Création d'une Application Web - Full Stack

- Frontend : React, Vite, React Router, TypeScript
- Backend : Node.js, Express.js
- Base de données : PostgreSQL
- Déploiement : Docker, Docker Compose
- Hébergement : Render.com
- Outils & Qualité du Code : Biome, Npm
- Gestion de code & CI/CD : Git, GitHub

1. Initialiser le projet et les dépendances

- 1.01 - Ouvrir Git BASH et se rendre dans le dossier des projets : **cd ~/desktop/LOCAL**
- 1.02 - Créer un dossier racine pour ton projet : **mkdir fullstack-starter-kit**
- 1.03 - Se rendre dans ce dossier : **cd fullstack-starter-kit**

Création du dossier client (frontend) avec Vite

- 1.04 - Créer le dossier client et initialiser Vite : **npm create vite@latest client**
  - Use rolldown-vite (Experimental)? **no**
  - Install with npm and start now? **no**
- 1.05 - Se rendre dans le dossier client : **cd client**
- 1.06 - Installer les dépendances du client : **npm install && npm install -D @types/react**
- 1.07 - Installer le plugin React pour Vite : **npm install -D @vitejs/plugin-react**
- 1.08 - Installer React Router : **npm install react-router && npm install --save-dev @types/react-router**
- 1.09 - Installer react-toastify : **npm install react-toastify && npm install -D @types/react-toastify**
- 1.10 - Installer les types Node pour TypeScript : **npm install --save-dev @types/node**
- 1.11 - Installer les types React et ReactDOM : **npm install --save-dev @types/react @types/react-dom**

Création du dossier server (backend)

- 1.12 - Revenir à la racine du projet : **cd ..**
- 1.13 - Créer le dossier server : **mkdir server**
- 1.14 - Se rendre dans le dossier server : **cd server**
- 1.15 - Initialiser un projet Node : **npm init -y**
- 1.16 - Installer Express : **npm install express && npm install -D @types/express**
- 1.17 - Installer TypeScript et outils dev : **npm install --save-dev typescript tsx @types/node @types/express**
- 1.18 - Installer dotenv : **npm install dotenv**
- 1.19 - Installer PostgreSQL : **npm install pg && npm install -D @types/pg && npm install -D @types/node**
- 1.20 - Installer cors : **npm install cors && npm install -D @types/cors**
- 1.21 - Installer bcrypt : **npm install bcrypt && npm install -D @types/bcrypt**
- 1.22 - Installer cookie parser : **npm install cookie-parser @types/cookie-parser**
- 1.23 - Installer jsonwebtoken : **npm install jsonwebtoken && npm install --save-dev @types/jsonwebtoken**

Installation les outils pour le projet full stack

- 1.24 - Revenir à la racine du projet : **cd ..**
- 1.25 - Installer concurrently : **npm install concurrently@latest --save-dev**
- 1.26 - Installer Biome en tant que dépendance de développement : **npm install -D @biomejs/biome**
- 1.27 - Initialiser la configuration Biome : **npx biome init**
- 1.28 - Ouvrir Vs Code : **code .**

2. Préparer la structure RACINE de l'application

- 2.01 - Racine > Créer un dossier app_ressources
- 2.02 - Racine > app_ressources > Créer un fichier readme.md
- 2.03 - Racine > app_ressources > README.md > Coller le contenu suivant :

```
### app_ressources

Ce dossier contient les fichiers **annexes** au projet qui ne font pas directement partie du code source.
**Ce contenu n'est pas versionné** et est destiné uniquement à un usage local.

- **Documents PDF / Word** - Textes à intégrer dans le site, conditions générales, contrats, guides, notices
- **Screenshots / Maquettes** - Aperçus de design ou d'interface, références visuelles pour le développement
- **Documents de travail** - Cahier des charges, spécifications techniques ou fonctionnelles, idées, notes de projet, to-do lists
- **Tableaux Excel / CSV** - Liste de produits, services, utilisateurs, etc.
- **Brouillons** - Fichiers non finalisés ou archivés pour consultation ultérieure

---

Code couleur présent dans le code :
🟢 Succès / Mis à jour : Test réussi, correction appliquée.
🔴 Erreur / Défaut : Problème détecté, nécessite intervention.
```

