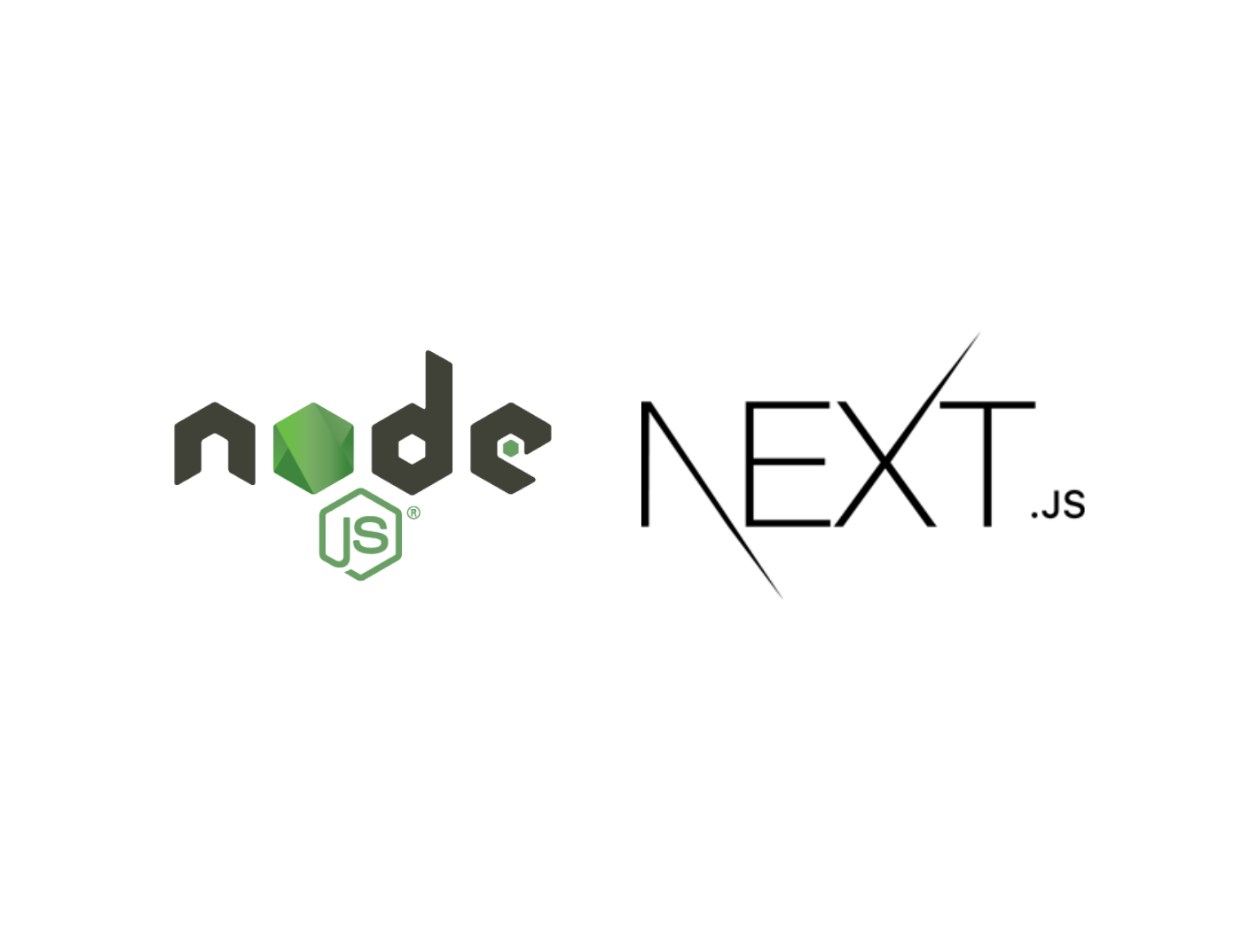
Logotipo, nombre de la empresa

Descripción generada automáticamente

**DESARROLLO DE APLICACIONES WEB AVANZADO**

**LABORATORIO N° 13**

**NEXTAUTH.JS, MANEJO DE COOKIES Y SESIONES**



|  |  |  |
| --- | --- | --- |
| **Alumnos** | | **Nota** |
|  | |  |
| **Grupo** |  | |
| **Fecha de Entrega** |  | |
| **Docente** | **Renato Usnayo Cáceres** | |

**OBJETIVOS:**

* Implementa sistemas seguros de autenticación y manejo de sesiones en aplicaciones web utilizando herramientas como NextAuth.js.

**SEGURIDAD:**

|  |  |
| --- | --- |
|  | **Advertencia:**  **En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red; así como la ingestión de alimentos o bebidas.** |

**FUNDAMENTO TEÓRICO:**

* Revisar el texto guía que está en el campus Virtual.

**NORMAS EMPLEADAS:**

* No aplica

**RECURSOS:**

* En este laboratorio cada alumno trabajará con un equipo con Windows 11.

**METODOLOGÍA PARA EL DESARROLLO DE LA TAREA:**

* El desarrollo del laboratorio es en parejas

**PROCEDIMIENTO:**

**Nota:**

|  |
| --- |
| Las secciones en cursivas son demostrativas, pero sirven para que usted pueda instalar las herramientas de desarrollo en un equipo externo. |

**Procedimiento:**

1. Instalar Next.js (si no lo tienes)

|  |
| --- |
| npx create-next-app@latest mi-proyecto-nextauth  cd mi-proyecto-nextauth  npm install next-auth  npm run dev |

npm run dev iniciará el servidor en http://localhost:3000/.

