

Manual de usuario para aplicación del Sistema de monitoreo de parámetros fisicoquímicos de calidad del agua para la conservación ex-situ de especies de género *ambystoma*.

Trabajo Terminal No. 2024-B105

*Alumnos: Cazares Cruz, Jeremy Sajid, Bucio Barrera Oscar Daniel, *Guerrero Pérez Brandon Josué*

Directores: Morales Rodríguez Úrsula Samantha, Rodríguez Jordán Gabriel de Jesús

Email: bguerrerop1600@alumno.ipn.mx


Resumen – AjoloApp es una herramienta integral diseñada para monitorear y gestionar parámetros fisicoquímicos del agua, contribuyendo a la conservación ex-situ de las especies del género *Ambystoma*. La aplicación ofrece un panel de control con vistas generales de instalaciones, tanques activos, ajolotes registrados y alertas críticas, permitiendo también la generación de informes en PDF. Entre sus principales funcionalidades, destaca la gestión de ajolotarios, tanques, sensores, usuarios y alertas, cada una con opciones para agregar, editar, visualizar y filtrar información. La aplicación soporta el registro de nuevos usuarios, el cambio de roles y la visualización detallada de mediciones recolectadas por sensores. Además, integra un sistema de generación de informes y una interfaz para la administración de dispositivos y parámetros asociados al monitoreo. Su diseño intuitivo y capacidad de personalización facilitan la toma de decisiones para asegurar un ambiente óptimo para la conservación de los ajolotes.

1. Introducción

AjoloApp es una aplicación web diseñada para el monitoreo y la gestión de parámetros fisicoquímicos del agua, enfocada en la conservación ex-situ de las especies del género *Ambystoma*. La aplicación proporciona una solución integral que combina tecnología de vanguardia con una interfaz intuitiva. Este documento tiene como objetivo describir detalladamente la arquitectura, configuración y componentes técnicos de AjoloApp, facilitando el mantenimiento, escalabilidad y extensión de la aplicación.

2. Tecnologías utilizadas

AjoloApp está construida utilizando una pila tecnológica moderna y robusta. En el frontend, se utiliza React junto con Next.js para gestionar el enrutamiento y el renderizado del lado del cliente y del servidor. Los componentes de interfaz están estilizados con Tailwind CSS, lo que permite una personalización rápida y eficiente del diseño. En el backend, se implementa Next.js con rutas API que interactúan con una base de datos PostgreSQL a través de Prisma ORM. La autenticación está gestionada mediante NextAuth, utilizando credenciales seguras. Además, se integran servicios externos como Telegram y WhatsApp para notificaciones en tiempo real, y jsPDF junto con html2canvas para la generación de informes PDF.



The image shows a code editor window with the title 'package.json'. The editor displays the content of a 'package.json' file for a project named 'ajolotarios'. The file includes metadata like name, version, and private status, as well as scripts for development, building, starting, linting, testing, and seeding. It also lists a comprehensive set of dependencies, including Prisma, Radix UI components, Jest, and various utility libraries.

```

1  {
2    "name": "ajolotarios",
3    "version": "0.1.0",
4    "private": true,
5    "scripts": {
6      "dev": "next dev",
7      "build": "prisma generate && prisma migrate deploy && next build",
8      "start": "next start",
9      "lint": "next lint",
10     "test": "jest",
11     "seed": "ts-node prisma/seed.ts"
12   },
13   "dependencies": {
14     "@prisma/client": "^5.6.0",
15     "@radix-ui/react-avatar": "^1.1.0",
16     "@radix-ui/react-dialog": "^1.1.2",
17     "@radix-ui/react-dropdown-menu": "^2.1.2",
18     "@radix-ui/react-icons": "^1.3.0",
19     "@radix-ui/react-label": "^2.1.0",
20     "@radix-ui/react-popover": "^1.1.2",
21     "@radix-ui/react-progress": "^1.1.0",
22     "@radix-ui/react-select": "^2.1.2",
23     "@radix-ui/react-slot": "^1.1.0",
24     "@radix-ui/react-switch": "^1.1.1",
25     "@testing-library/jest-dom": "^6.4.5",
26     "@tsparticles/engine": "^3.7.1",
27     "@vis.gl/react-google-maps": "^0.4.2",
28     "axios": "^1.6.5",
29     "bcrypt": "^5.1.1",
30     "chart.js": "^4.4.7",
31     "class-variance-authority": "^0.7.0",
32     "clsx": "^2.1.1",
33     "date-fns": "^3.6.0",
34     "html2canvas": "^1.4.1",
35     "jspdf": "^2.5.2",
36     "leaflet": "^1.9.4",
37     "lucide-react": "^0.446.0",
38     "next": "^14.2.3",
39     "next-pwa": "^5.6.0",
40     "react": "^18
  