- 2.04 - Racine > Créer un fichier .gitignore
- 2.05 - Racine > .gitignore > Coller le contenu suivant :

```
# Global
node_modules
.env
dist
build
.DS_Store
.vscode
.idea
*.log
npm-debug.log*

# Vite (client)
client/dist
client/.vite

# TypeScript (server)
server/dist
server/*.tsbuildinfo

# Ressources locales (non versionnées)
app_ressources/

# Logs
logs
*.log
```

- 2.06 - Racine > Créer un fichier package.json
- 2.07 - Racine > package.json > Coller le contenu suivant :

```
{
  "name": "projet",
  "version": "1.0.0",
```

```
  "private": true,
  "type": "module",
  "scripts": {
    "predev": "node server/database/init-db.js",
    "dev": "concurrently -c green,yellow -t \"HH:mm:ss\" -p \"{name} {time}\" \"npm run dev --prefix server\" \"npm run dev --prefix client\"",
    "dev:client": "npm run dev --prefix client -- --host",
    "dev:server": "npm run dev --prefix server",
    "build:client": "npm run build --prefix client",
    "start": "npm run start --prefix server",
    "setup": "node setup/update-package-name.js",
    "postinstall": "npm run setup",
    "setup:project": "node setup/update-package-name.js",
    "render-build": "npm install --prefix client && npm run build --prefix client && npm run build --prefix server"
  },
  "devDependencies": {
    "@biomejs/biome": "^2.3.8",
    "concurrently": "^9.2.1",
    "vite": "^7.2.4",
    "@vitejs/plugin-react": "^5.1.2"
  }
}
```

- 2.08 - Racine > Créer un fichier tsconfig.json
- 2.09 - Racine > tsconfig.json > Coller le contenu suivant :

```
{
  "files": [],
  "references": [{ "path": "./client" }, { "path": "./server" }],
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "lib": ["ES2022"],
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "baseUrl": "."
  }
}
```

- 2.10 - Racine > Créer un fichier docker-compose.yml
- 2.11 - Racine > docker-compose.yml > Coller le contenu suivant :

```YAML
services:
  postgres:
    image: postgres:16
    container_name: postgres
    restart: always
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
```

```
      - "5432:5432"
    volumes:
      - ./setup/init.sql:/docker-entrypoint-initdb.d/init.sql
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata:
```

- 2.12 - Racine > Créer un dossier setup
- 2.13 - Racine > setup > Créer un fichier update-package-name.js
- 2.14 - Racine > setup > update-package-name.js > Coller le contenu suivant :

```
import fs from "node:fs";
import path from "node:path";

const rootFolderName = path.basename(path.resolve());

const constantsPath = path.resolve("setup/constants.ts");
if (fs.existsSync(constantsPath)) {
  let constantsContent = fs.readFileSync(constantsPath, "utf8");

  if (constantsContent.includes("PENDING_GENERATION")) {
    constantsContent = constantsContent.replace(/PENDING_GENERATION/g, rootFolderName);
    fs.writeFileSync(constantsPath, constantsContent);
    console.log(`🟢 constants.ts mis à jour avec : ${rootFolderName}`);
  } else {
    console.log("🟢 Aucune occurrence 'PENDING_GENERATION' trouvée dans constants.ts");
  }
} else {
  console.log("🟢 Fichier constants.ts introuvable");
}

function updateEnv(envPath) {
  if (fs.existsSync(envPath)) {
    let envContent = fs.readFileSync(envPath, "utf8");

    if (/PGDATABASE=/.test(envContent)) {
      envContent = envContent.replace(/PGDATABASE=.*/, `PGDATABASE=${rootFolderName}`);
    } else {
      envContent += `\nPGDATABASE=${rootFolderName}`;
    }

    fs.writeFileSync(envPath, envContent);
    console.log(`🟢 ${envPath} mis à jour avec PGDATABASE=${rootFolderName}`);
  } else {
    console.log(`🟢 Aucun fichier .env trouvé à ${envPath}`);
  }
}

updateEnv(path.resolve("server/.env"));
updateEnv(path.resolve("client/.env"));
updateEnv(path.resolve(".env"));

console.log("🟢 Mise à jour terminée.");
```

