Frontend Recruitment Technical Test

- **Test:** Senior Frontend Technical Test (Vue 3)
- **Estimated duration:** 3–4 hours
- **Delivery:** Link to a public repository (GitHub, GitLab, or ZIP format)

💡Context

At Vivara.io, the frontend must be **pragmatic**, **scalable**, **maintainable**, and provide a **great user experience**.

In this test, we want to see **how you design and structure a small application**, rather than expecting everything to be perfectly polished or fully completed.

🗒️ Task

Build a **Vue 3** application that consumes a **public REST API**.

- We will use https://dummyjson.com/

The application must:

- Display a **list** of the items.
- Implement basic **CRUD** operations (create, edit, delete).
- Include a minimum level of **thoughtful design and user experience**.

We will use the public **DummyJSON** API as a simulated backend.

- **Base URL:** https://dummyjson.com
- **Main resource:** products https://dummyjson.com/docs/products

Minimum endpoints to use:

- `GET /products?limit=10&skip=0` – paginated product list
- `GET /products/:id` – product detail
- `GET /products/search?q=...` – full text search
- `GET /products/categories` and `GET /products/category/:category?limit=10&skip=0` – category filters
- `POST /products/add`, `PUT /products/:id`, `DELETE /products/:id` – simulated CRUD operations (writes are not persisted on the server, but the frontend should handle them as if they were real)

💻 Functional requirements

1. **Main list**
    - A table-based list view showing the main product fields.
    - Regarding tables, you only need to handle pagination (no extra features like sorting, configurable columns, grouping, etc. are required).
2. **Filters and search**
    - Full-text search (e.g., by name, title, etc.).
    - Category filter (dropdown or tags).
3. **Detail**
    - A detail view for an item:
        - It can be a separate route, a modal, or a side panel.
    - Display extended information about the item.
4. **CRUD**
    - **Create** a new product via a form.
    - **Edit** an existing product (using the same form for create).
    - **Delete** a product with confirmation.

    - Since DummyJSON does not persist writes, you are expected to handle create, update, and delete operations on the frontend as if they were real (e.g., updating local state accordingly).

🛠️ Technical requirements

- **Framework:** VUE 3 with **Composition API (TypeScript)**.
- **Build tool:** Vite.
- **Router:** vue-router.
- **Mockups / Wireframes / Low Fidelity**
  - You have some wireframes available as a guide. Based on them, you have full creative freedom.
- **Styling / UI:**
  - You may use CSS, SCSS, CSS Modules, or Tailwind.
  - We expect a responsive design.
- **Component framework:**
  - You may use any component framework.
- **Icons:**
  - You may use any icon library (e.g., https://heroicons.com).
- **Testing:**
  - Unit test: Vitest/Jest + Vue Test Utils
  - E2E (e.g, https://playwright.dev)
- **API:**
  - https://dummyjson.com
- **Tooling:**
  - Working scripts for dev, build, and test.
  - ESLint / Prettier configuration is a plus.

🔍 What we will take in account

- Code organization and clarity.

- State and data flow management.
- User experience.
- Navigation and user flows between list and detail views.
- Separation of concerns and architectural decisions.
- Testing approach and code quality.
- Ability to explain technical choices and trade-offs.

📦 Delivery

- Link to a public repository (GitHub, GitLab, or ZIP format).
- Include a README with:
    - Instructions to run the project.
    - A brief explanation of the project structure.
    - Key technical decisions and trade-offs.
    - Any limitations or pending improvements.