```

Figura 1. Tecnologías y librerías usadas
Fuente(s): Elaboración propia

3. Estructura del proyecto

La organización del proyecto está diseñada para facilitar la navegación y el mantenimiento del código. Dentro de la carpeta principal del proyecto se encuentran las siguientes secciones clave. En la carpeta `src/app`, se alojan todas las páginas principales de la aplicación, como el Dashboard y las secciones de gestión. En la carpeta `src/components`, están disponibles componentes reutilizables como botones, formularios y tablas. Las rutas API se encuentran en la carpeta `src/api` y manejan la comunicación entre el frontend y el backend. Finalmente, el esquema de la base de datos está definido en la carpeta `prisma`, utilizando archivos `.prisma` para modelar las entidades y sus relaciones.

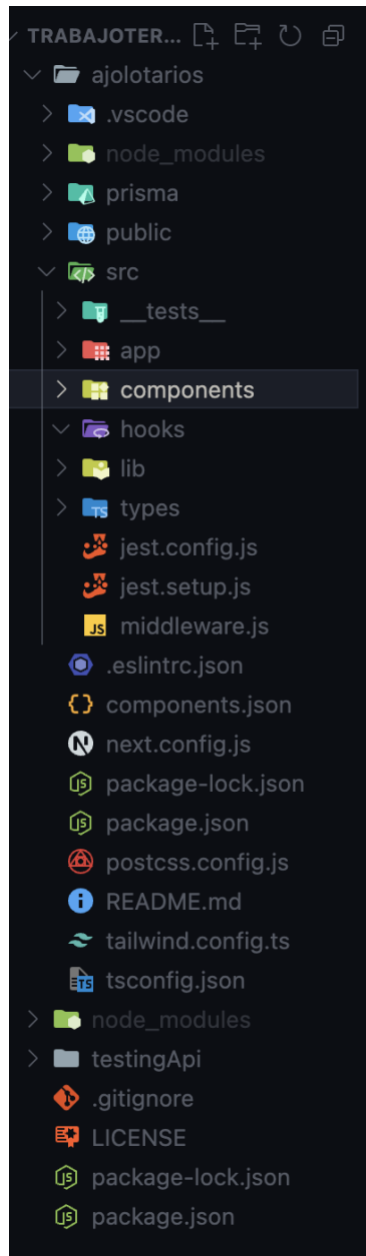
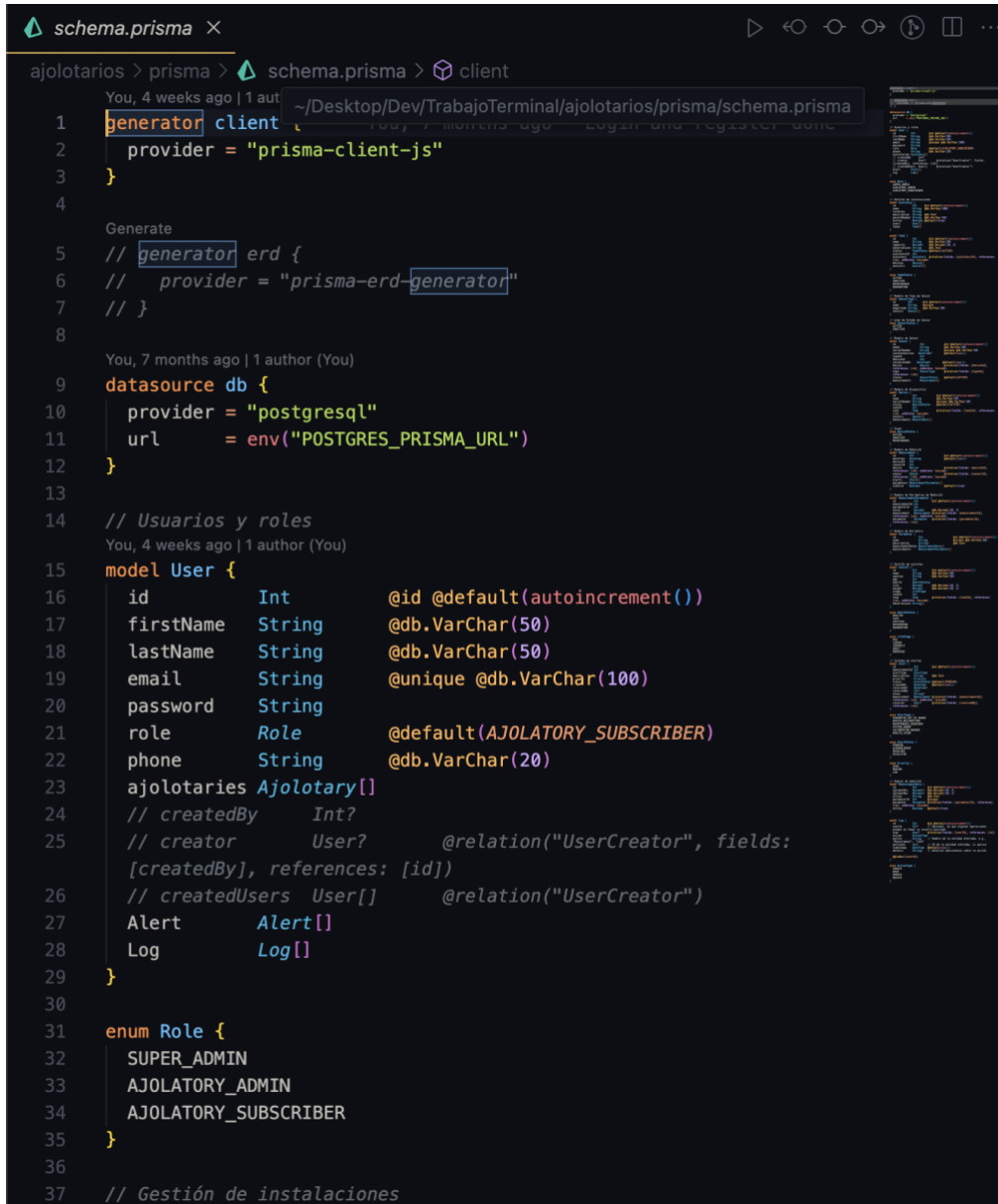


Figura 2. Estructura del proyecto
Fuente(s): Elaboración propia

4. Base de datos

La base de datos de AjoloApp utiliza PostgreSQL, gestionada con Prisma ORM para garantizar eficiencia y flexibilidad. Las principales entidades incluyen User, que administra a los usuarios con atributos como nombre, correo electrónico, rol y contraseña encriptada. La entidad Ajolotary representa las instalaciones, almacenando información como su ubicación, descripción y estado operativo. Los tanques, definidos en la entidad Tank, están asociados a los ajolotarios y contienen detalles como capacidad y estado. Los sensores, descritos en la entidad Sensor, se vinculan a los tanques para monitorear parámetros específicos. Además, las mediciones y alertas son gestionadas a través de las entidades Measurement y Alert, que permiten almacenar los datos recolectados y generar notificaciones críticas.

