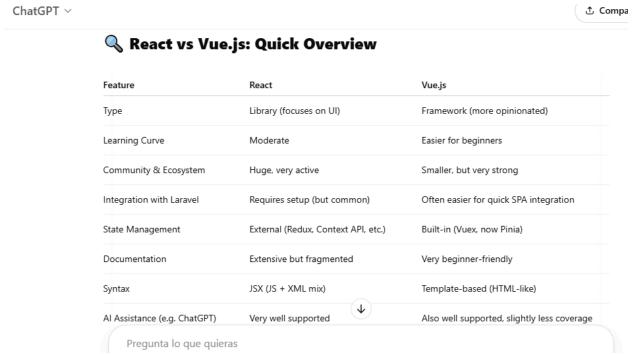
Laravel API REST Front-End Development with Generative Al

Although chatgpt is a bit behind in terms of generating images etc, and can be a bit redundant at times, I find that it makes up for it in simplifying and explaining things more basically and also being able to work with you according to your view, instructions and engagement.....

Deepseek on the other hand can be quite over generative sometimes, like chat gpt. It is also terrible with image generation and frontend designs but makes up for it in code generation. Considering the ease of use and the quality of results these two options work best for me in providing a balance and redefined understanding of codes and logic.

Prompt:

can we discuss between choosing viujs and react in the light of creating a frontend for my apirest???



Vue feels like Laravel's best friend.
React feels like the world's most popular
power tool.

Which Should You Choose?

Choose React if:

- You already have some experience with it.
- You want to learn the industry-standard frontend library.
- You're aiming for reusability and scalability in larger apps.
- You want broader support and more career opportunities.

Choose Vue.js if:

- You're looking for faster development, easier learning curve.
- You prefer a clearer separation of HTML, JS, and CSS.
- You want tighter Laravel integration without complex tooling.
- You're newer to frontend but want to build someting functional quickly.

% How to Run These

For React:

- 1. Create app: npx create-react-app my-app
- Install Axios: npm install axios
- 3. Place component in src/components, then import into App.js

For Vue:

- 1. Create app: npm create vite@latest my-vue-app -- --template vue
- 2. Install Axios: npm install axios
- 3. Use the component in App.vue



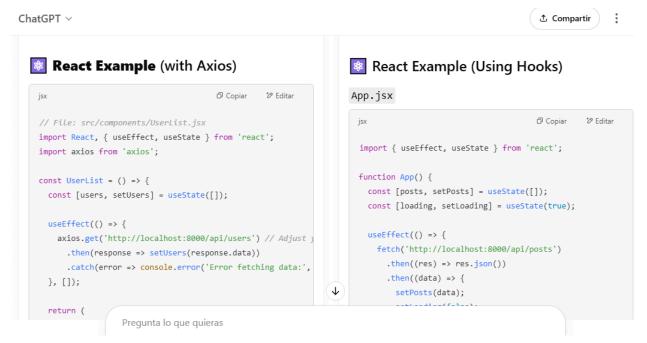
My Take (Strategic Choice):

finally choose react as it was recommended and also because its a project i would like to see grow in the future in which case react it is....

Also i would add the experience of working on something that's largely used all around the world was an eye capturing factor.

Prompt:

show sample code of react and viujs......



side by view of react code with axios and hooks. Whats the difference???

≪ What is fetch?

- fetch() is a native JavaScript function for making HTTP requests (e.g., GET, POST).
- It returns a Promise and is used to retrieve or send data to an API.
- · It has nothing to do with React specifically.