Adjunte captura de la ejecución

|  |
| --- |
|  |

1. **Paso 1: Configurar NextAuth.js en Next.js**

* NextAuth requiere una API personalizada para manejar autenticación.
* Debemos crear un archivo en /pages/api/auth/[...nextauth].js

Archivo: /pages/api/auth/[...nextauth].js

|  |
| --- |
| import NextAuth from "next-auth";  import GoogleProvider from "next-auth/providers/google";  export default NextAuth({  providers: [  GoogleProvider({  clientId: process.env.GOOGLE\_CLIENT\_ID,  clientSecret: process.env.GOOGLE\_CLIENT\_SECRET,  }),  ],  secret: process.env.NEXTAUTH\_SECRET,  callbacks: {  async session({ session, token }) {  session.user.id = token.sub;  return session;  },  },  }); |

¿Qué hace este código?

* Configura Google OAuth como método de autenticación.
* Agrega un callback para incluir el id del usuario en la sesión.

Configurar Variables de Entorno (.env.local)

Archivo: .env.local

|  |
| --- |
| GOOGLE\_CLIENT\_ID=tu-client-id  GOOGLE\_CLIENT\_SECRET=tu-client-secret  NEXTAUTH\_SECRET=clave-secreta-aleatoria |

Para obtener GOOGLE\_CLIENT\_ID y GOOGLE\_CLIENT\_SECRET:

* Ir a Google Developers Console.
* Crear un nuevo proyecto.
* Habilitar "Google OAuth".
* Configurar los URIs de redirección como

|  |
| --- |
| http://localhost:3000/api/auth/callback/google |

Copiar y pegar las credenciales en .env.local.

Reinicia el servidor después de modificar el archivo .env.local:

|  |
| --- |
| npm run dev |

Adjunte captura de la ejecución

|  |
| --- |
|  |

1. **Paso 2: Crear Página de Inicio de Sesión**

Archivo: /pages/login.js

|  |
| --- |
| import { signIn } from "next-auth/react";  export default function LoginPage() {  return (  <div style={{ textAlign: "center", marginTop: "100px" }}>  <h1>Iniciar Sesión</h1>  <button onClick={() => signIn("google")}>Iniciar con Google</button>  </div>  );  } |

¿Qué hace este código?

* Usa signIn("google") para autenticar al usuario con Google.
* Redirige a la página de autenticación de Google.

1. **Paso 3: Crear Página Protegida (Dashboard)**

Archivo: /pages/dashboard.js

|  |
| --- |
| import { useSession, signOut } from "next-auth/react";  export default function Dashboard() {  const { data: session } = useSession();  if (!session) {  return <p>No tienes acceso. Inicia sesión.</p>;  }  return (  <div style={{ textAlign: "center", marginTop: "50px" }}>  <h1>Bienvenido, {session.user.name}</h1>  <p>Email: {session.user.email}</p>  <img src={session.user.image} alt="Foto de perfil" width={100} />  <br />  <button onClick={() => signOut()}>Cerrar Sesión</button>  </div>  );  } |

¿Qué hace este código?

* Usa useSession() para obtener la sesión del usuario.
* Si no hay sesión, muestra un mensaje de error.
* Si hay sesión, muestra los datos del usuario y un botón para cerrar sesión.

1. **Paso 4: Proteger Rutas con Middleware**

Para evitar que usuarios no autenticados accedan a páginas protegidas, podemos usar middleware.

Archivo: middleware.js

|  |
| --- |
| import { withAuth } from "next-auth/middleware";  export default withAuth({  pages: {  signIn: "/login",  },  }); |

¿Qué hace este código?

* Redirige a /login si el usuario no está autenticado.
* Aplica la protección a todas las rutas automáticamente.

1. **Paso 5: Manejo de Cookies y Sesiones**

NextAuth almacena la sesión usando cookies seguras.

Ver las cookies en el navegador:

* Abre http://localhost:3000/dashboard.
* Inspecciona el sitio (F12 → Application → Cookies).
* Verás una cookie llamada next-auth.session-token.

Manejo de sesiones con getSession (Ejemplo en API Route)

Archivo: /pages/api/user.js

|  |
| --- |
| import { getSession } from "next-auth/react";  export default async function handler(req, res) {  const session = await getSession({ req });  if (!session) {  return res.status(401).json({ message: "No autorizado" });  }  res.status(200).json({ user: session.user });  } |

* Devuelve datos del usuario autenticado.
* Si no hay sesión, responde con 401 Unauthorized.

Adjunte captura de la ejecución

|  |
| --- |
|  |

**TAREA**

**Agregar GitHub como Proveedor de Autenticación**

Objetivo: Configurar NextAuth.js para permitir autenticación con GitHub además de Google.

Requisitos:

* Registrar una aplicación en GitHub en GitHub Developer Settings.
* Obtener GITHUB\_CLIENT\_ID y GITHUB\_CLIENT\_SECRET.
* Modificar /pages/api/auth/[...nextauth].js para agregar GitHub como proveedor.

