v on GitHub

JS Frameworks and Server | Next.js Labs

Lab 4 – Next.js Portfolio: Auth, Data & Dashboard

Auth0 login/logout, Neon-backed CRUD, and a protected dashboard.

🎯 Lab 4 Objectives

1. Auth0: configure tenant, wire SDK, and expose helpers to read the session in server components and API routes.
2. Protected routes: gate `/dashboard` (and future admin areas) so unauthenticated visitors are redirected to Auth0 login.
3. Database: connect to Neon Postgres, seed projects, and build reusable CRUD helpers.
4. API: ship authenticated CRUD endpoints under `/api/projects`.
5. UI wiring: show/hide Add/Edit/Delete controls based on session and fetch projects through API routes.
6. Editing: add `/projects/[uuid]/edit` with a prefilled form that updates or deletes DB rows.

📚 What You'll Learn

- Auth0 in the Next.js App Router (login/logout endpoints + server-side session reads).
- Protecting server components with `redirect`/`notFound` based on auth state.
- Using Neon's serverless Postgres client in edge/serverless runtimes.
- Structuring CRUD endpoints with zod validation and consistent JSON responses.
- Client-side auth state with `useUser`, SWR/useEffect fetching, and router refreshes after writes.
- Toast-driven UX for edits/deletes.

✅ End Result (By the End of Lab 4)

- `/api/auth/login` and `/api/auth/logout` work end-to-end with Auth0; `useUser()` shows profile data in client components.
- `/dashboard` redirects unauthenticated visitors; signed-in users see their email.
- Neon-backed project data available via helpers and API routes (GET/POST/PUT/DELETE).
- Project cards/detail pages only show edit/delete controls to logged-in users.
- `/projects/[uuid]/edit` updates DB rows; delete removes rows and refreshes list/detail views.

🧰 Prerequisites

- Finished Lab 3 working copy (same repo/branch).
- Auth0 tenant + application (free tier is fine).
- Neon Postgres project (connection string with `?sslmode=require`).
- `.env.local` ready for new secrets (`AUTH0_*`, `NEON_DB_URL`, etc.).

📁 Suggested Folder Additions

```
src/
├── app/
│   ├── api/
│   │   ├── auth/
│   │   │   └── [auth0]/route.js       # Auth0 handler
│   │   └── projects/
│   │       ├── [uuid]/route.js       # GET/PUT/DELETE for a single project
│   │       ├── new/route.js          # POST /api/projects/new (create)
│   │       └── route.js              # GET /api/projects (list)
│   ├── dashboard/
│   │   └── page.jsx                  # protected dashboard
│   └── projects/
│       └── [uuid]/edit/page.jsx      # edit form page
├── components/
│   ├── edit-project-form.jsx         # client form for editing
│   └── project-card.jsx              # shows edit/delete when session exists
└── lib/
    ├── auth0.js                      # Auth0 helpers
    └── db.js                         # Neon client + CRUD helpers
```

🧭 Step-by-Step Instructions

1) Auth0 Integration & Session Helpers

- Create an Auth0 Application (Regular Web App). Callback: `http://localhost:3000/api/auth/callback`. Logout URL: `http://localhost:3000/`.
- Install SDK:

```
npm install @auth0/nextjs-auth0
```

- Add to `.env.local` (replace placeholders):

```
AUTH0_SECRET=generate_a_random_32_char_string
AUTH0_BASE_URL=http://localhost:3000
AUTH0_ISSUER_BASE_URL=https://<your-tenant>.us.auth0.com
AUTH0_CLIENT_ID=...
AUTH0_CLIENT_SECRET=...
```

- Create `src/lib/auth0.js`:

```
import { handleAuth, getSession } from "@auth0/nextjs-auth0";

export const auth0 = {
  handle: handleAuth(),
  async getSession() {
    return getSession();
  },
  async requireSession() {
    const session = await getSession();
    if (!session?.user) throw new Error("Unauthorized");
    return session;
  },
};
```

- Route handler: `app/api/auth/[auth0]/route.js`

```
rt { auth0 } from "@/lib/auth0";
rt const { GET, POST } = auth0.handle;
```

- Exercises:
  - Hit `http://localhost:3000/api/auth/login` and inspect `useUser()` output in a client component.
  - Protect a server component by throwing `notFound()` or `redirect("/api/auth/login")` when no session.