Example:

```
js

fetch('https://api.example.com/posts')
   .then(res => res.json())
   .then(data => console.log(data));
```

How They Work Together

You use fetch() inside a React hook like useEffect() to load data when the component mounts:

```
isx

useEffect(() => {
  fetch('https://api.example.com/posts')
    .then(res => res.json())
    .then(data => setPosts(data));
}, []);
```

- Here, fetch() does the data fetching.
- useEffect() makes sure the request runs once when the component loads.
- useState() stores the data in state.



| Concept | Туре | Purpose |
|-------------|----------------|-------------------------|
| fetch() | JavaScript API | Makes HTTP requests |
| useEffect() | React Hook | Runs side effects (like |

calling fetch)

useState() React Hook Stores and updates state (e.g., API data)

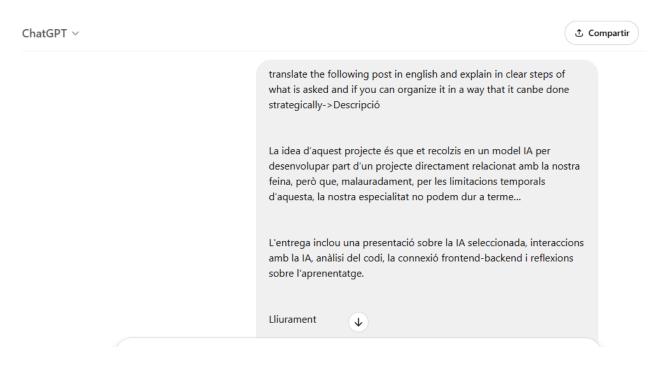
Vue.js Example (Vue 3 with Composition API)

React (Hooks)

Vue (Composition API)

| Importing | <pre>import axios from 'axios' import { useEffect, useState } from 'react'</pre> | · |
|--------------------|----------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| Data State | <pre>const [users, setUsers] = useState([])</pre> | <pre>const users = ref([])</pre> |
| Fetching Data | <pre>Inside useEffect(() => { }, [])</pre> | <pre>Inside onMounted(() => { })</pre> |
| Updating State | setUsers(response.data) | users.value = response.data |
| Rendering Data | <pre>JSX: {users.map(u => {u.name})}</pre> | <pre>Template: <li :key="u.id" v-for="u in users">{{ u.name }}</pre> |
| Reactivity | <pre>via useState() + useEffect()</pre> | via ref() — auto reactive in template |
| JSX vs Template | Uses JSX (JavaScript + HTML mix) | Uses HTML template with logic bindings |

First steps understanding the project



Strategic Plan: How to Approach This Project Step-by-Step

- Phase 1: Setup & Planning
- 1. Pick your AI tool (e.g., ChatGPT, GitHub Copilot).
- 2. Review your previous Sprint 5 API/backend project.
- 3. Choose the frontend framework (React is recommended, or try a new one if you already know React).
- 4. Design how your app will look wireframe/sketch it.

Phase 2: Al Interaction

- 5. Use the AI to help:
 - · Generate the frontend code (components, styles, structure).
 - Assist with API integration (fetch, axios, etc.).
- 6. Document key conversations:
 - . Take screenshots or notes on what you asked and what the AI gave back.

Phase 3: Development

- 7. Build your frontend using AI help.
- 8. Connect it to the backend (Laravel API).
- 9. Adjust/fix any problems document what the A t right/wrong and how you corrected it.

Pregunta lo que quieras

ChatGPT ~

₾ Com

Project Overview (Translated & Explained)

You are tasked with developing part of a web application using **Al support**, especially focusing on parts of the project that your team normally **doesn't have time or expertise to complete** on its own. The goal is **learning**, not just functionality.

Deliverables Overview

You need to create a presentation and a GitHub repository, documenting:

- 1. AI Model Used:
 - What AI did you use (e.g., ChatGPT, Copilot, Gemini)?
 - · Why did you choose it for this task?
 - What made it suitable?