**Personalizar la página de inicio de sesión (signIn con un UI mejorado).**

**Implementar Roles de Usuario (Admin y Usuario)**

Objetivo: Modificar el sistema de autenticación para asignar roles a los usuarios y restringir el acceso a ciertas páginas.

Requisitos:

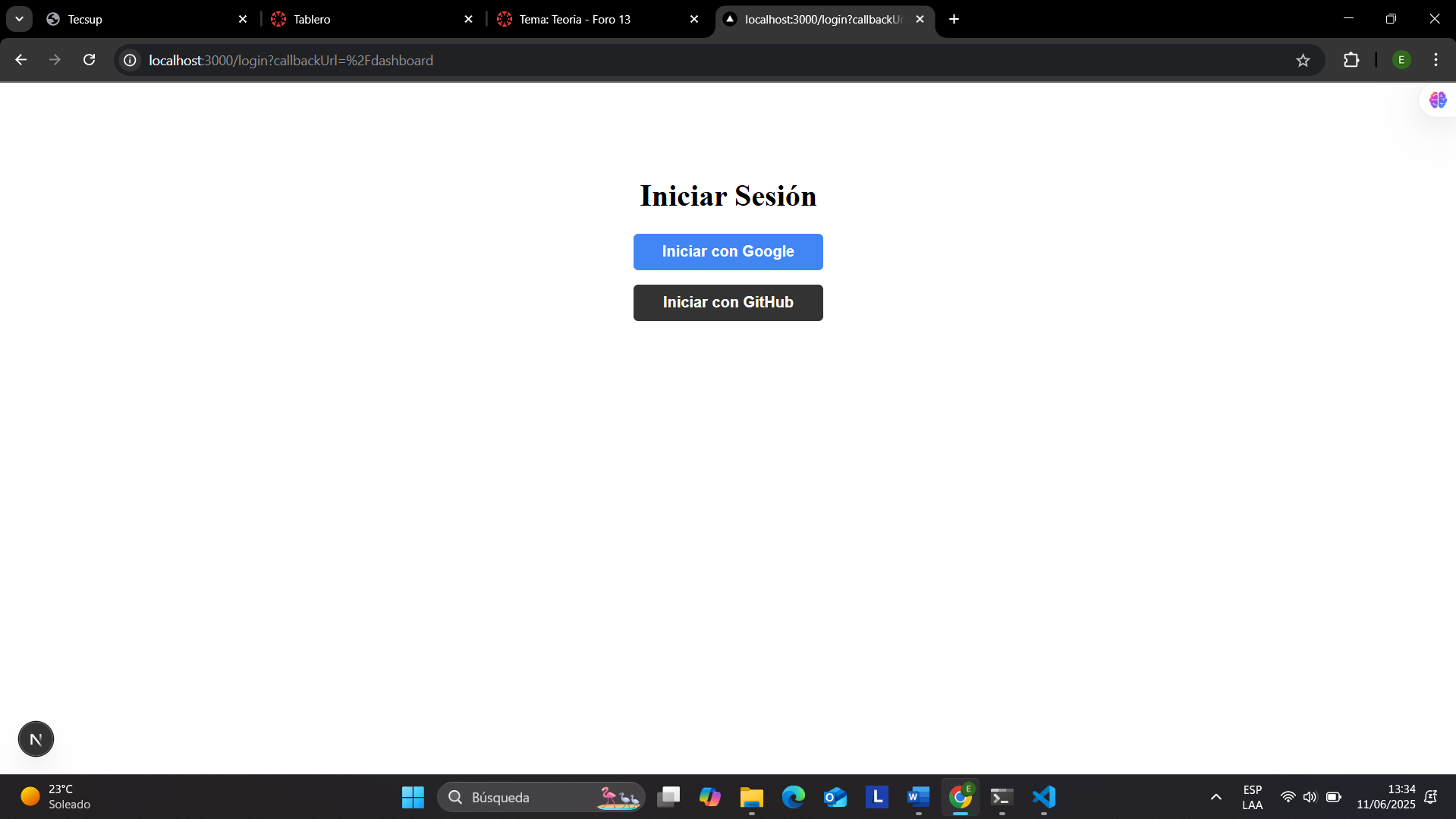
* Agregar un campo role (admin o user) en la sesión del usuario.
* Restringir el acceso a /admin solo para usuarios con rol "admin".

