E4



Light Grey
#F6F7F9

Cards



Light
Yellow
#F0DDB5



Light
Green
#C9CABA



Light
Red
#EEDED2



Light
Brown
#EBEED2

Logo



Icons



Favicon



Component

Button

Button

Catégorie ▼

- 1 +

Bouton primaire

Bouton secondaire

Select

Quantité

Complex component



Chêne €100,00

Bourgogne, France

Forêt du Morvan

- 1 +

Ajouter

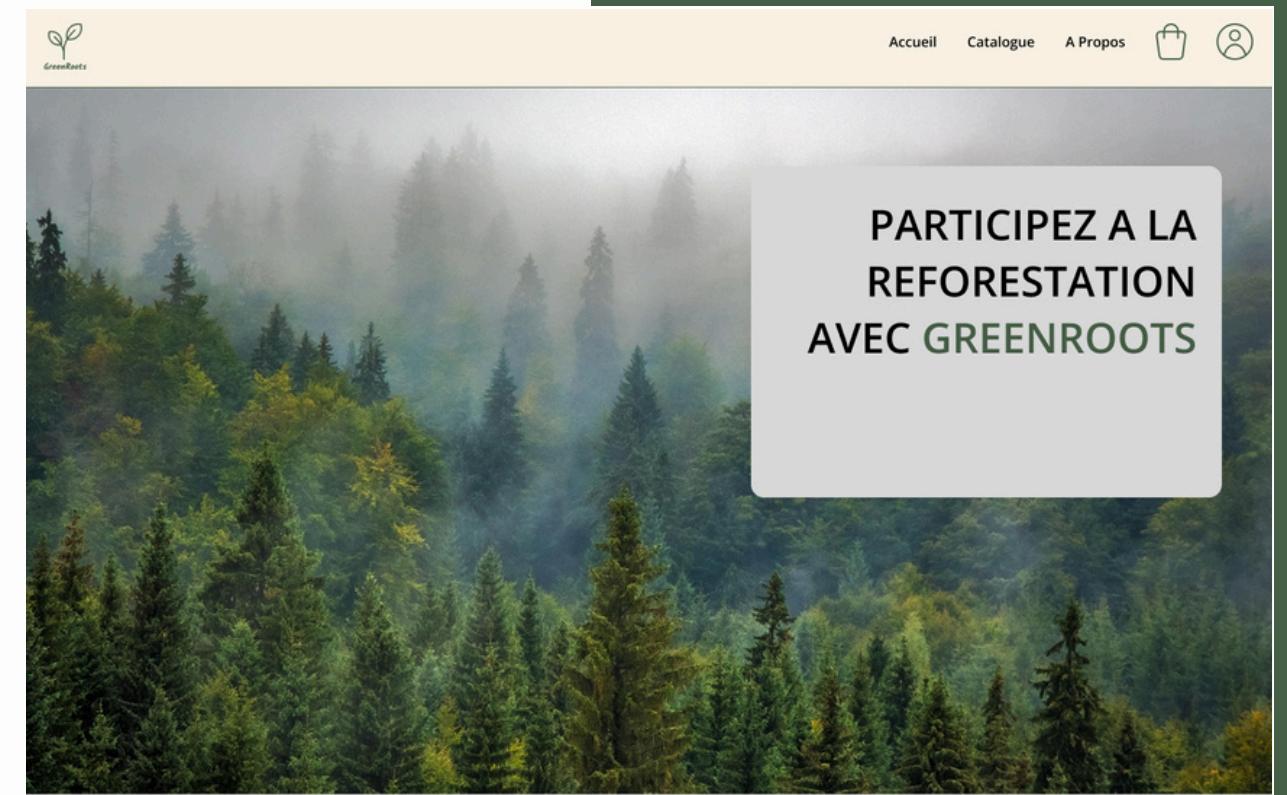
CONCEPTION ECO CONCEPTION

Élément central de GreenRoots:

- uniquement **frameworks et librairies nécessaires**
- utilisation des **polices par défaut**
- limiter les **requêtes**
- utilisation de la **pagination**
- navigation la plus directe possible
- taille des **images optimisées**, pas de vidéo
- **choix des couleurs**: pas de fond blanc
- **gestion du cache**



PARTICIPEZ A LA REFORESTATION AVEC GREENROOTS



Partout dans le monde, nos projets revitalisent la planète
make our planet green again!

[Découvrez tous nos arbres](#)

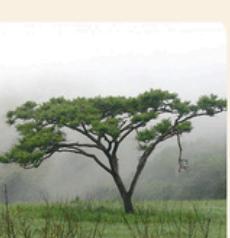


Chêne €100,00

Bourgogne, France

Forêt du Morvan

- 1 + Ajouter



Acacia €90,00

Quảng Bình, Vietnam

Plantation Dong Chau

- 1 + Ajouter



Cocotier €125,00

Abidjan, Côte d'Ivoire

Plantation Grand-Bassam

- 1 + Ajouter

Notre entreprise Catalogue A propos / contact Ressources CGV Mentions légales Politique de confidentialité

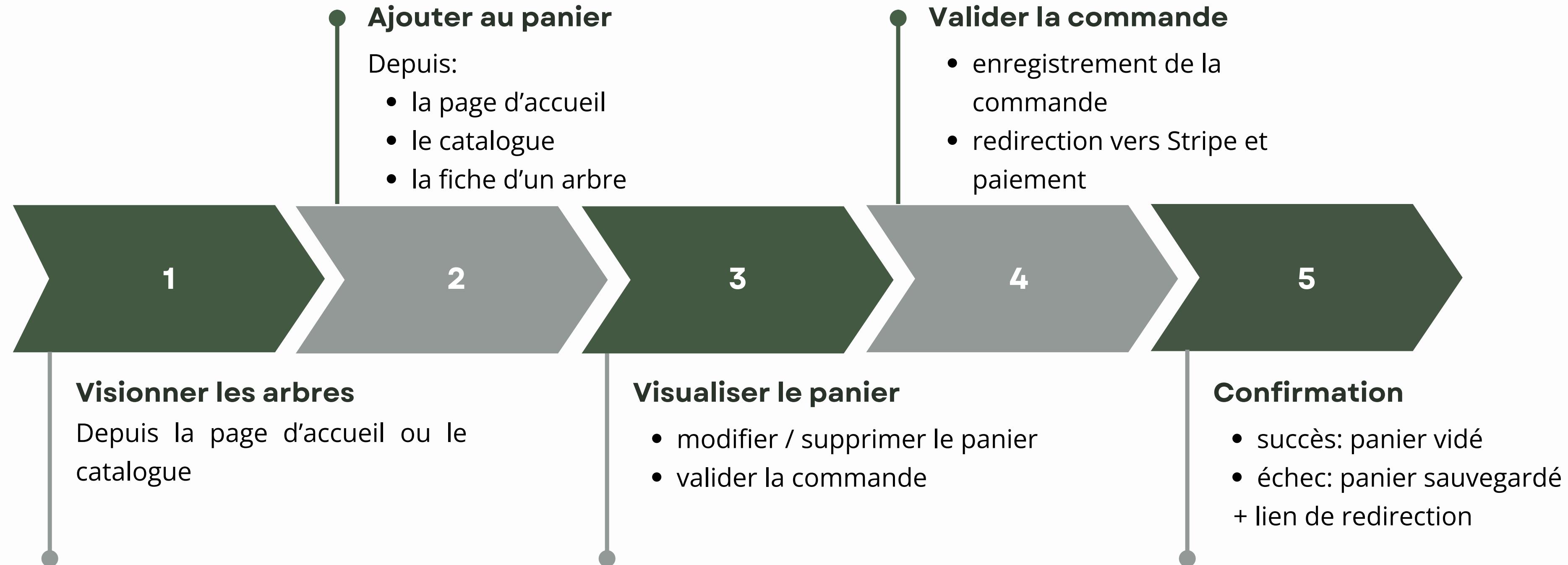
ECO CONCEPTION **CACHE**

React Router:

- **SSR** = le serveur génère le HTML complet à chaque requête avant de l'envoyer au navigateur
- HTML stocké dans le navigateur ou sur le réseau du serveur
 - max-age=300 → les navigateurs peuvent garder la page en cache 5 minutes
 - s-maxage=3600 → les CDN / reverse proxies peuvent la garder 1 heure

```
// Http headers for caching
export function headers() {
  return {
    "Cache-Control": "max-age=300, s-maxage=3600",
  };
}
```