---

2) Protected `/dashboard`

- Create `src/app/dashboard/page.jsx`:

```
import { auth0 } from "@/lib/auth0";
import { redirect } from "next/navigation";

export default async function DashboardPage() {
  const session = await auth0.getSession();
  if (!session?.user) redirect("/api/auth/login");

  return (
    <section className="min-h-screen flex flex-col items-center gap-3">
      <h1 className="text-3xl font-semibold">Dashboard</h1>
      <p className="text-muted-foreground">Welcome {session.user.email}</p>
      <p className="text-sm text-muted-foreground">
        Future labs will let you edit the hero from here; for now, render placeholders.
      </p>
    </section>
  );
}
```

---

3) Neon Postgres Setup

- In Neon console: create Project → Branch → Database. Copy the connection string (include `?sslmode=require`) into `.env.local`:

```
NEON_DB_URL=postgresql://<user>:<password>@<host>/<db>?sslmode=require
```

- Install Neon client:

```
npm install @neondatabase/serverless
```

- DB helper `src/lib/db.js` (outline):

Download db.js reference from here

```
"use server";
import { neon } from "@neondatabase/serverless";

const sql = neon(process.env.NEON_DB_URL);

function mapProject(row) {
  return {
    id: row.id,
    title: row.title,
    description: row.description,
    image: row.image,
    link: row.link,
    keywords: row.keywords ?? [],
    createdAt: row.created_at,
    updatedAt: row.updated_at,
  };
}

export async function ensureProjectsTable() {
  await sql`
    CREATE TABLE IF NOT EXISTS projects (
      id uuid PRIMARY KEY,
      title text NOT NULL,
      description text NOT NULL,
      img text NOT NULL,
      link text NOT NULL,
      keywords jsonb NOT NULL DEFAULT '[]'::jsonb,
      created_at timestamptz NOT NULL DEFAULT now(),
      updated_at timestamptz NOT NULL DEFAULT now()
    )
  `;
}

export async function seedProjectsTable(seed) {
  for (const item of seed) {
    await sql`
      insert into projects (title, description, image, link, keywords)
      values (${item.title}, ${item.description}, ${item.image}, ${item.link}, ${item.keywords})
      on conflict do nothing;
    `;
  }
}

export async function fetchProjects() {
  const rows = await sql`select * from projects order by created_at desc`;
  return rows.map(mapProject);
}

export async function getProjectById(id) {
  const [row] = await sql`select * from projects where id = ${id} limit 1`;
  return row ? mapProject(row) : null;
}

export async function insertProject(data) {
  const [row] = await sql`
    insert into projects (title, description, image, link, keywords)
    values (${data.title}, ${data.description}, ${data.image}, ${data.link}, ${data.keywords})
    returning *;
  `;
  return mapProject(row);
}

export async function updateProject(id, updates) {
  const [row] = await sql`
    update projects
    set title = coalesce(${updates.title}, title),
        description = coalesce(${updates.description}, description),
        image = coalesce(${updates.image}, image),
        link = coalesce(${updates.link}, link),
        keywords = coalesce(${updates.keywords}, keywords),
        updated_at = now()
    where id = ${id}
    returning *;
  `;
  return row ? mapProject(row) : null;
}

export async function deleteProject(id) {
  onst [row] = await sql`delete from projects where id = ${id} returning *;`;
  eturn row ? mapProject(row) : null;
}
```

- Checkpoint: call `await fetchProjects()` in a server route and log results to confirm seed data works.

---

4) Backend CRUD API

- `app/api/projects/route.js` → `GET` uses `fetchProjects()`.
- `app/api/projects/new/route.js` → wrap handler with `auth0.requireSession()`; parse JSON or `FormData` with zod, call `insertProject`, return 201.
- `app/api/projects/[uuid]/route.js`
  - `GET` → `getProjectById`.
  - `PUT` → validate payload, `updateProject(uuid, payload)`.
  - `DELETE` → `deleteProject(uuid)`.
- Respond with `NextResponse.json({ message, data })`; return 404 when null.