Ejemplo de validación en /pages/admin.js

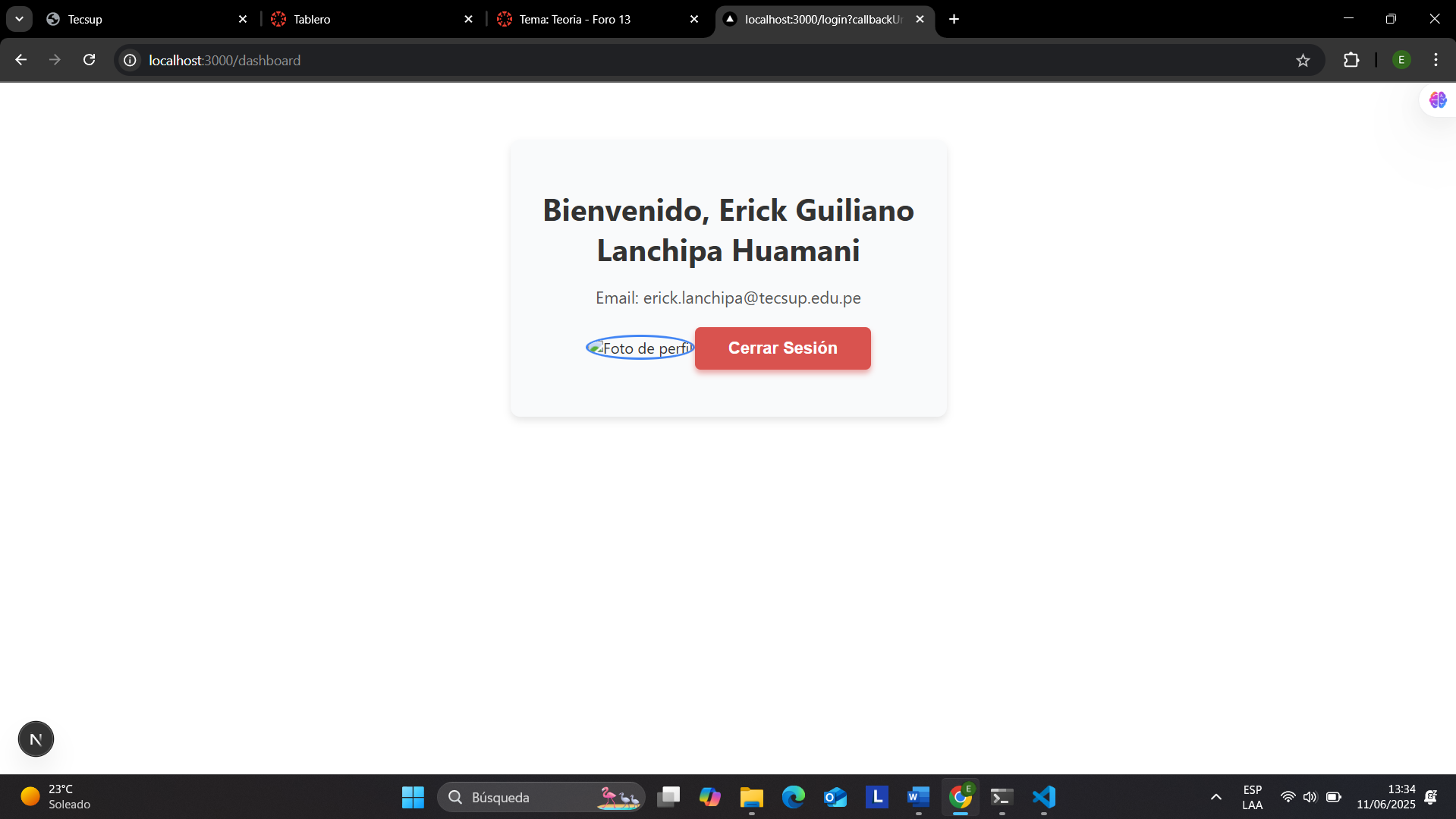
|  |
| --- |
| import { useSession } from "next-auth/react";  export default function AdminPage() {  const { data: session } = useSession();  if (!session || session.user.role !== "admin") {  return <p>Acceso denegado. Debes ser administrador.</p>;  }  return <h1>Panel de Administración</h1>;  } |

**Agregar una opción en el frontend para que un administrador pueda cambiar el rol de un usuario.**

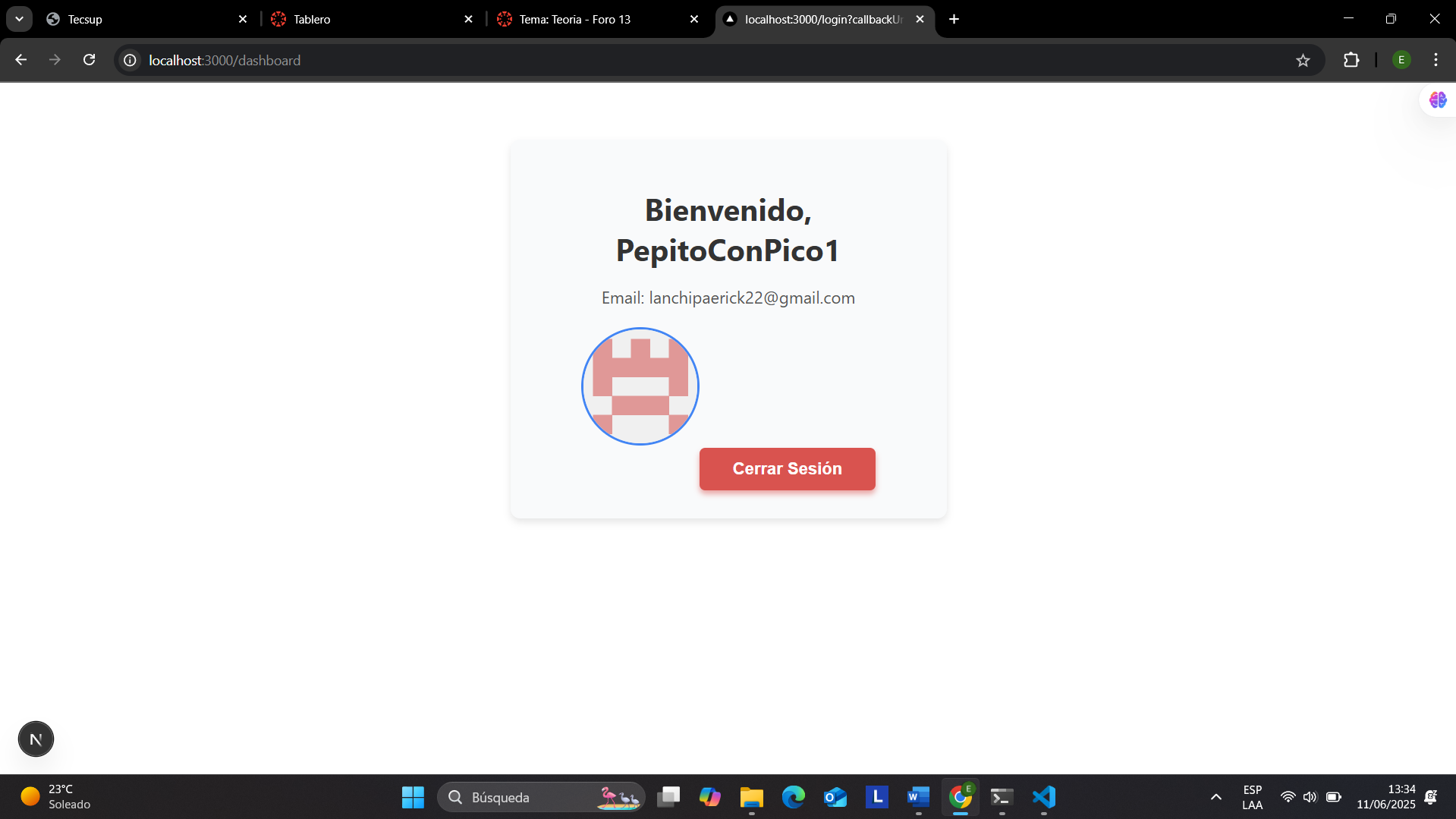
***Colocar capturas del código y capturas de ejecución***