FONCTIONNALITE ACHAT



ETAPE 1

VISIONNER

LES ARBRES

BACKEND

1. treeRouter définit les **endpoints** liés aux arbres
2. treeController: **reçoit les requêtes** et **coordonne** la logique en faisant appel à treeModel

```
treeRouter.get('/api/trees', treeController.index);
treeRouter.get('/api/trees/homepage', treeController.homepage);
treeRouter.get('/api/trees/:id', treeController.show);
treeRouter.get('/api/trees/continent/:continent', treeController.findByContinent);
```

```
async show(req: Request, res: Response) {
  try {
    const idParam = req.params.id;

    if (!idParam) {
      throw new Error("Missing id param");
    }

    const id = parseInt(idParam);

    if (isNaN(id)) {
      return res.status(400).json({
        message: 'Invalid tree ID',
        status: 400
      });
    }

    const tree = await treeModel.findByIdWithProjectsAndLocalizations(id);

    if (!tree) {
      return res.status(404).json({
        message: 'Tree not found',
        status: 404
      });
    }

    res.json({
      message: 'Tree retrieved successfully',
      data: tree,
      status: 200
    });
  } catch (error) {
    res.status(500).json({
      message: 'Error retrieving tree',
      error: error instanceof Error ? error.message : 'Unknown error',
      status: 500
    });
  }
},
```

ETAPE 1 VISIONNER LES ARBRES

BACKEND

1. treeRouter définit les endpoints liés aux arbres
2. treeController: reçoit les requêtes et coordonne la logique en faisant appel à treeModel
3. treeModel gère la **logique métier** et interroge la base de données via des requêtes SQL

```
// Get a specific tree with its projects and localizations
async findByIdWithProjectsAndLocalizations(id: number): Promise<any | null> {
  try {
    const query = `

      SELECT
        t.tree_id,
        t.name as tree_name,
        t.description as tree_description,
        t.price,
        t.image as tree_image,
        t.price_id,
        t.created_at as tree_created_at,
        t.updated_at as tree_updated_at,
        p.project_id,
        p.name as project_name,
        p.description as project_description,
        p.image as project_image,
        p.localization_id,
        p.created_at as project_created_at,
        p.updated_at as project_updated_at,
        l.country,
        l.continent
      FROM tree t
        LEFT JOIN project_tree pt ON t.tree_id = pt.tree_id
        LEFT JOIN project p ON pt.project_id = p.project_id
        LEFT JOIN localization l ON p.localization_id = l.localization_id
      WHERE t.tree_id = $1
      ORDER BY p.created_at DESC
    `;
    const result = await this.db.query(query, [id]);

    if (result.rows.length === 0) {
      return null;
    }

    const firstRow = result.rows[0];
    const tree = {
      tree_id: firstRow.tree_id,
      name: firstRow.tree_name,
      description: firstRow.tree_description,
      price: firstRow.price,
      image: firstRow.tree_image,
      created_at: firstRow.tree_created_at,
      updated_at: firstRow.tree_updated_at,
      projects: result.rows
        .filter((row: TreeProjectRow) => row.project_id !== null)
        .map((row: TreeProjectRow) => {
          project_id: row.project_id,
          name: row.project_name,
          description: row.project_description,
          image: row.project_image,
          localization_id: row.localization_id,
          created_at: row.project_created_at,
          updated_at: row.project_updated_at,
          localization: row.localization_id ? {
            localization_id: row.localization_id,
            country: row.country,
            continent: row.continent
          } : null
        })
    };

    return tree;
  } catch (error) {
    throw new Error(`Error fetching tree with ID ${id}, projects and localizations: ${error}`);
  }
}
```

ETAPE 1 VISIONNER LES ARBRES

FRONTEND

1. URLs définies dans le dossier /routes
2. la page tree affiche le détail d'un arbre

FONCTION LOADER

- extrait l'id de l'arbre depuis l'URL
(params.id)
- appelle l'API via fetch

```
route("catalog/:continent?", "pages/catalog/catalog.tsx"),
route("tree/:id", "pages/tree/tree.tsx"),

export async function loader(args: Route.LoaderArgs) {
  // Use environment variable for API URL
  const apiUrl = import.meta.env.VITE_API_URL;
  const { params } = args;
  const treeId = params.id;

  const response = await fetch(`${apiUrl}/trees/${treeId}`);
  const json = await response.json();

  const tree = json.data;
  const project = tree.projects?.[0] ?? null;

  return { tree, project };
}
```

ETAPE 1

VISIONNER

LES ARBRES

COMPOSANT TREE

- gère la quantité d'arbres à ajouter au panier selon un état local
- ajoute au panier via handleToCart

```
export default function Tree(props: Route.ComponentProps) {
  const { tree, project } = props.loaderData;

  // local state
  const [quantity, setQuantity] = useState(1);
  const [isAddingToCart, setIsAddingToCart] = useState(false);
  const [cartMessage, setCartMessage] = useState<string | null>(null);
  const [cartError, setCartError] = useState<string | null>(null);

  const navigation = useNavigation();
  const isSubmitting = navigation.formAction === "/add-to-shopping-cart";

  // manage cart
  const handleAddToCart = async () => {
    setIsAddingToCart(true);
    setCartMessage(null);
    setCartError(null);

    try {
      const result = await cartService.addItem(tree.tree_id, quantity);

      if (result.success) {
        setCartMessage(`>${quantity} ${tree.name}(s) ajouté(s) au panier !`);

        // Message de succès disparaît après 3 secondes
        setTimeout(() => setCartMessage(null), 3000);
      } else {
        setCartError(result.error || "Erreur lors de l'ajout au panier");
      }
    } catch (error) {
      console.error("Erreur ajout panier:", error);
      setCartError("Erreur de connexion. Veuillez réessayer.");
    } finally {
      setIsAddingToCart(false);
    }
  };
}
```

ETAPE 1

VISIONNER

LES ARBRES

COMPOSANT TREE

- gère la quantité d'arbres à ajouter au panier selon un état local
- ajoute au panier via handleToCart
- utilise les données reçues pour afficher dynamiquement les informations de l'arbre et du projet associé

```
return (
  <main className="tree-page">
    {/* main section : product */}
    <section className="tree-product">
      <div className="tree-image-container">
        <img
          src={tree.image}
          alt={`${tree.name} - Arbre à parrainer`}
          loading="lazy"
          className="tree-image"
        />
      </div>

      <div className="tree-details">
        <div className="tree-header-info">
          <h1
            className={`tree-title continent-${project?.localization?.continent?.
              toLowerCase()?.replace(/\s+/g, "-")?.replace("é", "e") || "default"}`}
          >
            {tree.name}
          </h1>

          <div className="tree-meta">
            <div className="tree-location">
              <img src={localizationIcon} alt="" aria-hidden="true" />
              <span>{project?.localization?.country}</span>
            </div>
            <div className="tree-project-ref">
              <img src={projectIcon} alt="" aria-hidden="true" />
              <span>{project?.name}</span>
            </div>
          </div>
        </div>

        <div className="tree-description">
          <p>{tree.description}</p>
        </div>

        <div className="tree-price">
          <span className="price-value">{tree.price}€</span>
          <span className="price-unit">par arbre</span>
        </div>
      </div>
    </section>
  </main>
)
```

The screenshot shows a product page for a coconut tree. At the top right, there's a navigation bar with icons for menu, search, cart, and user profile. Below it, a breadcrumb trail says "Retour au catalogue". The main content features a large image of a palm tree trunk and canopy. To the left, the product name "Cocotier" is displayed with a location pin icon and "Abidjan, Côte d'Ivoire". Below that is a "Plantation Grand-Bassam" section with a smaller image of palm trees. The central text describes the coconut tree (Cocos nucifera) as a tall tropical palm reaching up to 30 meters, with long leaves and coconuts. It mentions its use in agriculture and fiber production. A price of €125,00 is shown with a quantity selector set to 1 and an "Ajouter au panier" button. At the bottom, there's a footer with links to "Notre entreprise", "Catalogue", "A propos / contact", "Ressources", "CGV", "Mentions légales", and "Politique de confidentialité".