2. Interaction Log with Al:

 (\mathbf{V})

Pregunta lo que quieras

creating and testing the admin register and login parts code generation with deepseek.

fo

Folder Structure



Folders divided and organized into sub folders based on responsibilities and functions.....

available at http://localhost:5173

Features Implemented

- 1. Authentication:
 - Login/logout functionality
 - Token refresh mechanism
 - Protected routes
- 2. User Management:
 - List all users
 - Create new users (admin, technician, customer)
 - Edit existing users
 - Delete users

3. Product Management:

- List all products with status indicators
- Create new products
- Edit existing products
- Delete products
- Assign products to technicians

4. Maintenance Record Management:

- List all maintenance records
- Create new maintenance records
- Edit existing records
- Delete records
- Filter by status

5. Dashboard:

- Overview statistics
- Quick access to main sections

Styling and UI

- Material-UI components for consistent design
- Responsive layout that works on desktop and mobile
- Form validation and error handling
- Loading indicators for async operations
- Confirmation dialogs for destructive actions

Security Considerations

- 1. All API calls include the authentication token
- 2. Protected routes ensure only authenticated users can access the dashboard
- 3. Admin-only routes restrict sensitive operations to admin users
- 4. Password fields are properly secured
- 5. Error messages are user-friendly but don't expose sensitive information.

.......

API Services

Let's create the API service files first:

src/api/auth.js

```
import axios from 'axios';

const API_URL = 'http://localhost:8000/api';

export const login = async (email, password) => {
  const response = await axios.post('${API_URL}/login', {
    email,
      password
    });
  return response.data;
};
```

This sections relates to the users registration, login and logout functionality. It uses **axios** which is a library that helps send data to and get data from your backend (Laravel, in this case).

Sends the user's email and password to the backend and then
Uses a **POST request** to /api/login. Likewise logout and refresh token etc
it returns the **data from the server** (usually a token) since it's with a passport.

Main App and Routing

src/main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { BrowserRouter } from 'react-router-dom';
import { AuthProvider } from './utils/auth';
import { CssBaseline, ThemeProvider, createTheme } from '@mui/material';

const theme = createTheme({
   palette: {
    primary: {
      main: '#1976d2',
    },
    secondary: {
      main: '#dc004e',
    }
```

It's the entry point — the first file React runs when starting your app. It has an important role: it's like a wrapper around the whole structure.

☐ App.jsx

This file is responsible for the setting up of routes and deciding which pages to show eg /register or /dashboard etc...

src/App.jsx

```
jsx
import { Routes, Route, Navigate } from 'react-router-dom';
import { useAuth } from './utils/auth';
import AdminLayout from './components/Layout/AdminLayout';
import Login from './pages/Login';
import Dashboard from './pages/Dashboard';
import ListUsers from './pages/Users/ListUsers';
import CreateUser from './pages/Users/CreateUser';
import EditUser from './pages/Users/EditUser';
import ListProducts from './pages/Products/ListProducts';
import CreateProduct from './pages/Products/CreateProduct';
import EditProduct from './pages/Products/EditProduct';
import ListMaintenanceRecords from './pages/MaintenanceRecords/ListMaintenanceRecords';
import CreateMaintenanceRecord from './pages/MaintenanceRecords/CreateMaintenanceRecord';
import EditMaintenanceRecord from './pages/MaintenanceRecords/EditMaintenanceRecord';
function App() {
 const { isAuthenticated } = useAuth();
  return (
    <Routes>
     <Route path="/login" element={!isAuthenticated ? <Login /> : <Navigate to="/dashboard" />}
     <Route path="/" element={isAuthenticated ? <AdminLayout /> : <Navigate to="/login" />}>
        <Route index element={<Navigate to="/dashboard" />} />
        <Route path="dashboard" element={<Dashboard />} />
```

Message DeenSeek

Authorization: 'Rearer \$/localStorage getTtem('token'));

src/api/users.js javascript import axios from 'axios'; const API_URL = 'http://localhost:8000/api'; export const getUsers = async () => { const response = await axios.get(`\${API_URL}/users`, { headers: { Authorization: `Bearer \${localStorage.getItem('token')}` } }); return response.data; }; export const createUser = async (userData) => { const response = await axios.post(`\${API_URL}/users`, userData, { headers: {

