

My AYP Portal 1.0 – Frontend

Documentation written by **Muhammad Ilham Rafiannandha** for the **Take Home Technical Test – AYP Group Recruitment**.

This repository hosts the Next.js (App Router) frontend that accompanies a Laravel API. The UI focuses on displaying and editing employee records with a clean, responsive control panel suitable for managing ~1,000 entries.

1. System Requirements & Setup

Requirement	Version / Notes
Node.js	>= 18.18.x (Next.js 16 requirement)
npm	>= 9.x
OS	Linux, macOS, or WSL2 on Windows

Installation

1. Clone the repository (include the `.git` folder as requested in the brief).
2. Install dependencies:

```
npm install
```

3. Configure environment variables by copying `.env.local` and editing when needed:

```
cp .env.local .env
# NEXT_PUBLIC_API_URL defaults to http://localhost:8000/api
```

4. Ensure the Laravel backend runs at the URL defined above with `/auth/login`, `/employees`, and `/employees/{id}` endpoints.

Useful npm scripts

Script	Description
<code>npm run dev</code>	Start Next.js dev server on <code>http://localhost:3000</code> .
<code>npm run build</code>	Production build.
<code>npm run start</code>	Run the compiled build.
<code>npm run lint</code>	Run ESLint (Next.js preset).

2. Feature Overview

1. Authentication screen ([/login](#)):

- Email/password form tied to `POST /auth/login`.
- Stores JWT + user profile to `localStorage` via `lib/auth.ts`.
- Redirects authenticated users to `/employees`.

2. Employee dashboard ([/employees](#)):

- Fetches `GET /employees` through `lib/api.ts`.
- Scrollable table supports up to 1,000 records (virtualization is not yet required by spec).
- “Update” button appears only for active employees.
- Logout button clears auth state.

3. Update modal (`components/EmployeeEditModal.tsx`):

- Pre-fills selected employee.
- Name/email text inputs + `ToggleSwitch` for `isActive`.
- Save triggers `PATCH /employees/:id` and optimistically updates the table.

4. Design system (see `app/globals.css`):

- Theming, table, modal, and responsive shell styles.
- Focus on keyboard-friendly inputs and modest mobile tweaks.

Extra polish

- Sticky table header for better navigation across long lists.
- Compact action column messaging when updates are disabled (inactive employees).
- Token-aware fetch wrapper with consistent error handling.

3. Architecture & Code Map

```

app/
  layout.tsx      -> Shell wrapper (navbar + container)
  page.tsx        -> Redirects root to /login
  login/page.tsx -> Auth form and onboarding copy
  employees/page.tsx -> Protected employee dashboard
components/
  EmployeeEditModal -> CRUD modal, uses Modal + ToggleSwitch
  Modal            -> Generic backdrop/panel
  ToggleSwitch     -> Reusable UI control
lib/
  api.ts          -> Fetch helper with auth headers + error parsing
  auth.ts         -> Token/user persistence helpers
types/
  index.ts        -> Shared interfaces

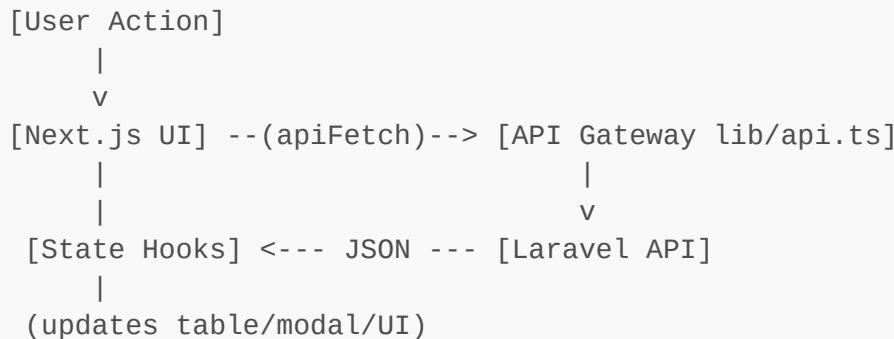
```

Key architectural choices:

- **App Router** for file-based routing and layout composition.
- **Client components** on stateful pages/components to leverage hooks.
- **Single source of truth** for employee data on `/employees`, updated via callbacks from the modal.

- **API layer** centralizes headers, base URL, and error handling, keeping components declarative.

4. Data Flow & Sequence



Sequence example for editing an employee:

1. User clicks **Update** → modal opens with selected employee state.
2. On submit, form calls `apiFetch('PATCH /employees/:id')`.
3. API success returns updated employee → `onSaved` merges it into `employees` state.
4. Modal closes and the table row reflects the new values.

5. API Contract (expected backend behavior)

Endpoint	Description	Payload / Response
<code>POST /auth/login</code>	Returns JWT + user profile.	<code>{ email, password } → { token, user }</code>
<code>GET /employees</code>	List of employees.	<code>[{ id, name, email, isActive }]</code>
<code>PATCH /employees/:id</code>	Update record.	<code>{ name, email, isActive } → updated employee</code>

The frontend expects JSON responses with either `{ data: ... }` or raw objects (handled in `api.ts`).

6. Testing, Performance & Future Work

- **Manual testing:** flows covered during development – login success/fail, unauthorized redirect, modal validation, update success/failure paths.
- **Performance considerations:**
 - Table is simple but scrollable; virtualization could be introduced if the dataset regularly exceeds 1,000 rows.
 - API calls are cached in component state; SWR/React Query can be plugged in later for stale-while-revalidate patterns.
- **Next steps:**
 - Add unit tests for `EmployeeEditModal` and integration tests for `/employees`.

- Implement skeleton loaders.
 - Expand responsive layout for narrower screens (currently optimized for >=768px with basic tweaks).
-

7. AI Tool Disclosure

I used generative AI assistance for:

1. Drafting this README structure and wording.
2. Crafting portions of `app/globals.css` to accelerate theming decisions.
3. Speeding up boilerplate setup for client components such as `EmployeeEditModal.tsx` and `ToggleSwitch.tsx`.

All logic and final decisions were reviewed, adapted, and verified manually.

8. Video & Submission Notes

- Record a short walkthrough video covering login, listing, and update flows once both frontend and Laravel backend are running.
 - Include the `.git` directory in the final archive so reviewers can inspect commit history.
 - Keep this repository private per the instructions.
-

9. Author

Muhammad Ilham Rafiannandha

Take Home Technical Test – AYP Group Recruitment

Frontend stack: Next.js 16 + TypeScript. Backend counterpart: Laravel (develop separately).