ETAPE 2 AJOUTER AU PANIER

- clic sur le bouton ajouter au panier déclenche handleToCart
- handleToCart = fonction asynchrone qui appelle cartService
- vérification des paramètres
- validation auprès de l'API
- récupération des informations de l'arbre depuis l'API

```
// Add item to cart with API validation
async addItem(tree_id: number, quantity: number) {
  try {
    // Convert to numbers to ensure correct data types
    const numericTreeId = Number(tree_id);
    const numericQuantity = Number(quantity);

    // Validate that conversion was successful
    if (isNaN(numericTreeId) || isNaN(numericQuantity)) {
      throw new Error('Invalid tree_id or quantity - must be valid numbers');
    }

    // First, validate with API
    const response = await fetch(this.API_BASE, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        tree_id: numericTreeId,
        quantity: numericQuantity
      })
    });

    const result = await response.json();

    if (!response.ok) {
      throw new Error(result.message || 'Failed to validate item');
    }
  }
}
```

ETAPE 2 AJOUTER AU PANIER

- clic sur le bouton ajouter au panier déclenche handleToCart
- handleToCart = fonction asynchrone qui appelle cartService

récupération du panier existant depuis le local storage

si l'arbre existe déjà → quantité et prix mis à jour

s'il n'existe pas → ajout d'un nouvel objet au tableau items

le total est recalculé
le panier est sauvegardé dans le local storage

affichage d'un message de confirmation

```
// If validation successful, add to localStorage
const cart = this.getCart();
const existingItemIndex = cart.items.findIndex((item: { tree_id: number; }) =>
item.tree_id === numericTreeId);

if (existingItemIndex > -1) {
    // Update existing item quantity
    const oldQuantity = cart.items[existingItemIndex].quantity;
    cart.items[existingItemIndex].quantity += numericQuantity;
    cart.items[existingItemIndex].price = result.tree.price; // Update price

    cart.total = this.calculateTotal(cart.items);
    this.saveCart(cart);

    // Log quantity change
    this.logCartContents('QUANTITY UPDATED', {
        item: result.tree.name,
        old_quantity: oldQuantity,
        new_quantity: cart.items[existingItemIndex].quantity,
        added: numericQuantity
    });
} else {
    // Add new item
    const cartItem = {
        tree_id: result.tree.tree_id,
        name: result.tree.name,
        description: result.tree.description,
        price: result.tree.price,
        image: result.tree.image,
        quantity: numericQuantity
    };
    cart.items.push(cartItem);

    cart.total = this.calculateTotal(cart.items);
    this.saveCart(cart);

    // Log new item added
    this.logCartContents('ITEM ADDED', {
        item: result.tree.name,
        quantity: numericQuantity,
        price: result.tree.price
    });
}

return {
    success: true,
    cart,
    message: 'Item added to cart successfully'
};

} catch (error) {
    console.error('Error adding item to cart:', error);
    if (error instanceof Error) {
        return {
            success: false,
            message: error.message
        }
    }
}
```

ETAPE 3

VISUALISER

LE PANIER

PAGE shopping-cart

Fonction asynchrone action:

- déclenchée à la soumission du panier
- récupère la session à partir du cookie
- sinon l'utilisateur est redirigé vers la page de connexion

```
// action by clicking on validate the cart
export async function action(args: Route.ActionArgs) {
  // check if a user is logged in
  const session = await getSession(args.request.headers.get("Cookie"));
  const token = session.get("token");
  // if not, redirect to the login page
  if (!token) {
    return redirect("/login");
  }
  // if yes, retrieve the formData
  const formData = await args.request.formData();
  const rawItems = formData.get("items");
  if (!rawItems) {
    return new Response("Panier vide ou invalide", { status: 400 });
  }
  let items: Item;
  try {
    const parsedItems = JSON.parse(rawItems.toString());
    // each item is filtered to retain only the three required fields by backend
    items = parsedItems.map((item: Item) => ({
      tree_id: item.tree_id,
      quantity: item.quantity,
      price: item.price,
    }));
  } catch (e) {
    return new Response("Erreur de parsing JSON", { status: 400 });
  }
}
```

ETAPE 3

VISUALISER

LE PANIER

PAGE shopping-cart

Fonction asynchrone action:

- extrait le champ items du formData
- le champ est parsé pour obtenir la liste des articles



```
// action by clicking on validate the cart
export async function action(args: Route.ActionArgs) {
  // check if a user is logged in
  const session = await getSession(args.request.headers.get("Cookie"));
  const token = session.get("token");
  // if not, redirect to the login page
  if (!token) {
    return redirect("/login");
  }
  // if yes, retrieve the formData
  const formData = await args.request.formData();
  const rawItems = formData.get("items");
  if (!rawItems) {
    return new Response("Panier vide ou invalide", { status: 400 });
  }
  let items: Item;
  try {
    const parsedItems = JSON.parse(rawItems.toString());

    // each item is filtered to retain only the three required fields by backend
    items = parsedItems.map((item: Item) => {
      tree_id: item.tree_id,
      quantity: item.quantity,
      price: item.price,
    });
  } catch (e) {
    return new Response("Erreur de parsing JSON", { status: 400 });
  }
}
```

ETAPE 3

VISUALISER

LE PANIER

PAGE shopping-cart

Fonction asynchrone action:

- les articles sont envoyés au backend
- en cas d'erreur un message est envoyé
- en cas de succès, urlSession est renournée → redirection vers l'interface de paiement Stripe

```
// save the cart to the backend
const response = await fetch(`${API_URL}/orders`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
  body: JSON.stringify({ items }),
});

if (!response.ok) {
  const result = await response.json().catch(() => null);
  const message =
    result?.message || "Erreur lors de l'enregistrement de la commande.";
  return new Response(message, { status: 500 });
}

const {urlSession} = await response.json();

return redirect(urlSession);
}
```

ETAPE 3 VISUALISER LE PANIER

PAGE shopping-cart

Composant ShoppingCartPage:

- affiche la page “panier”
- composant ShoppingCartList = détail du panier
- au clic sur “valider le panier”
 - le panier est transformé en json et inséré dans un formData
 - la fonction action est déclenchée
- hook useShoppingCart fournit 2 actions
 - updateItem
 - removeItem

```
// table of items in shopping cart
const ShoppingCartList = () => {
  const [shoppingCart, { updateItem, removeItem }] = useShoppingCart();

  const submit = useSubmit();

  return (
    <>
      <table className="cart-container">
        <thead>
          <tr>
            <th>Produit</th>
            <th>Quantité</th>
            <th>Prix</th>
          </tr>
        </thead>
        <tbody>
          {shoppingCart.items.map((item: Item) => (
            <ShoppingCartItem
              key={item.tree_id}
              item={item}
              onUpdate={(quantity) => {
                updateItem(item.tree_id, quantity);
              }}
              onRemove={() => removeItem(item.tree_id)}
            />
          ))}
        </tbody>
        <tfoot>
          <tr>
            <td colSpan={2} className="cart-total-text">
              TOTAL prévisionnel
            </td>
            <td className="cart-total-value">
              <Currency value={shoppingCart.total} />
            </td>
          </tr>
        </tfoot>
      </table>
      <div className="validation-button-container">
        <button
          className="validation-button"
          type="button"
          onClick={() => {
            const formData = new FormData();
            formData.append("items", JSON.stringify(shoppingCart.items));
            submit(formData, { method: "post" });
            //submit({ items: shoppingCart.items }, { method: "post" });
          }}
        >
          Valider mon panier
        </button>
      </div>
    </>
  );
}
```

ETAPE 3 VISUALISER LE PANIER

ENREGISTREMENT DE LA COMMANDE

- endpoint appelé par la fonction action de la page shopping-cart
- données de l'utilisateur récupérées de puis la session
- données de la commande récupérées depuis la requête
- enregistrement de la commande si les données sont valides

```
orderRouter.post('/api/orders', requireAuth, createOrderValidation,
ordersController.create);

async create(req: Request, res: Response) {
  try {
    const { user } = req;
    const { items } = req.body;

    if (!user || !user.user_id || !items?.length) {
      return res.status(400).json({
        error: "User and items are required",
      });
    }

    // Validate items structure
    for (const item of items) {
      if (!item.tree_id || !item.quantity || !item.price) {
        return res.status(400).json({
          error: "Each item must have tree_id, quantity, and price",
        });
      }
    }

    // Create the order
    const order = await orderModel.create({
      user_id: user.user_id,
      status: OrderStatus.PENDING,
    });
  }
}
```

ETAPE 4 VALIDER LA COMMANDE

- méthode `findByIdForCheckout` appelée pour chaque items → récupération des informations de l'arbre
- si l'arbre n'est pas trouvé: renvoie une erreur
- sinon les lignes de la commande sont enregistrée dans la base de données
- le tableau `line_items` est créé (nécessaire à Stripe)

```
for (const item of items) {
  const tree = await treeModel.findByIdForCheckout(item.tree_id);

  if (!tree || typeof tree === undefined) {
    return res.status(404).json({
      error: `Tree with ID ${item.tree_id} not found`,
    });
  }

  orderLinesData.push({
    tree_id: item.tree_id,
    order_id: order.order_id!,
    quantity: item.quantity,
    price: item.price,
  });

  line_items.push({
    price: tree.price_id,
    quantity: item.quantity,
  });
}

await orderLineModel.createMany(orderLinesData);

// Create multiple order lines
async createMany(orderLines: Omit<OrderLine, 'order_line_id'>[]): Promise<OrderLine[]> {
  try {
    const values: any[] = [];
    const placeholders: string[] = [];
    let paramCount = 1;

    for (const orderLine of orderLines) {
      placeholders.push(`(${paramCount}, ${paramCount + 1}, ${paramCount + 2}, ${paramCount + 3})`);
      values.push(orderLine.tree_id, orderLine.order_id, orderLine.quantity, orderLine.price);
      paramCount += 4;
    }

    const query = `
      INSERT INTO order_line (tree_id, order_id, quantity, price)
      VALUES ${placeholders.join(', ')}
      RETURNING order_line_id, tree_id, order_id, quantity, price
    `;

    const result: QueryResult<OrderLine> = await this.db.query(query, values);
    return result.rows;
  } catch (error) {
    throw new Error(`Error creating multiple order lines: ${error}`);
  }
}
```