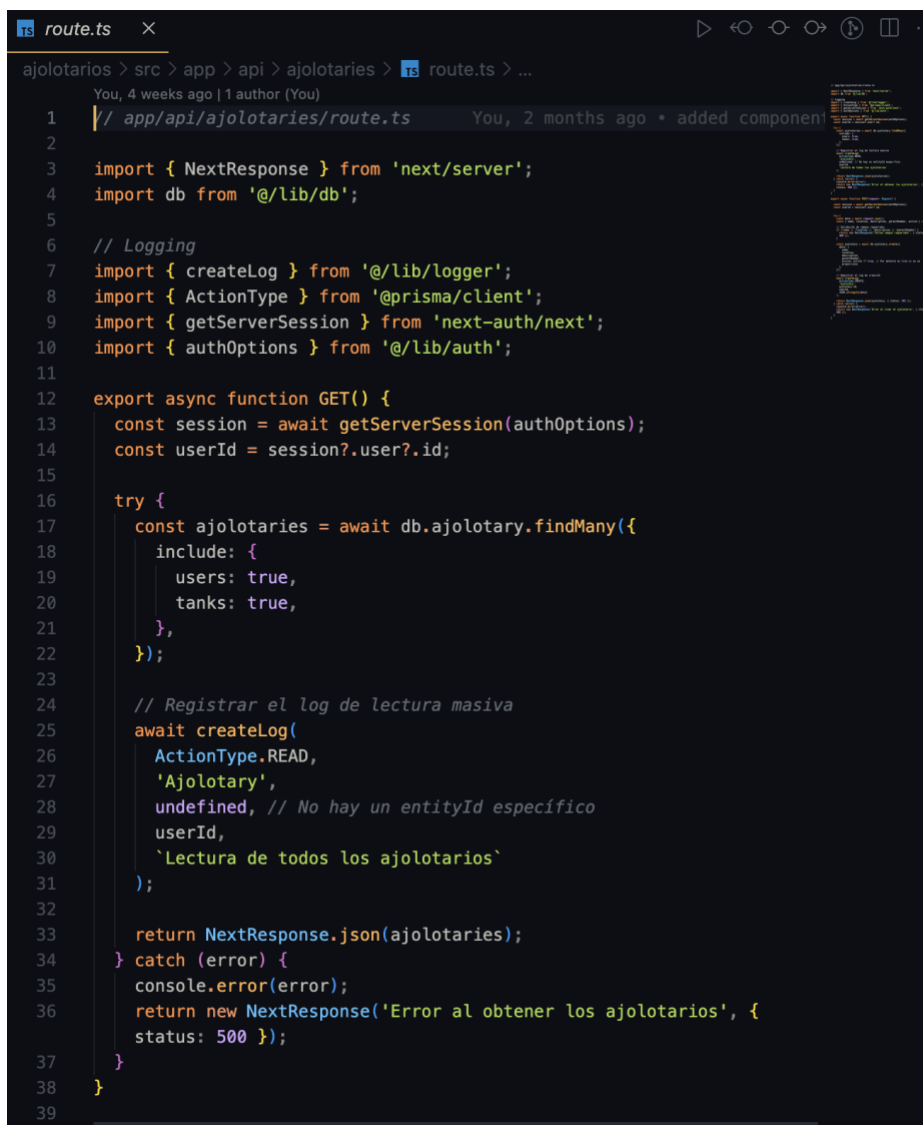


```
schema.prisma X
ajolotarios > prisma > schema.prisma > client
You, 4 weeks ago | 1 author (You)
1 generator client {
2   provider = "prisma-client-js"
3 }
4
5 // generator erd {
6 //   provider = "prisma-erd-generator"
7 // }
8
9 You, 7 months ago | 1 author (You)
10 datasource db {
11   provider = "postgresql"
12   url      = env("POSTGRES_PRISMA_URL")
13 }
14
15 // Usuarios y roles
16 You, 4 weeks ago | 1 author (You)
17 model User {
18   id          Int          @id @default(autoincrement())
19   firstName   String       @db.VarChar(50)
20   lastName    String       @db.VarChar(50)
21   email       String       @unique @db.VarChar(100)
22   password    String
23   role        Role         @default(AJOLATORY_SUBSCRIBER)
24   phone       String       @db.VarChar(20)
25   ajolotaries Ajolotary[]
26   // createdBy Int?
27   // creator    User? @relation("UserCreator", fields:
28   [createdBy], references: [id])
29   // createdUsers User[] @relation("UserCreator")
30   Alert        Alert[]
31   Log          Log[]
32 }
33
34 enum Role {
35   SUPER_ADMIN
36   AJOLATORY_ADMIN
37   AJOLATORY_SUBSCRIBER
38 }
39
40 // Gestión de instalaciones
```