- 2.15 - Racine > setup > Créer un fichier constants.ts
- 2.16 - Racine > setup > constants.ts > Coller le contenu suivant :

```
// Importez simplement `constants` où vous en avez besoin pour garantir une cohérence et
faciliter la maintenance.
// import { constants } from "../../../../setup/constants"

export const constants = {
  ROOT_FOLDER_NAME: "PENDING_GENERATION",
  APP_NAME: "PENDING_GENERATION",

    DEFAULT_USER_NAME: "utilisateur",
    DEFAULT_AVATAR: "/images/avatar_profil.png",

    ROUTE_HOME: "/",
    ROUTE_LOGIN: "/login",
    ROUTE_AUTH: "/auth",
    ROUTE_DASHBOARD: "/dashboard",
    ROUTE_CONTACT: "/contact",

    TEXT_WELCOME: "Bienvenue",
    TEXT_LOGOUT: "Se déconnecter",
    TEXT_LOGIN: "Se connecter",
    TEXT_SEARCH_PLACEHOLDER: "Rechercher...",

    DEFAULT_LANGUAGE: "fr",
    DEFAULT_THEME: "light",
    DEFAULT_TIMEOUT: 10000,
    DEFAULT_PAGE_SIZE: 10,

    API_BASE_URL: "http://localhost:3310/api",
    API_AUTH_ENDPOINT: "/auth",
};
```

- 2.17 - Racine > Créer un fichier .env.sample
- 2.18 - Racine > .env.sample > Coller le contenu suivant :

```
VITE_API_URL=/api
CLIENT_URL=http://localhost:5173
MODE=development
POSTGRES_USER=Your_user
POSTGRES_PASSWORD=Your_password
```

- 2.19 - Racine > Créer une copie du fichier .env.sample, puis renomme-le .env et renseigne les vraies infos.

3. Préparer la structure SERVER de l'application

- 3.01 - server > package.json > Ajouter et supprimer les scripts par le contenu suivant :

```
{
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "tsx src/index.ts",
    "build": "tsc",
    "start": "ts-node --transpile-only src/index.ts",
  }
}
```

```
et supprumer celui ci
"type": "commonjs",
```

- 3.02 - server > Créer un fichier tsconfig.json
- 3.03 - server > tsconfig.json > Coller le contenu suivant :

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "lib": ["ES2022"],
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "composite": true
  },
  "include": ["src", "services"]
}
```

- 3.04 - server > Créer un dossier database
- 3.05 - server > database > Créer un fichier init-db.js

```
import fs from "node:fs";
import path from "node:path";
import { fileURLToPath } from "node:url";
import dotenv from "dotenv";
import { Client } from "pg";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

dotenv.config({ path: path.resolve("server/.env") });

const DB_NAME = process.env.PGDATABASE;

const SQL_SCRIPT_PATH = path.join(__dirname, "script", "schema.sql");

const baseConfig = {
  host: process.env.PGHOST,
  port: Number(process.env.PGPORT),
  user: process.env.PGUSER,
  password: String(process.env.PGPASSWORD || ""),
};

async function initDb() {
  let tempClient;
  let finalClient;

  try {
    console.log(`Tentative de connexion à PostgreSQL pour vérifier/créer la DB
"${DB_NAME}"...`);

    tempClient = new Client({ ...baseConfig, database: "postgres" });
    await tempClient.connect();
```

```
    const res = await tempClient.query(`SELECT 1 FROM pg_database WHERE datname = $1`,
[DB_NAME]);

    if (res.rowCount === 0) {
      console.log(`Base de données "${DB_NAME}" non trouvée. Création...`);

      await tempClient.query(`CREATE DATABASE "${DB_NAME}"`);

      console.log(`🟢 Base de données "${DB_NAME}" créée avec succès.`);
    } else {
      console.log(`Base de données "${DB_NAME}" existe déjà. Initialisation des tables...`);
    }

    await tempClient.end();

    finalClient = new Client({ ...baseConfig, database: DB_NAME });
    await finalClient.connect();

    const sql = fs.readFileSync(SQL_SCRIPT_PATH, "utf8");
    await finalClient.query(sql);

    console.log("🟢 Tables et Types créés/mis à jour.");
  } catch (err) {
    if (err.code === "ENOENT") {
      console.error(`🔴 Erreur init DB: Le fichier SQL est introuvable au chemin :
${SQL_SCRIPT_PATH}`);
    } else {
      console.error("🔴 Erreur init DB:", err.message);
    }
  } finally {
    if (finalClient) await finalClient.end();
  }
}

initDb();
```

- 3.06 - server > Créer un dossier script
- 3.07 - server > database > script > Créer un fichier schema.sql
- 3.08 - server > database > script > schema.sql > Coller le contenu suivant :