Example PUT handler:

```
import { NextResponse } from "next/server";
import { z } from "zod";
import { auth0 } from "@/lib/auth0";
import { updateProject } from "@/lib/db";

const projectSchema = z.object({
  title: z.string().min(1).optional(),
  description: z.string().min(1).optional(),
  image: z.string().url().optional(),
  link: z.string().url().optional(),
  keywords: z.array(z.string()).optional(),
});

export async function PUT(request, { params }) {
  await auth0.requireSession();
  const body = projectSchema.parse(await request.json());
  const updated = await updateProject(params.uuid, body);
  if (!updated) {
    return NextResponse.json({ message: "Not found" }, { status: 404 });
  }
  return NextResponse.json({ message: "Project updated", data: updated });
}
```

---

5) Connect UI ↔ API & Auth States

- Projects page `app/projects/page.jsx` → mark `"use client"`. Fetch `/api/projects` with `useEffect/useState` or SWR. Use `useUser()` to pass session into cards.
- `components/project-card.jsx` → render `EditProjectButton` and `DeleteProjectButton` only when a session exists.
- Project detail page → server component that fetches with `getProjectById` and passes session; show edit/delete controls for logged-in users.
- Delete button (`"use client"`) → confirm, `fetch("/api/projects/${id}", { method: "DELETE" })`, toast on success/error, then `router.refresh()`.

---

6) Editing UI

- Edit page `app/projects/[uuid]/edit/page.jsx` → server component: load project via `getProjectById`; if missing, `notFound()`. Pass data to `<EditProjectForm project={project} uuid={params.uuid} />`.
- `components/edit-project-form.jsx` (`"use client"`) → prefill React Hook Form defaults. Submit via:

```
await fetch(`/api/projects/${uuid}`, {
  method: "PUT",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(values),
});
```

- On success: `toast.success(...)`, `router.push(`/projects/${uuid}`)`. Consider a delete button that calls DELETE and redirects to `/projects`.

🧪 Test Checklist

- Auth0:
  - `/api/auth/login` and `/api/auth/logout` redirect correctly.
  - `useUser()` returns profile data when signed in.
- Dashboard:
  - Unauthenticated → redirect to Auth0 login.
  - Authenticated → shows email + placeholder content.
- Database:
  - `fetchProjects()` returns seeded rows; CRUD helpers map keywords + timestamps.
- API:
  - GET `/api/projects` returns an array.
  - POST `/api/projects/new` creates with auth requirement.
  - GET/PUT/DELETE `/api/projects/:uuid` work and return 404 for missing rows.
- UI:
  - Project list/detail render data from API.
  - Edit/Delete buttons only visible when logged in.
  - `/projects/[uuid]/edit` saves updates; delete removes rows and refreshes UI.

🧯 Troubleshooting & Notes

- Auth errors: confirm callback/logout URLs and `AUTH0_*` match your tenant.
- 401s: ensure `auth0.requireSession()` wraps write handlers; log `await auth0.getSession()` in the request context to debug.
- SSL/connection issues: Neon URLs must include `?sslmode=require`.
- Zod errors: surface validation messages in responses for faster debugging.
- Router refresh: call `router.refresh()` after deletes to invalidate server component fetches.

📌 Next Steps (Optional Enhancements)

- Add role-based access (e.g., allow only certain emails to edit/delete).
- Centralize toasts for API mutations.
- Add pagination or filtering powered by SQL (ORDER BY/LIMIT/OFFSET).
- Persist audit logs (who edited what/when) in a new table.

---

nal checklist before submission

- ☐ Auth0 integration works end-to-end (login/logout/useUser).
- ☐ `/dashboard` redirects unauthenticated users.
- ☐ Neon DB reachable; seed data present.
- ☐ CRUD API responds with correct statuses and JSON.
- ☐ UI shows Add/Edit/Delete only when authenticated.
- ☐ `/projects/[uuid]/edit` updates the DB; delete removes the row and refreshes.
- ☐ Meaningful error handling + toasts for mutations.

Submission

- Provide screenshots of:
  - `/dashboard` while logged in (showing email).
  - Project list/detail with edit/delete controls visible when logged in.
  - `/projects/[uuid]/edit` before/after saving; delete path confirmation.
- Commit and push all Lab 4 changes to your GitHub repo and submit the repository link.