```
src/api/products.js
                                                                         javascript
     import axios from 'axios';
     const API_URL = 'http://localhost:8000/api';
     export const getProducts = async () => {
       const response = await axios.get(`${API_URL}/products`, {
        headers: {
          Authorization: `Bearer ${localStorage.getItem('token')}`
        }
       });
       return response.data;
     export const createProduct = async (productData) => {
       const response = await axios.post(`${API_URL}/products`, productData, {
Message DeepSeek
```

javascript javascript import axios from 'axios'; const API_URL = 'http://localhost:8000/api'; export const getMaintenanceRecords = async () => { const response = await axios.get('\${API_URL}/maintenance_records', { headers: { Authorization: 'Bearer \${localStorage.getItem('token')}' } }); return response.data; }; export const createMaintenanceRecord = async (recordData) => { const response = await axios.post('\${API_URL}/maintenance_records', recordData, { headers: { Message DeepSeek

This file (utils/auth.js)sets up a "control center" that:

- Knows if the user is logged in
- Stores and shares the token
- Lets any part of your app log in, log out, or check login status

src/utils/auth.js

```
<sup>©</sup> Copy <u>⊀</u> Download
   javascript
   import { createContext, useContext, useState, useEffect } from 'react';
   import { login, logout, refreshToken } from '../api/auth';
   import { useNavigate } from 'react-router-dom';
   const AuthContext = createContext();
   export const AuthProvider = ({ children }) => {
     const [user, setUser] = useState(null);
     const [token, setToken] = useState(localStorage.getItem('token'));
     const navigate = useNavigate();
     useEffect(() => {
       if (token) {
          // You might want to verify the token here or get user info
          localStorage.setItem('token', token);
        } else {
         localStorage.removeItem('token');
essage DeepSeek
```

src/main.jsx

```
Copy & Download
     import React from 'react';
      import ReactDOM from 'react-dom/client';
      import App from './App';
      import { BrowserRouter } from 'react-router-dom';
      import { AuthProvider } from './utils/auth';
      import { CssBaseline, ThemeProvider, createTheme } from '@mui/material';
     const theme = createTheme({
       palette: {
         primary: {
          main: '#1976d2',
         },
         secondary: {
          main: '#dc004e',
         },
       },
      });
      ReactDOM.createRoot(document.getElementById('root')).render(
       <React.StrictMode>
         <BrowserRouter>
           <ThemeProvider theme={theme}>
             <CssBaseline />
             <AuthProvider>
Message DeepSeek
```

Main.jsx file Starts the React app like the entry point

- Wraps the app with:
 - Routing support (BrowserRouter)
 - Authentication context (AuthProvider)
 - UI styling from MUI (ThemeProvider, CssBaseline)