```
CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(100) NOT NULL UNIQUE,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS user_profiles (
  user_id INT PRIMARY KEY,
  address VARCHAR(255),
  city VARCHAR(100),
  postal_code VARCHAR(20),
  profile_photo VARCHAR(510),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

```
DO $$
BEGIN
  IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'role_type') THEN
    CREATE TYPE role_type AS ENUM ('free', 'premium', 'admin');
  END IF;
END$$;

CREATE TABLE IF NOT EXISTS roles (
  id SERIAL PRIMARY KEY,
  role role_type NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS user_roles (
  user_id INT NOT NULL,
  role_id INT NOT NULL,
  PRIMARY KEY(user_id, role_id),
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
  FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE CASCADE
);
```

- 3.09 - server > Créer un dossier services
- 3.10 - server > services > Créer un dossier db
- 3.11 - server > services > db > Crée un fichier index.ts
- 3.12 - server > services > db > index.ts > Coller le contenu suivant :

```
import pkg from "pg";

const { Pool } = pkg;

const PGUSER = process.env.PGUSER || "postgres";
const PGPASSWORD = process.env.PGPASSWORD || "password";
const PGHOST = process.env.PGHOST || "localhost";
const PGPORT = process.env.PGPORT || "5432";
const PGDATABASE = process.env.PGDATABASE || "project name";

const connectionString = process.env.DATABASE_URL ? process.env.DATABASE_URL :
`postgresql://${PGUSER}:${PGPASSWORD}@${PGHOST}:${PGPORT}/${PGDATABASE}`;

export const db = new Pool({
  connectionString,
  ssl: process.env.NODE_ENV === "production" ? { rejectUnauthorized: false } : false,
});

const dbName = process.env.DATABASE_URL ? new
URL(process.env.DATABASE_URL).pathname.substring(1) : PGDATABASE;

console.log(`Using database ${dbName}`);
```

- 3.13 - server > services > db > Créer un fichier initDB.ts
- 3.14 - server > services > db > initDB.ts > Coller le contenu suivant :

```
import { readFileSync } from "node:fs";
import path from "node:path";
import { fileURLToPath } from "node:url";

import { db } from "./index.js";

const __filename = fileURLToPath(import.meta.url);
```

```
const __dirname = path.dirname(__filename);

export async function initDB() {
  try {
    const schemaPath = path.resolve(__dirname, "../../database/script/schema.sql");

    const schema = readFileSync(schemaPath, "utf-8");

    const client = await db.connect();

    try {
      await client.query(schema);
      console.log("Base de données initialisée avec toutes les tables");
    } finally {
      client.release();
    }
  } catch (error) {
    if (error instanceof Error && "code" in error && error.code === "42710") {
      console.log("Base de données initialisée avec toutes les tables (type ENUM déjà
existant)");
      return;
    }
    console.error("Erreur lors de l'initialisation de la base :", error);
  }
}
```

- 3.15 - server > services > Créer un fichier index.ts
- 3.16 - server > services > index.ts > Coller le contenu suivant :

```
export { db } from "./db/index.js";
export { initDB } from "./db/initDB.js";
```

- 3.17 - server > Créer le dossier src
- 3.18 - server > src > créer le dossier routes
- 3.19 - server > src > routes > api.ts > Coller le contenu suivant :

```
import { Router } from "express";

const router = Router();

router.get("/", (_req, res) => {
    res.json({ message: "API prête" });
});

export default router;
```