Figura 3. Esquema de DB en prisma
Fuente(s): Elaboración propia

5. Funcionalidades del backend

El backend está diseñado con rutas API que proporcionan endpoints REST para cada entidad principal. Por ejemplo, la ruta `/api/ajolotaries` permite obtener o crear nuevos ajolotarios, mientras que `/api/sensors/[id]` se utiliza para actualizar o eliminar sensores específicos. Estas rutas están protegidas mediante middleware de autenticación, garantizando que solo los usuarios autorizados puedan acceder a ellas. Además, las notificaciones en tiempo real se implementan mediante integraciones con Telegram y WhatsApp. Estas funciones utilizan llamadas HTTP seguras para enviar mensajes de alerta directamente a los dispositivos móviles de los usuarios.



```
route.ts
ajolotarios > src > app > api > ajolotaries > route.ts > ...
You, 4 weeks ago | 1 author (You)
1 // app/api/ajolotaries/route.ts You, 2 months ago • added component
2
3 import { NextResponse } from 'next/server';
4 import db from '@lib/db';
5
6 // Logging
7 import { createLog } from '@lib/logger';
8 import { ActionType } from '@prisma/client';
9 import { getServerSession } from 'next-auth/next';
10 import { authOptions } from '@lib/auth';
11
12 export async function GET() {
13   const session = await getServerSession(authOptions);
14   const userId = session?.user?.id;
15
16   try {
17     const ajolotaries = await db.ajolotary.findMany({
18       include: {
19         users: true,
20         tanks: true,
21       },
22     });
23
24     // Registrar el log de lectura masiva
25     await createLog(
26       ActionType.READ,
27       'Ajolotary',
28       undefined, // No hay un entityId específico
29       userId,
30       'Lectura de todos los ajolotarios'
31     );
32
33     return NextResponse.json(ajolotaries);
34   } catch (error) {
35     console.error(error);
36     return new NextResponse('Error al obtener los ajolotarios', {
37       status: 500
38     });
39   }
40 }
```

Figura 4. Ejemplo api route
Fuente(s): Elaboración propia

6. Frontend

El frontend de AjoloApp está compuesto por páginas React que interactúan con las rutas API del backend. Cada página está diseñada para manejar la lógica de negocio correspondiente, incluyendo validaciones de formularios y actualización del estado. El Dashboard sirve como la interfaz principal, mostrando datos resumidos y herramientas de visualización como gráficas y mapas. Las secciones de gestión permiten a los usuarios agregar o editar información, mientras que los componentes personalizados aseguran un diseño consistente en toda la aplicación. Estos componentes incluyen botones, tarjetas y tablas que se pueden reutilizar en múltiples secciones.