```
src/App.jsx
                                                                             © Copy ± Download
 import { Routes, Route, Navigate } from 'react-router-dom';
 import { useAuth } from './utils/auth';
 import AdminLayout from './components/Layout/AdminLayout';
 import Login from './pages/Login';
 import Dashboard from './pages/Dashboard';
 import ListUsers from './pages/Users/ListUsers';
 import CreateUser from './pages/Users/CreateUser';
 import EditUser from './pages/Users/EditUser';
 import ListProducts from './pages/Products/ListProducts';
 import CreateProduct from './pages/Products/CreateProduct';
 import EditProduct from './pages/Products/EditProduct';
 import ListMaintenanceRecords from './pages/MaintenanceRecords/ListMaintenanceRecords';
 import CreateMaintenanceRecord from './pages/MaintenanceRecords/CreateMaintenanceRecord';
 import EditMaintenanceRecord from './pages/MaintenanceRecords/EditMaintenanceRecord';
 function App() {
   const { isAuthenticated } = useAuth();
   return (
       <Route path="/login" element={!isAuthenticated ? <Login /> : <Navigate to="/dashboard" />}
       <Route path="/" element={isAuthenticated ? <AdminLayout /> : <Navigate to="/login" />}
         <Route index element={<Navigate to="/dashboard" />} />
```

This file covers the main routing logic of the app that directs or redirects to the required route.

Experience:

It was interesting to review and study the way react frontends is set up and communicate with the api moreover by trying it out myself being that i have no anterior knowledge of react. During test the generated ai code i realized i faced a few error challenges and it was interesting trying to understand what the error message was pointing to a times it is very clear a times it just blur mambo jumbo but ai could clarify where the error was pointing at and whats to be done to fix it and that's interesting.

Products/EditProduct.jsx is missing and then lets see if we can test it out to see if it works Here's the complete EditProduct.jsx component, and then we'll discuss testing the application: src/pages/Products/EditProduct.jsx © Copy ± Download jsx import { useState, useEffect } from 'react'; import { useParams, useNavigate } from 'react-router-dom'; import { Box, Button. Typography, Paper, Container. CircularProgress } from '@mui/material'; import ProductForm from '../../components/Products/ProductForm'; import { getProduct, updateProduct } from '../../api/products'; import { getUsers } from '../../api/users'; const EditProduct = () => { const { id } = useParams(); const navigate = useNavigate(); const [formData cotEnrmData] - weastate()

\$ npm run dev

- > admin-frontend@0.0.0 dev
- > vite

VITE v6.3.3 ready in 2678 ms

- → Local: http://localhost:5173/
- → Network: use --host to expose
- → press h + enter to show help

Error: Failed to scan for dependencies from entries:

C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-frontend/index.html

X [ERROR] The JSX syntax extension is not currently enabled

src/utils/auth.js:64:9:

64 | return <AuthContext.Provider value={value}>{children}</AuthConte...

The esbuild loader for this file is currently set to "js" but it must be set to "jsx" to be able to parse JSX syntax. You can use "loader: { '.js': 'jsx' }" to do that.

at failureErrorWithLog (C:\xampp\htdocs\MyProject\Sprints\backup\Front_End_Version\adminfrontend\node_modules\esbuild\lib\main.js:1477:15)

VITE v6.3.3 ready in 545 ms → Local: http://localhost:5174/ → Network: use --host to expose → press h + enter to show help 22:59:53 [vite] (client) Pre-transform error: Failed to parse source for import analysis because the content contains invalid JS syntax. If you are using JSX, make sure to name the file with the .jsx or .tsx extension. 22:59:53 [vite] Internal server error: Failed to parse source for import analysis because the content contains invalid JS syntax. If you are using JSX, make sure to name the file with the .jsx or .tsx extension. Plugin: vite:import-analysis File: C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/adminfrontend/src/utils/auth.js:68:28 66 | 67 | export const useAuth = () => { 68 | return useContext(AuthContext); 69 | }; at TransformPluginContext_formatLog (file:///C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/adminfrontend/node_modules/vite/dist/node/chunks/dep-BMIURPaQ.js:42451:41) at TransformPluginContext.error (file:///C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-

frontend/node_modules/vite/dist/node/chunks/dep-BMIURPaQ.js:42448:16)

at TransformPluginContext.transform

vlessage DeepSeek