- 3.20 - server > src > Créer un fichier index.ts
- 3.21 - server > src > index.ts > Coller le contenu suivant :

```
import cors from "cors";
import "dotenv/config";
import type { NextFunction, Request, Response } from "express";
import express from "express";
import { db } from "../services/db/index.js";
import { initDB } from "../services/db/initDB.js";
import apiRoutes from "./routes/api.js";
```

```
const app = express();
const PORT = process.env.SERVER_PORT || 3310;
const CLIENT_URL = process.env.CLIENT_URL || "http://localhost:5173";

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use(
  cors({
    origin: CLIENT_URL,
    credentials: true,
    optionsSuccessStatus: 200,
  })
);

app.use("/api", apiRoutes);

app.use((err: Error, _req: Request, res: Response, _next: NextFunction) => {
  console.error(err);
  res.status(500).send("Erreur serveur");
});

db.query("SELECT 1")
  .then(async () => {
    const dbName = process.env.DATABASE_URL ? new
URL(process.env.DATABASE_URL).pathname.substring(1) : process.env.PGDATABASE;

    console.log(`
==========================
🟢 DATABASE
==========================
`);
    console.log(`Using database ${dbName}`);

    try {
      await initDB();

      console.log(`Connexion réussie à la base ${dbName}`);

      if (process.env.NODE_ENV === "development") {
        console.log("PGUSER:", process.env.PGUSER);
        console.log("PGDATABASE:", process.env.PGDATABASE);
        console.log("PGHOST:", process.env.PGHOST);
        console.log("PGPORT:", process.env.PGPORT);
        console.log("PGPASSWORD: 🔒");
      }

      app.listen(PORT, () => {
        console.log(`
==========================
🟢 SERVER
==========================
`);
        console.log(`Serveur lancé sur http://localhost:${PORT}`);
      });
    } catch (err: unknown) {
      console.error("🟢 Erreur init DB:", (err as Error).message);
      process.exit(1);
    }
```

```
  })
  .catch((err: unknown) => {
    console.error("Erreur de connexion PostgreSQL :", err);
    process.exit(1);
  });
```

- 3.22 - server > Créer un fichier .gitignore
- 3.23 - server > .gitignore > Coller le contenu suivant :

```
# Dépendances
node_modules/

# Environnement
.env

# Build TypeScript
dist/
*.tsbuildinfo

# Logs
logs/
*.log
npm-debug.log*

# IDE / OS
.DS_Store
.vscode/
.idea/
```

- 3.24 - server > Créer un fichier .env.sample
- 3.25 - server > .env.sample > Coller le contenu suivant :

```
NODE_ENV=development
PORT=3310
APP_SECRET=Générer une key, coller dans le terminal bash cette commande en entière : node -e
"console.log(require('crypto').randomBytes(48).toString('hex'))"

PGHOST=localhost
PGPORT=5432
PGUSER=Your_user
PGPASSWORD=Your_password
PGDATABASE=Your_db_name

CLIENT_URL=http://localhost:5173
```

- 3.26 - server > Fais une copie du fichier .env.sample, puis renomme-le .env et renseigne les vraies infos.

4. Préparer la structure CLIENT de l'application

- 4.01 - client > public > Ajouter les différents éléments accessible par le public
- 4.02 - client > public > Supprimer l'image vite.svg
- 4.03 - client > src > assets > Supprimer l'image react.svg
- 4.04 - client > src > Créer les dossiers et fichiers :

```
🗀 components
    🗀 header
```

```
        Header.css
        Header.tsx
    pages
        homePage
            HomePage.css
            HomePage.tsx
        notFoundPage
            NotFoundPage.css
            NotFoundPage.tsx
```

- 4.05 - client > src > Les différents composants ont les contenus suivant :

Header.tsx

```tsx
import "./Header.css";
import { constants } from "../../../../setup/constants";

function Header() {
    return (
        <header className="header">
            <div className="left-side">
                <h1>{constants.APP_NAME}</h1>
            </div>
        </header>
    );
}

export default Header;
```

Header.css

```css
.header {
    position: fixed;
    top: 0;
    left: 0;
    right: 0;
    z-index: 1000;
    display: flex;
    align-items: center;
    padding: 14px 24px;
    height: 72px;
    background-color: var(--secondary-color);
    color: var(--primary-color);
}

.header .left-side h1 {
    font-size: 20px;
    font-weight: bold;
    margin: 0;
    color: inherit;
}
```