******

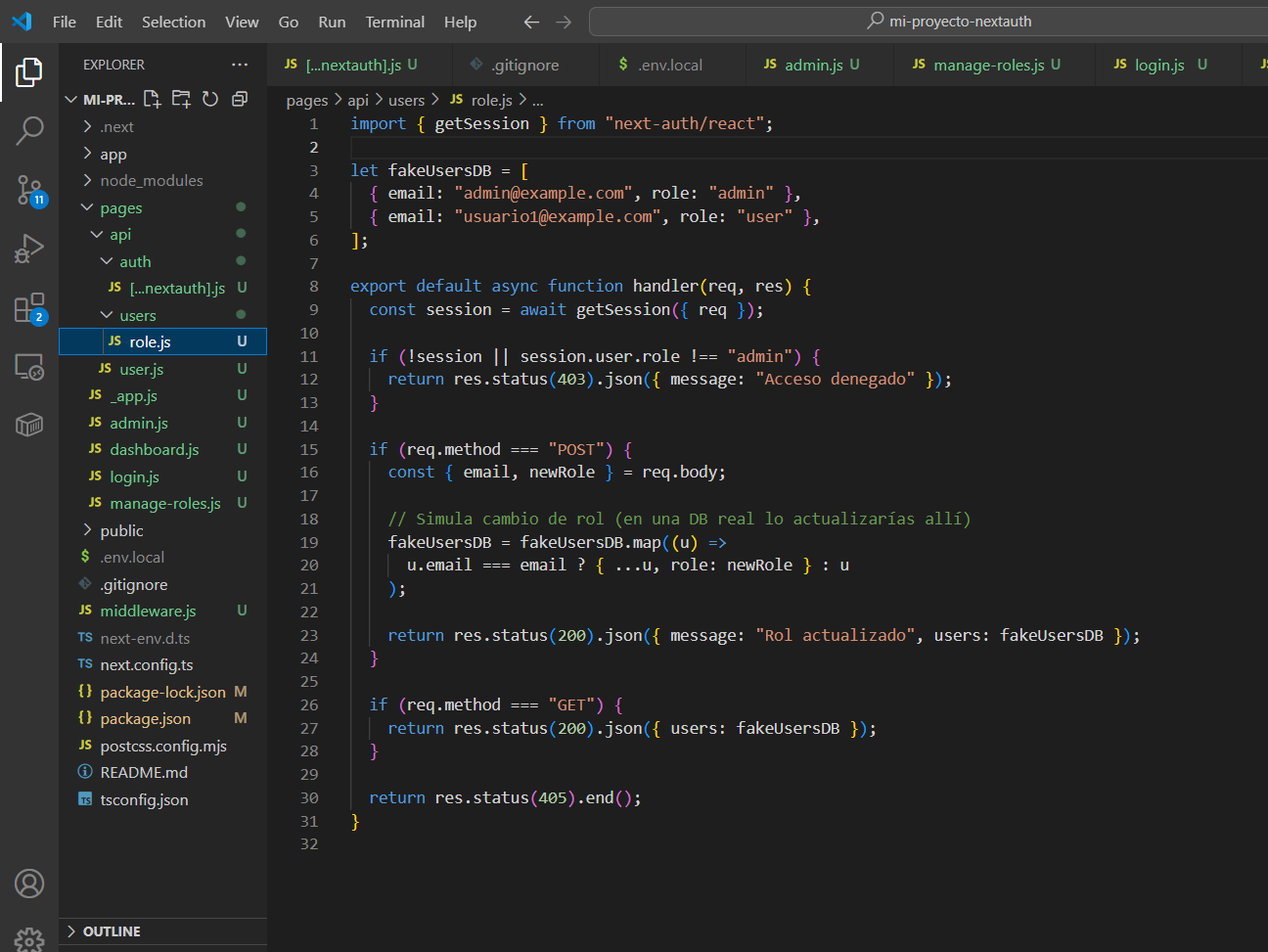
***GOOGLE:***

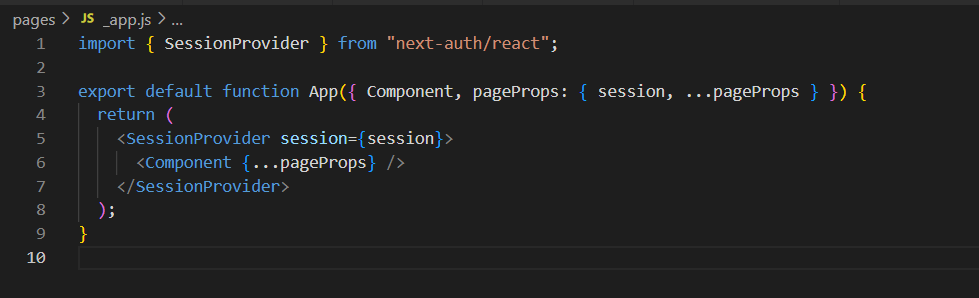
******

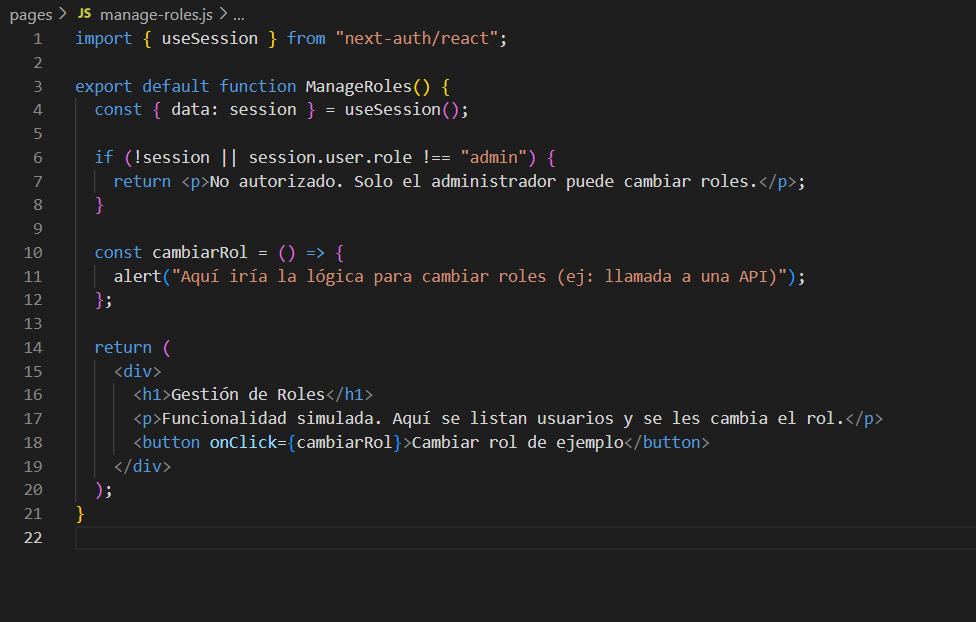
***GITHUB:***

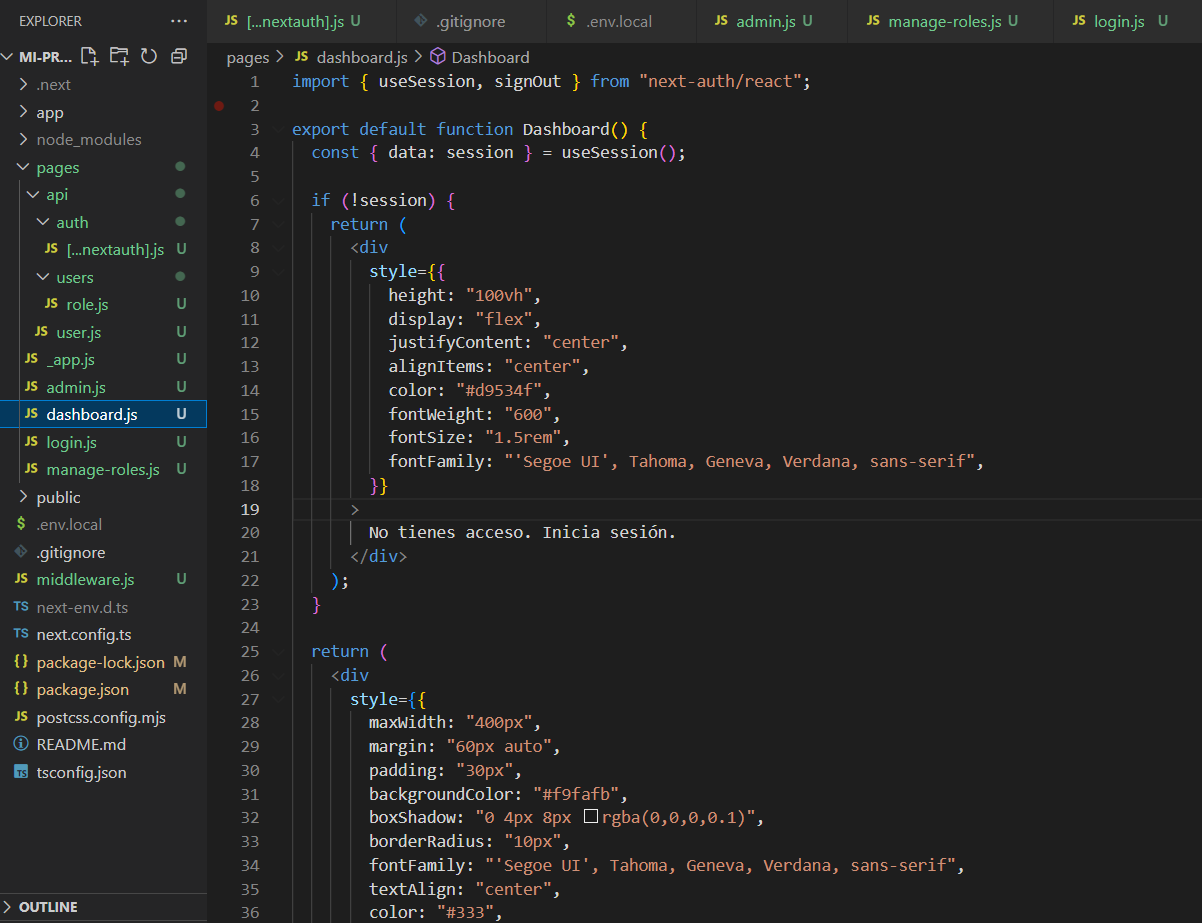
******

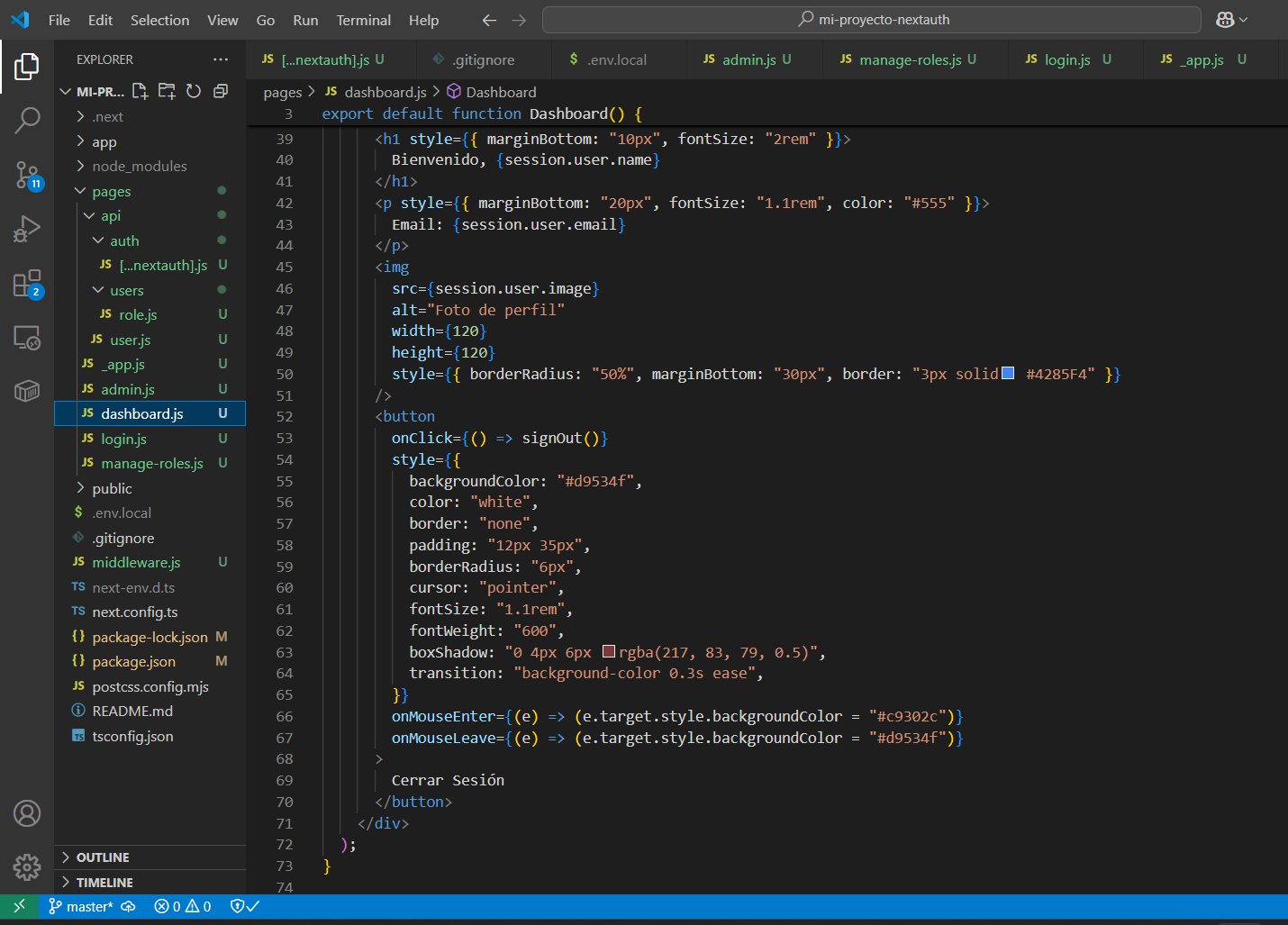
***CODIGO:***

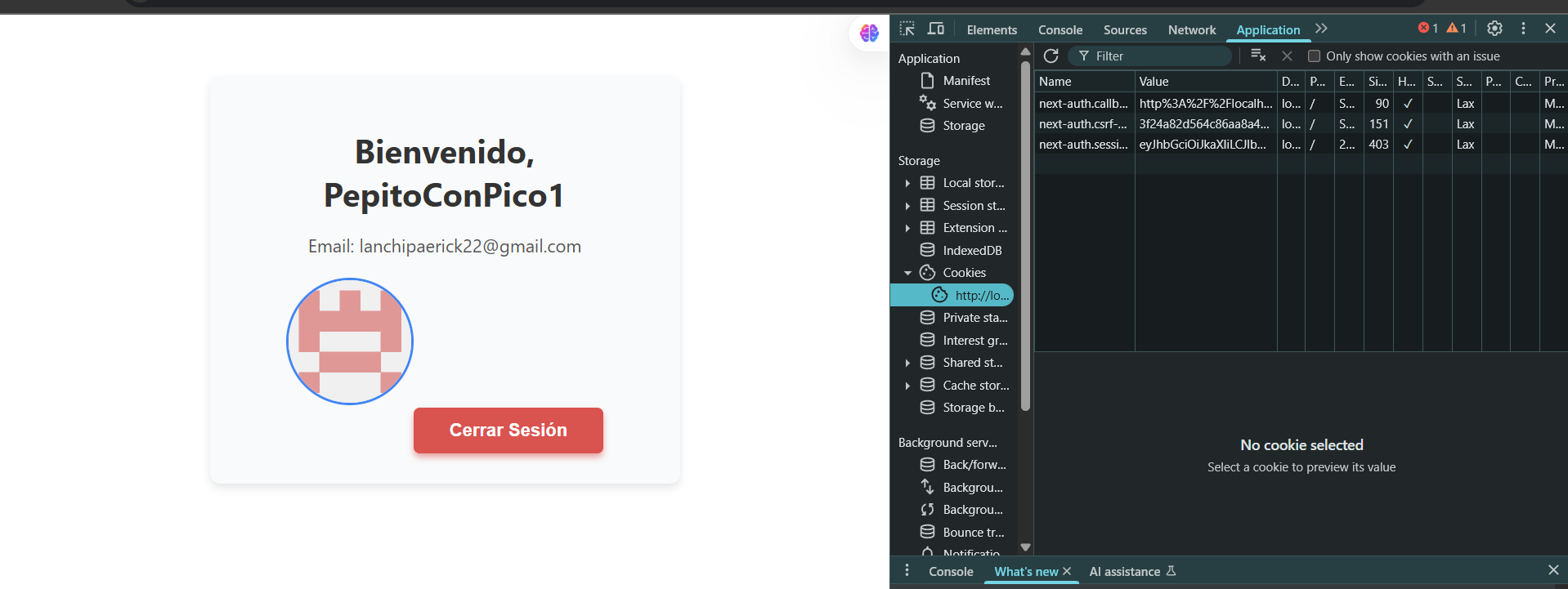
******

******

******

******

******

******

***Grabar explicando el código y ejecución de este en máximo 4 minutos***

**OBSERVACIONES:** *(Las observaciones son las notas aclaratorias, objeciones y problemas que se pudo presentar en el desarrollo del