```
31 | } from "@mui/icons-material";
        32 | import { getUsers, getProducts, getMaintenanceRecords } from "../api";
        33 | const Dashboard = () => {
        34 | _s();
          at TransformPluginContext,_formatLog
       (file:///C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-
       frontend/node_modules/vite/dist/node/chunks/dep-BMIURPaQ.js:42451:41)
          at TransformPluginContext.error
       (file:///C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-
       frontend/node_modules/vite/dist/node/chunks/dep-BMIURPaQ.js:42448:16)
          at normalizeUrl (file:///C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-
      frontend/node_modules/vite/dist/node/chunks/dep-BMIURPaQ.js:40427:23)
          at process.processTicksAndRejections (node:internal/process/task_queues:105:5)
          at async file:///C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-
       frontend/node_modules/vite/dist/node/chunks/dep-BMIURPaQ.js:40546:37
          at async Promise all (index 6)
          at async TransformPluginContext.transform
       (file:///C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-
       frontend/node_modules/v23:01:41 [vite] Internal server error: Failed to resolve import ".,/.,/api" from
       "src/pages/MaintenanceRecords/CreateMaintenanceRecord.jsx". Does the file exist?
        Plugin: vite:import-analysis
        File: C:/xampp/htdocs/MyProject/Sprints/backup/Front_End_Version/admin-
       frontend/src/pages/MaintenanceRecords/CreateMaintenanceRecord.jsx:18:38
        33 | import { DatePicker } from "@mui/x-date-pickers";
        34 | import { createMaintenanceRecord } from "../../api/maintenanceRecords";
        35 | import { getProducts, getUsers } from "../../api";
Message DeepSeek
```

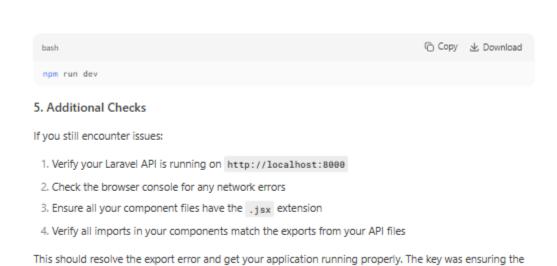
The errors where pointing at distinct issues in my frontend

Some of which where related to import issues other referring to syntax issues etc

Here's a step-by-step solution:

- fix JSX file extensions and configuration
- Fix API Import Errors
- Fix Dashboard Imports
- Update Auth Context

- Verify API File Structure
- Fix Maintenance Records Imports
- Clean and Reinstall Dependencies
- Start the Development Server
- npm run dev



API functions being imported actually exist in the exports from the API files.

With DeepSeek: I intended creating seeders with fake data to make my API more presentable for an admin frontend to be able to use.

wouldnt this new seeder affect my other seeder classes and my tdd tests??

< 2/2 >



You're absolutely right to be concerned about how the new FakeDataSeeder might affect your existing seeders and tests. Let me address this and provide a solution that maintains test integrity while giving you the fake data you need for development.

Potential Impacts and Solutions

1. Impact on Other Seeders

Your existing seeders (PassportSeeder , RolesAndPermissionsSeeder) should run first since they set up fundamental requirements. The FakeDataSeeder depends on these, so the order in DatabaseSeeder is correct.

Here: I had issues with its code generation because I was using specific tokens to test my TDD and hence it seemed conflicting data. One info going ahead of the other seems very conflicting and hence it agreed and seems to provide a different route to solve the issue...

Why This Happened:

- 1. The error occurred because:
 - Your code imports DatePicker from @mui/x-date-pickers
 - o This package wasn't listed in your package.json
 - Vite couldn't resolve the import during the build process
- 2. The @mui/x-date-pickers package is a separate library from the core Material-UI that provides advanced date/time picking components.

Additional Recommendations:

1. Check all dependencies in your package. json to ensure you have all required packages:



```
php
// database/seeders/FakeDataSeeder.php
namespace Database\Seeders;
use App\Models\User;
use App\Models\Product;
use App\Models\MaintenanceRecord;
use Illuminate\Database\Seeder;
class FakeDataSeeder extends Seeder
   public function run()
       // Create admin user
       $admin = User::factory()->create([
           'name' => 'Admin User',
           'email' => 'admin@example.com',
           'password' => bcrypt('password'),
           'role' => 'admin',
```

Frontend-Backend Connection Overview

The frontend connects to the Laravel backend through my **RESTful API endpoints which is** hosted at http://localhost:8000/api.

The frontend uses **Axios** for HTTP requests and handles **authentication** using a passport generated tokens stored in localStorage.

Key integration points:

- 1. All requests use **Bearer token authentication**.
- 2. There's an **Axios instance** (http.js) configured with interceptors for attaching tokens and handling unauthorized errors.
- 3. The frontend logic is cleanly abstracted in api/*.js files for each resource (auth, products, users, etc.), promoting modularity.

Api admin front end Flow:

- A user logs in via the login() function (POST to /api/login) from auth.js, which inturn returns a token that entails the user's info.
- The token is then saved to the file localStorage and used in all future requests via the Authorization: Bearer <token> header.
- The http.js file helps to globally intercept all requests made to include this token automatically.
- CRUD operations (e.g., for products or users) are handled in respective at api/*.js files.
- They all naturally hit Laravel's corresponding API routes, passing token headers and JSON data.