ETAPE 4 VALIDER LA COMMANDE

- session Stripe créée
- contient les informations de la commande (line_items) et son identifiant (order_id)
- paiement validé → redirection vers l'URL ?success=true
- échec du paiement → redirection vers ?canceled=true

```
const session = await stripe.checkout.sessions.create({
  line_items,
  mode: "payment",
  payment_intent_data: {
    metadata: {
      order_id: order.order_id!
    }
  },
  success_url: `${redirectCheckoutPage}?success=true`,
  cancel_url: `${redirectCheckoutPage}?canceled=true`,
});

if (!session.url) {
  return res.status(500).json({
    error: "Failed to create Stripe checkout session",
  });
}

res.status(201).json({ urlSession: session.url });
} catch (error) {
  console.error("Error creating order:", error);
  res.status(500).json({
    error: "Failed to create order",
  });
},
```

ETAPE 5 CONFIRMATION

PAGE success-page

- en cas de succès (`?success=true`) → un message de confirmation s'affiche et le panier est vidé via `cartService`.
- en cas d'échec (`?error=true`) → un message d'erreur s'affiche avec un lien permettant de retourner au panier

```
export default function SuccessPage() {
  const [searchParams] = useSearchParams();
  const isSuccess = searchParams.get("success");
  const isError = searchParams.get("error");

  useEffect(() => {
    if (isSuccess === "true") {
      // if payment successfull -> delete cart
      cartService.clearCart();
    }
  }, [isSuccess]);

  return (
    <main className="success-error-container">
      {/* if payment successfull */}
      {isSuccess === "true" && (
        <>
          <h2>Commande confirmée!</h2>
          <p>
            Merci pour votre commande. Vous recevrez un mail de confirmation avec les détails de votre commande.
          </p>
          <Link className="error-success-link" to="/">
            Retour à l'accueil
          </Link>
        </>
      )}
      {/* if payment failed */}
      {isError === "true" && (
        <>
          <h2>Une erreur est survenue</h2>
          <p>
            Une erreur est survenue lors du traitement de votre paiement.
            Veuillez réessayer.
          </p>
          <Link className="error-success-link" to="/shopping-cart">
            Retour au panier
          </Link>
        </>
      )}
    </main>
  );
}
```

ETAPE 5 CONFIRMATION

WEBHOOK

- après le paiement, Stripe appelle le serveur via un webhook
- vérification que l'appel provient de Stripe grâce à stripe_signature et STRIPE_WEBHOOK_SECRET
- si le paiement est validé → vérification si une transaction existe déjà
 - si oui: mise à jour
 - si non: nouvelle entrée
- statut de la commande = COMPLETED

```
async webhook(req: Request, res: Response) {
  const stripeSignature = req.headers['stripe-signature'];
  const endpointSecret = process.env.STRIPE_WEBHOOK_SECRET;

  if (!stripeSignature || !endpointSecret) {
    return res.status(400).json({ error: 'Missing signature or endpoint secret' });
  }

  try {
    const event = stripe.webhooks.constructEvent(req.body, stripeSignature, endpointSecret);

    switch (event.type) {
      case 'payment_intent.created':
        const createdPayment = event.data.object as any;
        console.log(`Payment intent created: ${createdPayment.id}`);
        break;

      case 'payment_intent.succeeded':
        const paymentIntent = event.data.object as any;

        // Vérifier si la transaction existe déjà
        const existingPayment = await paymentModel.findById(paymentIntent.id);

        if (existingPayment) {
          // Mettre à jour la transaction existante
          await paymentModel.updateByStripeId(paymentIntent.id, {
            status: PaymentStatus.COMPLETED
          });
        } else {
          // Créer une nouvelle transaction
          await paymentModel.create({
            order_id: paymentIntent.metadata.order_id,
            stripe_payment_id: paymentIntent.id,
            amount: paymentIntent.amount / 100,
            status: PaymentStatus.COMPLETED
          });
        }
      }

      const order = await orderModel.updateById(paymentIntent.metadata.order_id, {
        status: OrderStatus.COMPLETED
      });

      if (order) {
        const user = await userModel.findById(order.user_id);
        if (user && user.email) {
          const orderLines = await orderLineModel.findByOrderIdWithTreeDetails(order.order_id!);
          const totalAmount = await orderLineModel.calculateOrderTotal(order.order_id!);

          try {
            const emailResult = await emailService.sendInvoiceEmail({
              email: user.email,
              firstName: user.first_name,
              lastName: user.last_name,
              orderId: order.order_id!,
              paymentIntentId: paymentIntent.id,
              orderLines: orderLines,
              totalAmount: totalAmount
            });

            if (emailResult.success) {
              console.log('Email de facture envoyé avec succès:', emailResult.messageId);
            }
          } catch (emailError) {
            console.error('Erreur envoi email de facture:', emailError);
          }
        }
      }
    }

    console.log(`Payment succeeded: ${paymentIntent.id}`);
    console.log('paymentIntent', paymentIntent);
    break;
  }
}
```

ETAPE 5 CONFIRMATION

WEBHOOK

- si le paiement échoue
 - transaction = FAILED
 - commande = CANCELED
- payment_intent.requires_action
indique que Stripe attend une action,
comme une authentification 3D Secure
→ ici juste logguée.

```
case 'payment_intent.payment_failed':  
  const failedPayment = event.data.object as any;  
  
  // Vérifier si la transaction existe déjà  
  const existingFailedPayment = await paymentModel.findById(failedPayment.id);  
  
  if (existingFailedPayment) {  
    // Mettre à jour la transaction existante  
    await paymentModel.updateByStripeId(failedPayment.id, {  
      status: PaymentStatus.FAILED  
    });  
  } else {  
    // Créer une nouvelle transaction  
    await paymentModel.create({  
      order_id: failedPayment.metadata.order_id,  
      stripe_payment_id: failedPayment.id,  
      amount: failedPayment.amount / 100,  
      status: PaymentStatus.FAILED  
    });  
  }  
  
  await orderModel.findByIdAndUpdate(failedPayment.metadata.order_id, {  
    status: OrderStatus.CANCELLED  
  });  
  
  console.log(`Payment failed: ${failedPayment.id}`);  
  break;  
  
case 'payment_intent.requires_action':  
  const actionRequired = event.data.object as any;  
  console.log(`Payment requires action: ${actionRequired.id}`);  
  break;  
  
default:  
  console.log(`Unhandled event type ${event.type}`);  
  
  res.json({ received: true });  
} catch (error) {  
  console.error('Webhook error:', error);  
  res.status(400).json({ error: 'Webhook error' });  
}  
,
```

DEMO

AUTHENTIFICATION

The screenshot displays a web application interface for 'GreenRoots'. At the top, there is a navigation bar with links for 'Accueil', 'Catalogue', and 'Notre pépinière'. On the right side of the navigation bar are icons for a shopping bag and a user profile. Below the navigation bar, the main content area has a light beige background. The title 'Connexion' is centered at the top of this area. A red error message 'Email ou mot de passe incorrect' is displayed below the title. There are two input fields: one for 'Email' containing 'jean@email.com' and another for 'Mot de passe' containing a single character 'I'. Below these fields is a green button labeled 'Se connecter'. At the bottom left of the main content area, there is a link 'Mot de passe oublié?'. The footer of the page is dark green and contains several links: 'Notre entreprise', 'Catalogue', 'Notre pépinière', 'Blog nature', 'Ressources', 'CGV', 'Mentions légales', and 'Politique de confidentialité'. At the very bottom of the page, there is a small note about energy consumption: 'Pépinière éco-responsable • 100% local • 0 pesticide' and 'Ce site consomme 8x moins d'énergie que la moyenne des e-commerce. [En savoir plus](#)'.