HomePage.tsx

```tsx
import Header from "../../components/header/Header";
import "./HomePage.css";
import { constants } from "../../../../setup/constants";
```

```
function HomePage() {
    return (
        <div className="home-page">
            <Header />
            <h1>Bienvenue sur votre application web</h1>
            <p>Cette page d'accueil est prête à accueillir vos utilisateurs.</p>
            <h2>{constants.ROOT_FOLDER_NAME}</h2>
            <p>sera bientot disponible </p>
        </div>
    );
}

export default HomePage;
```

HomePage.css

```
body {
    padding-top: 72px;
}

.home-page {
    display: flex;
    flex-direction: column;
    align-items: center;
    text-align: center;
    padding: 120px var(--spacing) 40px;
    min-height: 100vh;
    background-color: var(--primary-color);
    color: var(--secondary-color);
}

.home-page h1 {
    font-size: 2.8rem;
    animation: fadeIn 0.8s ease-out;
}

.home-page p {
    font-size: 1.2rem;
    max-width: 600px;
    margin-bottom: 30px;
    animation: fadeInUp 1s ease-out 0.3s both;
}
```

NotFoundPage.tsx

```
import { useNavigate } from "react-router";
import Header from "../../components/header/Header";
import "./NotFoundPage.css";

function NotFoundPage() {
    const navigate = useNavigate();

    return (
        <div className="not-found-page">
            <Header />
            <h1>404 - Page non trouvée</h1>
```

```
            <button type="button" onClick={() => navigate("/")}>
                Retour à l'accueil
            </button>
        </div>
    );
}

export default NotFoundPage;
```

NotFoundPage.css

```
.not-found-page {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    text-align: center;

    min-height: 100vh;
    background-color: var(--primary-color);
    color: var(--secondary-color);
    padding: 40px;
}

.not-found-page h1 {
    font-size: 2.5rem;
    margin-bottom: 20px;
    animation: fadeIn 0.8s ease-out;
}

.not-found-page button {
    padding: 10px 20px;
    font-size: 1rem;
    font-weight: 600;
    border-radius: 8px;
    border: 2px solid var(--secondary-color);
    background-color: var(--secondary-color);
    color: var(--primary-color);
    cursor: pointer;
    transition: all 0.3s ease;
}

.not-found-page button:hover {
    background-color: var(--primary-color);
    color: var(--secondary-color);
    border-color: var(--secondary-color);
}
```

- 4.06 - client > package.json > Remplacer le contenu suivant :

```
"build": "tsc -b && vite build",
par
   "build": "npx vite build",
    "vite:build": "npx vite build",
```

- 4.07 - client > tsconfig.json > Coller le contenu suivant :

```
{
    "compilerOptions": {
        "target": "ESNext",
        "module": "ESNext",
        "moduleResolution": "node",
        "jsx": "react-jsx",
        "strict": true,
        "esModuleInterop": true,
        "skipLibCheck": true,
        "forceConsistentCasingInFileNames": true
    },
    "include": ["src"]
}
```

- 4.08 - client > src > Créer un fichier vite-env.d.ts
- 4.09 - client > src > vite-env.d.ts > Coller le contenu suivant :

```
interface ImportMetaEnv {
  readonly VITE_API_URL: string;
  readonly VITE_PROJECT_NAME: string;
}

/* biome-ignore lint/correctness/noUnusedVariables: Nécessaire pour la fusion de types
globale dans TypeScript/Vite. */
interface ImportMeta {
  readonly env: ImportMetaEnv;
}
```

- 4.10 - client > src > Créer un fichier router.tsx
- 4.11 - client > src > router.tsx > Coller le contenu suivant :

```
import { createBrowserRouter } from "react-router";
import HomePage from "../src/pages/homePage/HomePage";
import NotFoundPage from "../src/pages/notFoundPage/NotFoundPage";
import App from "./App";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    errorElement: <NotFoundPage />,
    children: [{ index: true, element: <HomePage /> }],
  },
  {
    path: "*",
    element: <NotFoundPage />,
  },
]);
export default router;
```