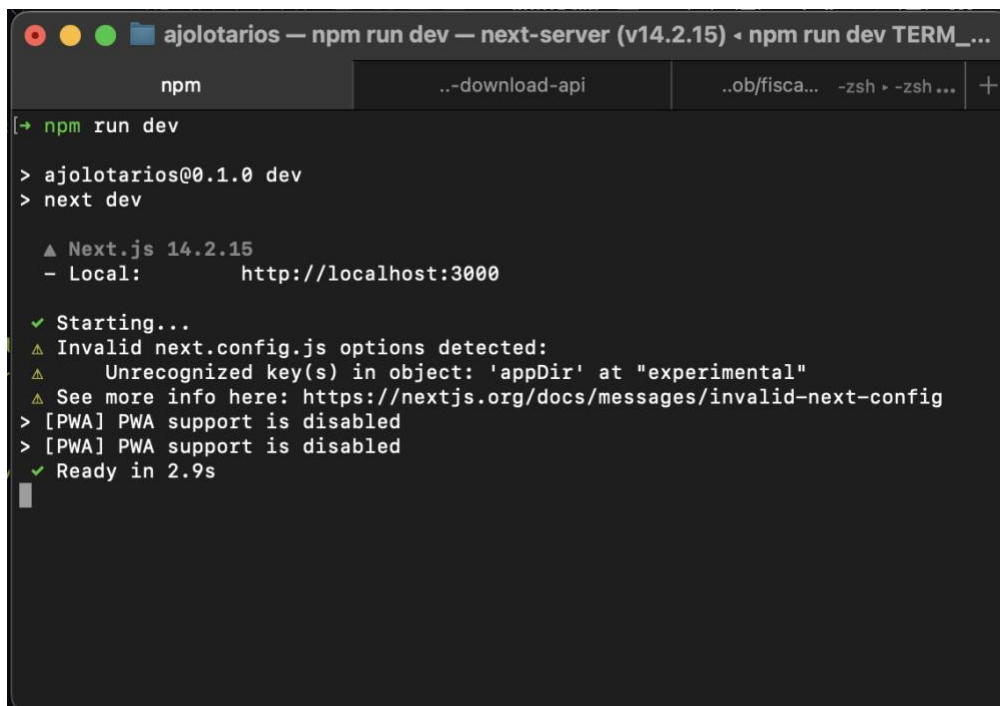


```
page.tsx x
ajolotarios > src > app > page.tsx > Dashboard > generatePDF > then() callback
You, 7 days ago | 1 author (You)
1 // src/app/dashboard/page.tsx
2
3 "use client";
4
5 import { useEffect, useState } from "react";
6 import { Bell, FileText, Thermometer, Droplet } from "lucide-react";
7 import { Card, CardContent, CardHeader, CardTitle } from "@components/
  ui/card";
8 import { Button } from "@components/ui/button";
9 import { Avatar, AvatarFallback, AvatarImage } from "@components/ui/
  avatar";
10 import dynamic from "next/dynamic";
11 import LoadingSpinner from "@components/LoadingSpinner";
12 import AjolotarySelector from "@components/AjolotarySelector";
13 import MeasurementHistory from "@components/MeasurementHistory";
14 import LogHistory from "@components/LogHistory";
15 import jsPDF from "jspdf";
16 import html2canvas from "html2canvas";
17
18 import {
19   Ajolotary,
20   Tank,
21   Axolotl,
22   Alert,
23   Measurement,
24   Sensor,
25 } from "@types/types";
26
27 import DashboardCharts from "@components/DashboardCharts";
28
29 const Map = dynamic(() => import("@components/Map"), {
30   ssr: false,
31   loading: () => <LoadingSpinner />,
32 });
33
34 export default function Dashboard() {
35   const [ajolotaries, setAjolotaries] = useState<Ajolotary[]>([]);
36   const [tanks, setTanks] = useState<Tank[]>([]);
37   const [axolotls, setAxolotls] = useState<Axolotl[]>([]);
38   const [alerts, setAlerts] = useState<Alert[]>([]);
39   const [measurements, setMeasurements] = useState<Measurement[]>([]);
```