DEMO ACHAT

GreenRoots

Accueil Catalogue Notre pépinière

Nos arbres - Monde entier

Chêne
15.50 €
France
Reforestation Bretagne

- 1 + Ajouter au panier

Eucalyptus
30.00 €
France
Reforestation Bretagne

- 1 + Ajouter au panier

Baobab
25.00 €
Madagascar
Sauvegarde Baobabs

- 1 + Ajouter au panier



SECURITE

ATTAQUES XSS CROSS SITE SCRIPTING

HELMET

- ajoute des **en-têtes HTTP de sécurité** à chaque réponse
- **empêche l'injection de scripts malveillants** dans le navigateur

COOKIES

- tokens JWT stockés dans un cookie
- **httpOnly**: empêche tout accès JS au cookie
- **secure**: le cookie n'est transmis que via HTTPS
- **sameSite**: restreint les envois de cookies aux requêtes provenant du même site

VALIDATION DES DONNÉES

- Avant toute connexion ou création:
- champs des formulaires (email, prénom, nom, etc.) sont validés et nettoyés par **express-validator**

MESSAGES D'ERREURS

- génériques: "Email ou mot de passe incorrects"
- **ne donne pas d'indications**



SECURITE

ATTAQUES CSRF CROSS SITE REQUEST FORGERY

RÔLES ET MIDDLEWARES

Routes protégées:

- actions sensibles (comme getAllUsers, updateUserRole)
réservées aux administrateurs
- **middlewares d'authentification** vérifient systématiquement le JWT avant d'accéder aux ressources

COOKIES

- **sameSite**: 'strict' → empêche les requêtes provenant d'autres domaines d'envoyer automatiquement les cookies

RATE LIMITING ET CORS

- **rate limiter** = restreint chaque IP à 100 requêtes / 15 min → limitant les attaques répétées ou automatisées
- **CORS**: n'autorisent que les origines front-end légitimes à interagir avec l'API → empêche les sites externes de faire des requêtes



SECURITE INJECTIONS SQL

REQUÊTES PRÉPARÉES

toutes les interactions SQL passent par des **requêtes préparées**
→ neutralise toute tentative d'injection SQL dans les champs utilisateurs

VALIDATION ET FILTRAGE DES ENTRÉES

Lors de l'enregistrement ou de la connexion:

- **données saisies validées** avant d'être transmises au controller

→ empêche l'exécution de requêtes malveillantes

HASHAGE ET STOCKAGE SÉCURISÉ DES MOTS DE PASSE

mots de passe jamais stockés en clair

- **Hachage** avec Argon2id (robuste et résistant aux attaques par force brute)
- paramètres:
 - **memoryCost**: 2^{16} (mémoire élevée)
 - **timeCost**: 3 (ralentit les tentatives)
 - **parallelism**: 1



SECURITE
PAIEMENT

STRIPE

- paiement géré par Stripe → **les données bancaires ne transitent jamais sur le serveur**
- webhooks Stripe signés et vérifiés pour prévenir toute falsification

TESTS UNITAIRES

- utilisation de Vitest
- test directement dans le dossier du composant à tester

```
import {describe, expect, vi, test} from 'vitest';
import { render, screen, fireEvent } from "@testing-library/react"
import { QuantitySelector } from './quantitySelector';

describe("QuantitySelector", () => {
    test("increments value if it's less than max", async () => {
        const onChange = vi.fn()
        const { unmount } = render(
            <QuantitySelector value={1} max={10} onChange={onChange} />
        )
        fireEvent.click(await screen.findByLabelText("Augmenter la quantité"))
        expect(onChange).toHaveBeenCalled();
        unmount();
    })

    test("does not increment value if it's higher or equal to max", async () => {
        const onChange = vi.fn()
        render(
            <QuantitySelector value={10} max={10} onChange={onChange} />
        )
        fireEvent.click(await screen.findByLabelText("Augmenter la quantité"))
        expect(onChange).not.toHaveBeenCalled();
    })
})|
```

TESTS CI/CD

- tests backend réalisés avec Jest
- regroupés dans un dossier /tests
- exemple: test de la logique du contrôleur treeController
- configuration de GitHub Actions pour qu'ils s'exécutent automatiquement

```
✓ tests
  JS auth.test.js
  JS connection.test.js
  JS database.test.js
  JS errorHandler.test.js
  {} insomnia_auth_collection.json
  JS localizationController.test.js
  JS projectController.test.js
  JS security.test.js
  JS treeController.test.js
  JS validation.test.js
```

```
// tests/treeController.test.js
import { jest, describe, it, beforeEach, expect } from '@jest/globals';

// Mock TreeModel
const mockTreeModel = {
  findAllWithProjectsAndLocalizations: jest.fn(),
  findByIdWithProjectsAndLocalizations: jest.fn()
};

// Create a mock controller that simulates your actual controller logic
const createMockController = () => {
  async index(req, res) {
    try {
      const page = parseInt(req.query.page) || 1;
      const limit = parseInt(req.query.limit) || 10;
      const offset = (page - 1) * limit;

      // Get all trees with projects and localizations
      const allTrees = await mockTreeModel.findAllWithProjectsAndLocalizations();

      // Apply pagination manually
      const total = allTrees.length;
      const trees = allTrees.slice(offset, offset + limit);

      res.json({
        message: 'Trees retrieved successfully',
        data: trees,
        pagination: {
          page,
          limit,
          total,
          pages: Math.ceil(total / limit)
        },
        status: 200
      });
    } catch (error) {
      res.status(500).json({
        message: 'Error retrieving trees',
        error: error instanceof Error ? error.message : 'Unknown error',
        status: 500
      });
    }
  }
},
```

DEPLOIEMENT CI/CD

- création d'un fichier .github/workflows/deploy.yml
- réagit à chaque push ou pull request sur la branche main
- site hébergé sur Digital Ocean

ETAPES

1

TEST:

- lancement des tests backend
- tables de la base de données initialisées
- build du front pour vérifier qu'il se compile

2

BUILD AND PUSH:

- création des images Docker pour le front et le back
- images poussées vers GitHub Container Registry

3

DEPLOY:

- projet déployé sur un serveur distant via SSH
- certificats SSL gérés automatiquement
- services Docker redémarrés

DEPLOIEMENT CI/CD

DECLENCHEMENT CI/CD

→ à chaque push / pull request sur main

```
on:  
  push:  
    branches: [ main ]  
  pull_request:  
    branches: [ main ]
```

TESTS

→ le script lançant les tests est exécuté

```
- name: Run backend tests  
  run:  
    cd @back  
    pnpm test
```

DEPLOY

→ extrait des 1ères étapes

- pull pour récupérer le code
- mise à jour des variables .env

```
deploy:  
  needs: [test, build-and-push]  
  runs-on: ubuntu-latest  
  if: github.ref == 'refs/heads/main'  
  
  environment: production  
  
  steps:  
    - name: Deploy to server  
      uses: appleboy/ssh-action@v1.0.3  
      with:  
        host: ${{ secrets.HOST }}  
        username: ${{ secrets.USERNAME }}  
        key: ${{ secrets.PRIVATE_KEY }}  
        port: ${{ secrets.PORT }}  
        script:  
          cd /home/ubuntu/greenroots  
  
          # Pull latest code  
          git pull origin ${github.ref_name}  
  
          # Create/update .env file with production values  
          cat > .env << EOF  
          VITE_API_URL=${secrets.VITE_API_URL}
```

RGPD

PRINCIPALES MESURES

EXTRAIT DU FOOTER

Notre entreprise	Ressources
Catalogue	CGV
Notre pépinière	Mentions légales
Politique de confidentialité	

01

INFORMER LES UTILISATEURS

- **types de données** collectées
- **finalités** (facturation, service client)
- **bases légales** du traitement (contrat, consentement, obligation légale, intérêt légitime)
- **durée de conservation** des données
- **droits des utilisateurs** (accès, rectification, suppression, opposition, portabilité)
- **destinataires des données** (ex. prestataires de paiement, transporteurs)
- **coordonnées** d'un responsable RGPD

RGPD

PRINCIPALES MESURES

EXTRAIT DU FOOTER

Notre entreprise	Ressources
Catalogue	CGV
Notre pépinière	Mentions légales Politique de confidentialité

01

INFORMER LES UTILISATEURS

- **types de données** collectées
- **finalités** (facturation, service client)
- **bases légales** du traitement (contrat, consentement, obligation légale, intérêt légitime)
- **durée de conservation** des données
- **droits des utilisateurs** (accès, rectification, suppression, opposition, portabilité)
- **destinataires des données** (ex. prestataires de paiement, transporteurs)
- **coordonnées** d'un responsable RGPD

The screenshot shows a website header with the logo "GreenRoots" (a stylized plant icon) and navigation links for "Accueil", "Catalogue", "Notre pépinière", a shopping cart icon, and a user profile icon.