- 4.12 - client > src > Supprimer le fichier app.css
- 4.13 - client > src > index.css > vider le contenu
- 4.14 - client > src > index.css > Renommer ce fichier global.css
- 4.15 - client > src > global.css > Coller le contenu suivant :

```css
@import url("https://fonts.googleapis.com/css2?family=Baloo+2&display=swap");

* {
    box-sizing: border-box;
}

:root {
    --primary-color: #f1f4f9;
    --secondary-color: #008080;
    --accent-color: #ff9800;
    --text-color: #333;
    --secondary-hover-color: #006666;
    --spacing: 20px;
}

body {
    margin: 0;
    background-color: var(--secondary-color);
    color: var(--primary-color);
    font-family: "Baloo 2", cursive;
    line-height: 1.6;
}

/* Animations globales */
@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(-10px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

@keyframes fadeInUp {
    from {
        opacity: 0;
        transform: translateY(20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}
```

- 4.16 - client > src > app.tsx > Coller le contenu suivant :

```tsx
import { useEffect } from "react";
import { Outlet } from "react-router";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

import { constants } from "../../setup/constants";

function App() {
  useEffect(() => {
```

```
    const appName = constants.APP_NAME;

    document.title = appName;

    console.log("Nom du projet (Constante utilisée pour le titre) :", appName);
  }, []);

  return (
    <>
      <Outlet />
      <ToastContainer position="bottom-right" aria-label="notification" />
    </>
  );
}

export default App;
```

- 4.17 - client > src > main.tsx > Remplacer le code par le contenu suivant :

```
import React from "react";
import ReactDOM from "react-dom/client";
import { RouterProvider } from "react-router";
import "./global.css";
import router from "../src/router";

const rootElement = document.getElementById("root");

if (!rootElement) {
    throw new Error("Root element not found");
}

ReactDOM.createRoot(rootElement).render(
    <React.StrictMode>
        <RouterProvider router={router} />
    </React.StrictMode>,
);
```

- 4.18 - client > .gitignore > Remplacer le code par le contenu suivant :

```
# Dépendances
node_modules/

# Builds
dist/
.vite/

# Env local
.env


# Logs
*.log
npm-debug.log*

# IDE / OS
.DS_Store
```

```
.vscode/
.idea/
```

- 4.19 - client > Créer un fichier .env.sample
- 4.20 - client > .env.sample > Coller le contenu suivant :

```
VITE_API_URL=http://localhost:3310/api
```

- 4.21 - client > Fais une copie du fichier .env.sample, puis renomme-le .env.
- 4.22 - client > vite.config.ts > Remplacer le code par le contenu suivant :

```
import path from "node:path";
import { fileURLToPath } from "node:url";
import react from "@vitejs/plugin-react";
import type { ConfigEnv } from "vite";
import { defineConfig, loadEnv } from "vite";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

export default defineConfig(({ mode }: ConfigEnv) => {
  const env = loadEnv(mode, process.cwd(), "");
  const isDevelopment = mode === "development";

  const rootDir = path.resolve(__dirname, "..");

  const projectFolderName = path.basename(rootDir);

  console.log(`\n========================\n LOCAL CONFIG
(ESM)\n========================\n`);
  console.log(`API          : ${isDevelopment ? env.VITE_API_URL : "/api"}`);
  console.log(`DOTENV       : Variables injectées depuis .env (mode: ${mode})`);
  console.log(`PROJECT NAME CALCULÉ: ${projectFolderName}`);
  const proxyConfig = isDevelopment
    ? {
        "/api": {
          target: env.VITE_API_URL,
          changeOrigin: true,
          secure: false,
        },
      }
    : undefined;

  return {
    plugins: [react()],

    server: {
      proxy: proxyConfig,
    },

    define: {
      "import.meta.env.VITE_PROJECT_NAME": JSON.stringify(projectFolderName),
      "import.meta.env.VITE_API_URL": isDevelopment ? JSON.stringify(env.VITE_API_URL) :
JSON.stringify("/api"),
    },

    build: {
```

```
      outDir: "dist",
      emptyOutDir: true,
    },
  };
});
```

- 4.23 - client > tsconfig.node.json > Ajouter ou remplcaer le contenu suivant :

```
{
  "compilerOptions": {
    "useDefineForClassFields": true,
    "lib": ["ES2022", "DOM", "DOM.Iterable"],
    "types": ["vite/client"],

}
```