laboratorio)*

- Se presentó un error de codigo debido al uso inapropiado del hook useSession en páginas que no estaban correctamente condicionadas para el entorno del cliente.

- En la página de login, al cerrar sesión e intentar iniciar nuevamente, la UI no se actualizaba -correctamente sin recargar manualmente la página.

- Se observó que, si el usuario ya estaba autenticado, la página de login aún se mostraba brevemente antes de redirigir al dashboard.

- Se solucionaron los errores agregando condiciones basadas en el estado status de useSession y redireccionando desde el login automáticamente.

- Se conservaron los estilos personalizados en los botones para mantener la coherencia visual con el diseño planteado.

**CONCLUSIONES:** *(Las conclusiones son una opinión sobre tu trabajo, explicar cómo resolviste las dudas o problemas presentados en el laboratorio. Además de aportar una opinión critica de lo realizado)*

- El laboratorio permitió comprender de forma práctica cómo manejar la autenticación con múltiples proveedores (Google y GitHub) utilizando NextAuth en Next.js.

- Los problemas presentados con el renderizado y el estado de sesión fueron solucionados utilizando useEffect, useSession y useRouter de forma coordinada.

- Se evidenció la importancia de manejar correctamente el ciclo de vida del renderizado para evitar errores de hidración, especialmente en aplicaciones SSR.

- Se reforzó la idea de que los componentes como <SessionProvider> deben envolver toda la aplicación para que los hooks de autenticación funcionen correctamente.

- En general, el laboratorio fue exitoso y permitió implementar una solución funcional, segura y visualmente agradable para el sistema de inicio de sesión y roles.