Politique de Confidentialité

1. Introduction

GreenRoots s'engage à protéger la vie privée de ses utilisateurs. Cette politique de confidentialité explique comment nous collectons, utilisons, stockons et protégeons vos données personnelles conformément au Règlement Général sur la Protection des Données (RGPD) et à la loi Informatique et Libertés.

En utilisant notre site, vous acceptez les pratiques décrites dans cette politique.

2. Responsable du traitement

Le responsable du traitement de vos données personnelles est :

GreenRoots SAS
Siège social : 42 Avenue de la Forêt, 69003 Lyon, France
Email : hello@greenroots.website

3. Données collectées

Nous collectons les données personnelles suivantes :

3.1. Données d'identification

- Nom et prénom
- Adresse email
- Numéro de téléphone (optionnel)

RGPD

PRINCIPALES MESURES

01

INFORMER LES UTILISATEURS

- **types de données** collectées
- **finalités** (facturation, service client)
- **bases légales** du traitement (contrat, consentement, obligation légale, intérêt légitime)
- **durée de conservation** des données
- **droits des utilisateurs** (accès, rectification, suppression, opposition, portabilité)
- **destinataires des données** (ex. prestataires de paiement, transporteurs)
- **coordonnées** d'un responsable RGPD

02

RESPECTER LES DROITS DES PERSONNES

- **donner accès aux données** d'un utilisateur sur demande
- **corriger ou supprimer** ses données
- **fournir une copie** des données sur demande
- **permettre l'opposition** au traitement (désabonnement newsletter)

RGPD

PRINCIPALES MESURES

01

INFORMER LES UTILISATEURS

- **types de données** collectées
- **finalités** (facturation, service client)
- **bases légales** du traitement (contrat, consentement, obligation légale, intérêt légitime)
- **durée de conservation** des données
- **droits des utilisateurs** (accès, rectification, suppression, opposition, portabilité)
- **destinataires des données** (ex. prestataires de paiement, transporteurs)
- **coordonnées** d'un responsable RGPD

02

RESPECTER LES DROITS DES PERSONNES

- **donner accès aux données** d'un utilisateur sur demande
- **corriger ou supprimer** ses données
- **fournir une copie** des données sur demande
- **permettre l'opposition** au traitement (désabonnement newsletter)

03

SÉCURISER LES DONNÉES

- site en **HTTPS**
- **stockage chiffré** des mots de passe
- **accès limité** aux données

RGPD

PRINCIPALES MESURES

INFORMER LES UTILISATEURS

- **types de données** collectées
- **finalités** (facturation, service client)
- **bases légales** du traitement (contrat, consentement, obligation légale, intérêt légitime)
- **durée de conservation** des données
- **droits des utilisateurs** (accès, rectification, suppression, opposition, portabilité)
- **destinataires des données** (ex. prestataires de paiement, transporteurs)
- **coordonnées** d'un responsable RGPD

RESPECTER LES DROITS DES PERSONNES

- **donner accès aux données** d'un utilisateur sur demande
- **corriger ou supprimer** ses données
- **fournir une copie** des données sur demande
- **permettre l'opposition** au traitement (désabonnement newsletter)

SÉCURISER LES DONNÉES

- site en **HTTPS**
- **stockage chiffré** des mots de passe
- **accès limité** aux données

PROCÉDURE EN CAS DE VIOLATION

- **notifier la CNIL** dans les 72h en cas de fuite ou d'accès non autorisé aux données
- **informer les utilisateurs** concernés

01

02

03

04

REFERENCLEMENT SEO

React Router:

- **fonction meta** permet de définir des métadonnées dans le head comme:
 - un titre
 - un contenu
- **génère les URLs** qui sont définies dans le dossier routes
- **rend une page HTML complète** qui bénéficie des balises HTML et de leur sémantique

```
export function meta() {
  return [
    {
      title: "GreenRoots - catalogue d'arbres",
    },
    {
      name: "description",
      content:
        "Découvrez notre sélection d'arbres à parrainer dans nos projets de reforestation à travers le monde. Choisissez une espèce, un lieu et participez dès aujourd'hui à la lutte contre la déforestation.",
    },
  ];
}
```

ACCESSIBILITE MESURES

- contrastes de couleurs respectent les ratios
- images ont l'attribut alt
- utilisation des rôles ARIA

```
<img src={image} alt={'${name} - Arbre de ${localization}'}/>
```

```
/* cart messages */
{cartMessage && (
  <div
    className="cart-message cart-success"
    role="status"
    aria-live="polite"
  >
    ✓ {cartMessage}
  </div>
)
{cartError && (
  <div
    className="cart-message cart-error"
    role="alert"
    aria-live="assertive"
  >
    ▲ {cartError}
  </div>
)}
```