5. Finalisation et Test du Démarrage

- 5.01 - Ouvrir le terminal Ctrl+T
- 5.02 - Verifier d'être à la racine du projet
- 5.03 - Installer toutes les dépendances : **npm install**
- 5.04 - Vérifier si l'installation fonctionne : **npm run dev**

```
npm run dev

> projet@1.0.0 predev
> node server/database/init-db.js

[dotenv@17.2.3] injecting env (9) from server\.env -- tip: 📡 add observability to secrets:
https://dotenvx.com/ops
Tentative de connexion à PostgreSQL pour vérifier/créer la DB "fullstack-starter-kit"...
Base de données "fullstack-starter-kit" non trouvée. Création...
🟢 Base de données "fullstack-starter-kit" créée avec succès.
🟢 Tables et Types créés/mis à jour.

> projet@1.0.0 dev
> concurrently -c green,yellow -t "HH:mm:ss" -p "{name} {time}" "npm run dev --prefix server"
"npm run dev --prefix client"

 19:03:41
 19:03:41 > server@1.0.0 dev
 19:03:41 > tsx src/index.ts
 19:03:41
 19:03:41
 19:03:41 > client@0.0.0 dev
 19:03:41 > vite
 19:03:41
 19:03:42
 19:03:42 ========================
 19:03:42 🟢 LOCAL CONFIG (ESM)
 19:03:42 ========================
 19:03:42
 19:03:42 API         : http://localhost:3310/api
 19:03:42 DOTENV      : Variables injectées depuis .env (mode: development)
 19:03:42 PROJECT NAME CALCULÉ: fullstack-starter-kit
```

```
19:03:42
19:03:42   VITE v7.3.0  ready in 687 ms
19:03:42
19:03:42   ➜  Local:   http://localhost:5173/
19:03:42   ➜  Network: use --host to expose
19:03:42 Using database fullstack-starter-kit
19:03:42
19:03:42 ========================
19:03:42 🔘 DATABASE
19:03:42 ========================
19:03:42
19:03:42 Using database fullstack-starter-kit
19:03:42 Base de données initialisée avec toutes les tables
19:03:42 Connexion réussie à la base fullstack-starter-kit
19:03:42 PGUSER: ceddevs
19:03:42 PGDATABASE: fullstack-starter-kit
19:03:42 PGHOST: localhost
19:03:42 PGPORT: 5432
19:03:42 PGPASSWORD: 🔒
19:03:42
19:03:42 ========================
19:03:42 🔘 SERVER
19:03:42 ========================
19:03:42
19:03:42 Serveur lancé sur http://localhost:3310
```

6. Publication du Projet sur GitHub

- 6.01 - Revenir à la racine du projet à lier > **cd ..**
- 6.02 - Formater les fichiers > **npx @biomejs/biome format --write**
- 6.03 - Initialiser le dépôt Git > **git init**
- 6.04 - Vérifier les fichiers à suivre > **git status**
- 6.05 - Ajouter les fichiers au staging > **git add .**
- 6.06 - Vérifier l'intégration dans le staging > **git status**
- 6.07 - Créer un commit avec un message > **git commit -m "first project commit"**
- 6.08 - Lier Git au dépôt GitHub (clé SSH ou HTTPS) > **git remote add origin clé**
- 6.09 - Vérifier la connexion au dépôt distant > **git remote -v**
- 6.10 - Envoyer le projet sur GitHub (branche main) > **git push -u origin main**
- 6.11 - 🔘 Vérifier votre dépot sur github

7. Déploiement de l'Application sur Render.com

- 7.01 - Se rendre sur le dashboard de Render.com
- 7.02 - Ajouter un nouveau projet en cliquant sur "+ Add New"
- 7.03 - Choisir "Static Site"
- 7.04 - Sélectionner la source du code (connexion GitHub)
- 7.05 - Renseigner la commande de build : **cd client && npm install && npm run build**
- 7.06 - Indiquer le répertoire de publication "Publish Directory": **client/dist**
- 7.07 - Lancer le déploiement en cliquant sur "Deploy Static Site"
- 7.08 - 🔘 Votre site est live
- 7.09 - Visualiser l'application web en cliquant sur le lien en haut de la page