Figura 5. Vista de dashboard
Fuente(s): Elaboración propia

7. Configuración inicial

Para configurar AjoloApp en un entorno local, primero es necesario clonar el repositorio del proyecto. Después, se deben instalar las dependencias utilizando el comando `npm install`. El archivo `.env` debe configurarse con variables como `DATABASE_URL`, que define la conexión a la base de datos, y `NEXTAUTH_SECRET`, utilizado para las sesiones de usuario. Una vez configurado el entorno, se deben ejecutar las migraciones de Prisma para crear las tablas en la base de datos. Finalmente, el proyecto puede iniciarse con el comando `npm run dev`, que lo ejecuta en <http://localhost:3000>. Esto es para el desarrollo en local, pero la aplicación puede ser desplegada en cualquier tipo de nube, pero se debe usar el comando `npm run build`



```
ajolotarios — npm run dev — next-server (v14.2.15) ◀ npm run dev TERM_...
npm      ..-download-api      ..ob/fisca... -zsh ▶ -zsh... +
[→ npm run dev

> ajolotarios@0.1.0 dev
> next dev

  ▲ Next.js 14.2.15
  - Local:      http://localhost:3000

  ✓ Starting...
  ▲ Invalid next.config.js options detected:
  ▲   Unrecognized key(s) in object: 'appDir' at "experimental"
  ▲ See more info here: https://nextjs.org/docs/messages/invalid-next-config
  > [PWA] PWA support is disabled
  > [PWA] PWA support is disabled
  ✓ Ready in 2.9s
```

Figura 6. Correr proyecto en local

Fuente(s): Elaboración propia

8. Gestión de Usuarios

Las integraciones clave incluyen el uso de Prisma para la base de datos, Tailwind CSS para los estilos y NextAuth para la autenticación. Las notificaciones se envían mediante las APIs de Telegram y WhatsApp, utilizando llamadas `fetch`. La generación de informes se realiza con `jsPDF` y `html2canvas`, que convierten el contenido HTML en documentos PDF descargables. Estas herramientas están configuradas para garantizar una integración fluida y eficiente en el sistema.



```
page.tsx x
ajolotarios > src > app > users > page.tsx > ...
You, 7 days ago | 1 author (You)
1 // app/users/page.tsx
2 "use client";
3
4 import { useEffect, useState } from "react";
5 import {
6   Table,
7   TableBody,
8   TableCell,
9   TableHead,
10  TableHeader,
11  TableRow,
12 } from "@components/ui/table";
13 import {
14   Select,
15   SelectContent,
16   SelectItem,
17   SelectTrigger,
18   SelectValue,
19 } from "@components/ui/select";
20 import { Alert, AlertDescription } from "@components/ui/alert";
21
22 You, 7 days ago | 1 author (You)
23 interface User {
24   id: number;
25   firstName: string;
26   lastName: string;
27   email: string;
28   role: "SUPER_ADMIN" | "AJOLATORY_ADMIN" | "AJOLATORY_SUBSCRIBER";
29 }
30
31 export default function UsersPage() {
32   const [users, setUsers] = useState<User[]>([]);
33   const [loading, setLoading] = useState<boolean>(true);
34   const [error, setError] = useState<string | null>(null);
35   const [successMessage, setSuccessMessage] = useState<string | null>(null);
36
37   useEffect(() => {
38     fetchUsers();
39   }, []);
40
41   // ... (rest of the component implementation)
42 }
```

Figura 12. Componente de usuarios
Fuente(s): Elaboración propia

9. Buenas prácticas

Para mantener la seguridad y eficiencia de AjoloApp, es fundamental seguir ciertas buenas prácticas. Las contraseñas deben estar encriptadas y las variables de entorno no deben exponerse en repositorios públicos. El código debe seguir un estilo consistente, utilizando herramientas como ESLint para detectar errores. Además, es importante modularizar las funcionalidades para facilitar futuras expansiones y utilizar paginación en consultas de grandes volúmenes de datos. Finalmente, se recomienda realizar pruebas unitarias y de integración para asegurar la estabilidad del sistema.