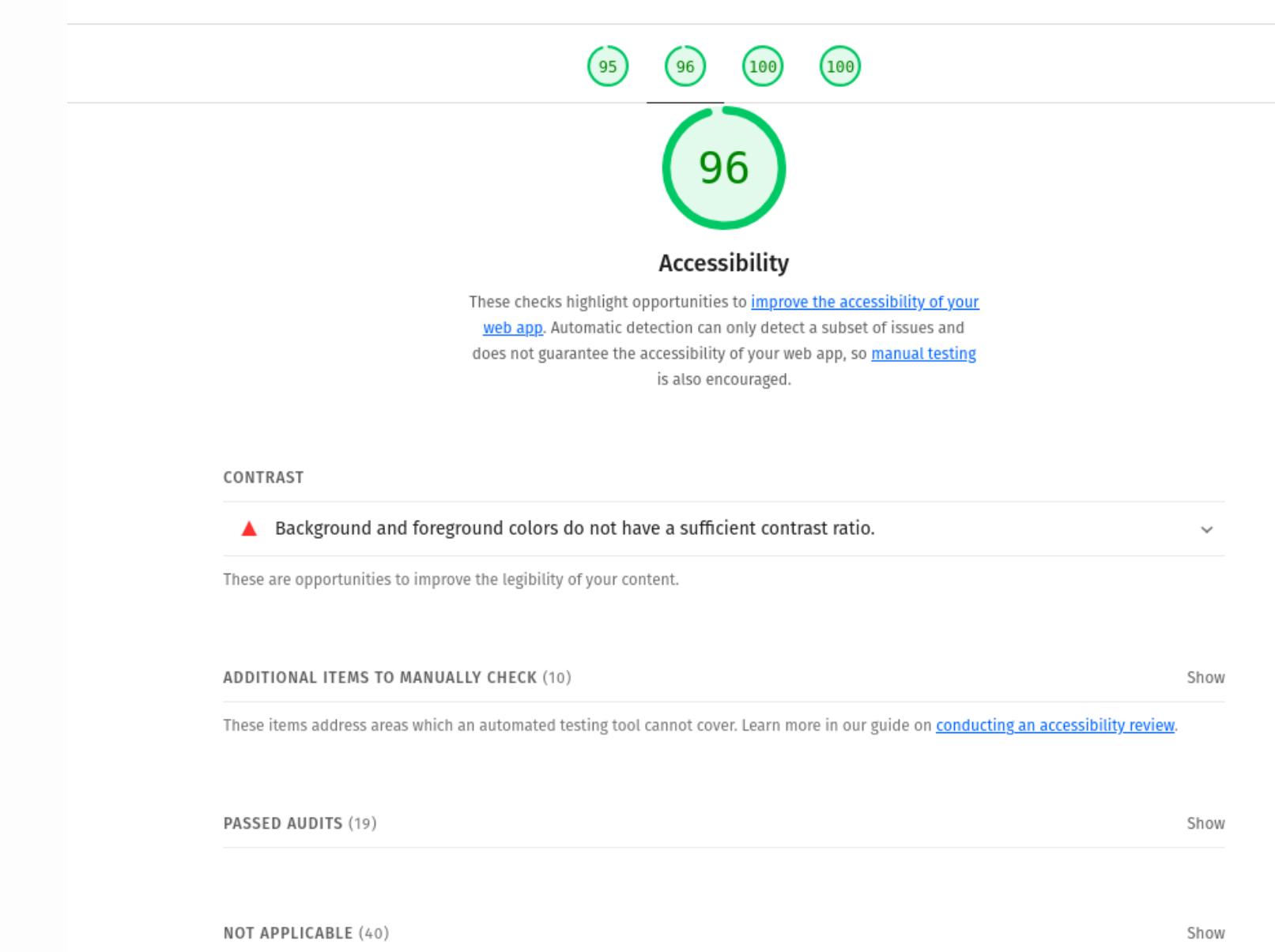
ACCESSIBILITE MESURES

- contrastes de couleurs respectent les ratios
- images ont l'attribut alt
- utilisation des rôles ARIA
- navigation au clavier

```
return (
  <nav className={className}>
    <button
      type="button"
      onClick={() => {
        setOpen((previousOpen) => !previousOpen);
      }}
      aria-label="Ouvrir le menu"
      className="toggle-btn"
    >
      <span className="menu-icon">{open ? "✖" : "☰"}</span>
    </button>
    {open ? (
      <ul className="mobile-menu">
        {navLinks.map((link) => (
          <li key={link.to}>
            {link.subLinks ? (
              <div className="mobile-submenu-container">
                <Link to={link.to} onClick={() => setOpen(false)} className="mobile-main-link">
                  {link.label}
                </Link>
                <details className="mobile-details">
                  <summary className="mobile-summary">Continents</summary>
                  <ul className="mobile-submenu">
                    <li>
                      <Link to="/catalog" onClick={() => setOpen(false)} className="mobile-submenu-link">
                        Voir tout
                      </Link>
                    </li>
                    {link.subLinks.map((sub) => (
                      <li key={sub.to}>
                        <Link to={sub.to} onClick={() => setOpen(false)} className="mobile-submenu-link">
                          {sub.label}
                        </Link>
                      </li>
                    )));
                  </ul>
                </details>
              </div>
            ) : (
              <Link to={link.to} onClick={() => setOpen(false)} className="mobile-main-link">
                {link.label}
              </Link>
            )}
          </li>
        ));
      </ul>
    ) : (
      <ul className="mobile-menu">
        {navLinks.map((link) => (
          <li key={link.to}>
            <Link to={link.to} onClick={() => setOpen(false)} className="mobile-main-link">
              {link.label}
            </Link>
          </li>
        ));
      </ul>
    )}
  </nav>
)
```

ACCESSIBILITE MESURES

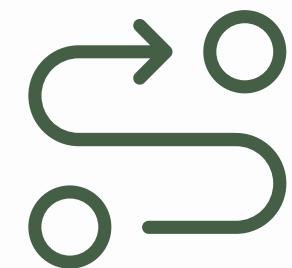
- contrastes de couleurs respectent les ratios
- images ont l'attribut alt
- utilisation des rôles ARIA
- navigation au clavier
- audit avec LightHouse



CONCLUSION

DIFFICULTES:

- conteneurisation
- branchement de Stripe
- gestion du temps



Gestion de projet

- 3 sprints intenses
- nécessité de redéfinir les objectifs
- heureuse de retrouver ma casquette de chef de projet

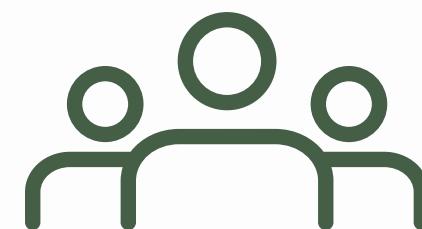
REUSSITES:

- finalisation de la fonctionnalité principale
- rendu global de GreenRoots agréable



Technologies

- application de tout ce qui a été vu pendant la formation
- découverte de nouvelles technologies: React Router et Stripe



Equipe

- aventure humaine
- gestion du stress, des différentes compétences, appétences



A dense forest of tall evergreen trees, heavily shrouded in thick, grey mist. The scene is atmospheric and suggests a cool, damp environment.

MERCI

DE VOTRE ATTENTION



www.greenroots-website



BONUS

DOC README

- documente les différents scripts
 - commandes principales pour lancer le projet
 - production
 - gestion de docker
 - lancer le front ou le back individuellement
 - consultation des logs
 - accès aux conteneurs

README

GreenRoots

Commandes Docker

Ce projet utilise Docker Compose avec différentes configurations pour les environnements de développement et de production.

Environnement de développement

Commandes principales

```
npm run dev          # Lance l'application en mode développement
npm run dev:build   # Lance l'application en reconstruisant les images
npm run dev:build:fresh # Reconstruit complètement (sans cache) et lance
npm run dev:down    # Arrête tous les services de développement
npm run dev:logs    # Affiche les logs en temps réel
```

Environnement de production

Commandes principales

```
npm run prod         # Lance l'application en mode production
npm run prod:build   # Lance l'application en reconstruisant les images
npm run prod:build:fresh # Reconstruit complètement (sans cache) et lance
npm run prod:down    # Arrête tous les services de production
```

DOCKER COMPOSE DEV

- Database:
 - Lance PostgreSQL
 - Configure DB, user, password via variables d'environnement
 - Monte le volume db_data pour persister les données
 - Initialise la DB
 - Healthcheck : attend que PostgreSQL soit prêt avant de lancer le backend
 - Expose le port 5432 localement

```
services:  
  database:  
    image: postgres:15  
    restart: unless-stopped  
    environment:  
      - POSTGRES_DB=${POSTGRES_DB}  
      - POSTGRES_USER=${POSTGRES_USER}  
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}  
    ports:  
      - "5432:5432"  
    volumes:  
      - db_data:/var/lib/postgresql/data  
      - ./back/database/migration/create-tables.sql:/docker-entrypoint-initdb.d/01-create-tables.sql  
      - ./back/database/seeds/sample-data.sql:/docker-entrypoint-initdb.d/02-sample-data.sql  
  
    healthcheck:  
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]  
      interval: 5s  
      timeout: 3s  
      retries: 5
```

DOCKER COMPOSE DEV

- Backend:
 - Build depuis ./@back
 - Dépend de la DB
 - Monte le code source pour hot-reload (./@back:/app) → modification instantanée sans rebuild
 - Expose le port 3001
 - Variables d'environnement pour Stripe, DB, etc
 - develop.watch : synchronise les changements du code et rebuild si nécessaire

```
backend:  
  build:  
    context: ./@back  
    dockerfile: Dockerfile.dev  
  restart: unless-stopped  
  ports:  
    - "3001:3001"  
  depends_on:  
    database:  
      condition: service_healthy  
  environment:  
    - NODE_ENV=${NODE_ENV}  
    - DATABASE_URL=${DATABASE_URL}  
    - STRIPE_PUBLIC_KEY=${STRIPE_PUBLIC_KEY}  
    - STRIPE_SECRET_KEY=${STRIPE_SECRET_KEY}  
    - STRIPE_WEBHOOK_SECRET=${STRIPE_WEBHOOK_SECRET}  
  volumes:  
    - ./@back:/app  
    - /app/node_modules  
  develop:  
    watch:  
      - action: sync  
        path: ./@back  
        target: /app  
        ignore:  
          - node_modules/  
      - action: rebuild  
        path: ./@back/package.json
```

DOCKER COMPOSE DEV

- Frontend:
 - Build depuis ./@front
 - Dépend du backend

```
frontend:
  build:
    context: ./@front
    dockerfile: Dockerfile.dev
  restart: unless-stopped
  ports:
    - "3000:3000"
  depends_on:
    - backend
  volumes:
    - ./@front:/app
    - /app/node_modules
  develop:
    watch:
      - action: sync
        path: ./@front
        target: /app
        ignore:
          - node_modules/
      - action: rebuild
        path: ./@front/package.json

  volumes:
    db_data:
```

DOCKER COMPOSE PROD

- Database:
 - Lance PostgreSQL
 - Configure DB, user, password via variables d'environnement
 - Monte le volume db_data pour persister les données
 - Initialise la DB
 - Expose un port configurable (DB_PORT)

```
services:
  database:
    image: postgres:15
    environment:
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    ports:
      - "${DB_PORT:-5433}:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
      - ./back/database/migration/create-tables.sql:/docker-entrypoint-initdb.d/01-create-tables.sql
      - ./back/database/seeds/prod-data.sql:/docker-entrypoint-initdb.d/02-prod-data.sql
```

DOCKER COMPOSE PROD

- Backend:
 - Build depuis ./@back
 - Dépend de la DB
 - Expose le port 3001
 - Variables d'environnement pour Stripe, DB, etc

```
backend:  
  build:  
    context: ./@back  
  ports:  
    - "3001:3001"  
  depends_on:  
    - database  
  environment:  
    - NODE_ENV=production  
    - DATABASE_URL=${DATABASE_URL}  
    - DB_HOST=${DB_HOST}  
    - DB_USER=${DB_USER}  
    - DB_PASSWORD=${DB_PASSWORD}  
    - DB_NAME=${DB_NAME}  
    - DB_PORT=5432  
    - STRIPE_PUBLIC_KEY=${STRIPE_PUBLIC_KEY}  
    - STRIPE_SECRET_KEY=${STRIPE_SECRET_KEY}  
    - STRIPE_WEBHOOK_SECRET=${STRIPE_WEBHOOK_SECRET}  
    - REDIRECT_CHECKOUT_PAGE=${REDIRECT_CHECKOUT_PAGE}  
    - SMTP_HOST=${SMTP_HOST}  
    - SMTP_PORT=${SMTP_PORT}  
    - EMAIL_USER=${EMAIL_USER}  
    - EMAIL_PASS=${EMAIL_PASS}  
    - EMAIL_FROM=${EMAIL_FROM}  
    - EMAIL_FROM_NAME=${EMAIL_FROM_NAME}
```

DOCKER COMPOSE PROD

- Frontend:
 - Build depuis ./@front
 - Dépend du backend
 - Expose le port 3000
 - Variables d'environnement pour l'API et redirections

```
frontend:
  build:
    context: ./@front
    args:
      - VITE_API_URL=${VITE_API_URL}
      - REDIRECT_CHECKOUT_PAGE=${REDIRECT_CHECKOUT_PAGE}
  ports:
    - "3000:3000"
  depends_on:
    - backend
  environment:
    - VITE_API_URL=${VITE_API_URL}
    - REDIRECT_CHECKOUT_PAGE=${REDIRECT_CHECKOUT_PAGE}
```

DOCKER COMPOSE PROD

- Nginx:
 - Sert le frontend et reverse-proxy vers le backend
 - Gère le HTTPS avec Certbot
 - Rechargement Nginx automatique toutes les 6h
- Certbot:
 - Renouvelle automatiquement les certificats HTTPS tous les 12h

```
nginx:
  image: nginx:alpine
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx.conf:/etc/nginx/conf.d/default.conf
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  depends_on:
    - frontend
    - backend
  command: "/bin/sh -c 'while :; do sleep 6h & wait $$!; nginx -s reload; done & nginx -g \"daemon off;\"'"
```

```
certbot:
  image: certbot/certbot
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h & wait $$!; done;'"
```

```
volumes:
  db_data:
```

BASE DE DONNÉES CREATE

```
1 BEGIN;
2
3 -- Suppression des tables existantes
4 DROP TABLE IF EXISTS payment_transaction;
5 DROP TABLE IF EXISTS planted_tree;
6 DROP TABLE IF EXISTS order_line;
7 DROP TABLE IF EXISTS "order";
8 DROP TABLE IF EXISTS "user";
9 DROP TABLE IF EXISTS project_tree;
10 DROP TABLE IF EXISTS project;
11 DROP TABLE IF EXISTS tree;
12 DROP TABLE IF EXISTS localization;
13 DROP TABLE IF EXISTS password_reset_tokens;
14 DROP TABLE IF EXISTS email_verification_tokens;
15
16 CREATE TABLE localization (
17   localization_id SERIAL PRIMARY KEY,
18   country TEXT NOT NULL,
19   continent TEXT NOT NULL
20 );
21
22 CREATE TABLE tree (
23   tree_id SERIAL PRIMARY KEY,
24   name TEXT NOT NULL,
25   description TEXT,
26   price DECIMAL(10, 2),
27   image TEXT,
28   price_id TEXT,
29   created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,
30   updated_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
31 );
32
33 CREATE TABLE project (
34   project_id SERIAL PRIMARY KEY,
35   localization_id INTEGER NOT NULL,
36   name TEXT NOT NULL,
37   description TEXT,
38   image TEXT,
39   created_at DATE DEFAULT CURRENT_DATE,
40   updated_at DATE DEFAULT CURRENT_DATE,
41   CONSTRAINT fk_localization FOREIGN KEY (localization_id)
42     REFERENCES localization(localization_id)
43     ON DELETE RESTRICT
44 );
45
```

BASE DE DONNÉES SEEDING

```
1  -- Données initiales de base
2  INSERT INTO localization (country, continent) VALUES
3  ('France', 'Europe'),
4  ('Madagascar', 'Afrique'),
5  ('Brésil', 'Amérique du Sud');
6
7  INSERT INTO tree (name, description, price, image) VALUES
8  ('Chêne', 'Le chêne est un arbre majestueux, reconnu pour sa longévité et ses feuilles lobées. Il pousse dans les régions tempérées et peut atteindre plusieurs dizaines de mètres. Ses glands nourrissent les animaux et son bois solide sert à la construction, au mobilier et aux tonneaux. Il favorise biodiversité et ombre.', 15.50, '/assets/images/trees/chene.webp'),
9  ('Baobab', 'Le baobab est un arbre majestueux surnommé "arbre de vie". Son tronc massif stocke l'eau pour survivre aux sécheresses. Ses feuilles, fruits et écorce sont utilisés en alimentation et médecine traditionnelle. Il fournit de l'ombre, nourrit de nombreuses espèces et joue un rôle vital dans l''écosystème.', 25.00, '/assets/images/trees/baobab.webp'),
10  ('Eucalyptus', 'L''eucalyptus est un arbre originaire d''Australie, reconnu pour sa croissance rapide et ses feuilles aromatiques. Son bois sert à la construction et au papier, tandis que ses feuilles produisent des huiles essentielles aux vertus médicinales. Il offre un habitat pour la faune et contribue à la biodiversité.', 30.00, '/assets/images/trees/eucalyptus.webp'),
11  ('Palissandre', 'Le palissandre est un arbre tropical réputé pour son bois dense, dur et coloré. Utilisé pour les meubles, l''ébénisterie et les instruments de musique, il se distingue par sa durabilité et son esthétique. Il contribue aussi à la biodiversité en offrant habitat et nourriture à de nombreuses espèces.', 45.00, '/assets/images/trees/palissandre.webp');
12
13  INSERT INTO project (localization_id, name, description, image) VALUES
14  (1, 'Reforestation Bretagne', 'Le projet Reforestation Bretagne ambitionne de restaurer les forêts endémiques et bocages, victimes de l''intensification agricole et des tempêtes. Par la plantation d''essences locales et la régénération naturelle, il agit pour la préservation de la biodiversité bretonne, la lutte contre l''érosion et le retour de corridors écologiques. L''implication des citoyens et agriculteurs fait de ce projet une démarche collective et durable, essentielle pour la résilience du territoire face aux nouveaux défis climatiques.', '/assets/images/projects/reforestation-bretagne.webp'),
15  (2, 'Sauvegarde Baobabs', 'La sauvegarde des baobabs vise à protéger ces arbres emblématiques, essentiels à la survie de nombreuses espèces et communautés d''Afrique. Menacés par le changement climatique, la déforestation et l''urbanisation, les baobabs sont replantés et leur environnement restauré grâce à des actions de sensibilisation locale et à l''appui scientifique. Ce projet favorise la transmission des savoirs, soutient l''agroécologie et garantit la pérennité de ces "arbres de vie" pour les générations futures.', '/assets/images/projects/sauvegarde-baobabs.webp'),
16  (3, 'Forêt Amazonienne', 'Le projet Forêt amazonienne s''attache à préserver le "poumon vert" de la planète face à la déforestation et à l''exploitation non durable. À travers la plantation d''essences natives, la protection des terres indigènes et le suivi de la faune, il soutient la régénération d''écosystèmes uniques et la lutte contre le réchauffement climatique. L''engagement des communautés locales assure la transmission des savoirs et la continuité des forêts, vitales pour l''équilibre mondial.', '/assets/images/projects/foret-amazonienne.webp');
17
18  INSERT INTO project_tree (project_id, tree_id) VALUES
19  -- Projet Reforestation Bretagne (France)
20  (1, 1), -- Chêne
21  (1, 3), -- Eucalyptus
22
23  -- Projet Sauvegarde Baobabs (Madagascar)
24  (2, 2), -- Baobab
25  (2, 4), -- Palissandre
26
```

SQL JOINTURE

```
// Get trees with their associated projects and localizations
async findAllWithProjectsAndLocalizations(): Promise<any[]> {
  try {
    const query = `

      SELECT
        t.tree_id,
        t.name as tree_name,
        t.description as tree_description,
        t.price,
        t.image as tree_image,
        t.price_id,
        t.created_at as tree_created_at,
        t.updated_at as tree_updated_at,
        p.project_id,
        p.name as project_name,
        p.description as project_description,
        p.image as project_image,
        p.localization_id,
        p.created_at as project_created_at,
        p.updated_at as project_updated_at,
        l.country,
        l.continent
      FROM tree t
        LEFT JOIN project_tree pt ON t.tree_id = pt.tree_id
        LEFT JOIN project p ON pt.project_id = p.project_id
        LEFT JOIN localization l ON p.localization_id = l.localization_id
      ORDER BY t.created_at DESC, p.created_at DESC
    `;
    const result = await this.db.query(query);
  }
}
```

SQL PREPAREE

```
async findByIdForCheckout(id: number): Promise<any | null> {
  try {
    const query = `
      SELECT
        t.tree_id,
        t.price,
        t.price_id
      FROM tree t
      WHERE t.tree_id = $1
    `;
    const result = await this.db.query(query, [id]);
  }
}
```