Frontend de Laredu: Introducción y Entorno

Antes de meternos de lleno en **React 19** y **Tailwind CSS 4**, vamos a preparar un **entorno claro** para el **frontend** de la plataforma **Laredu**.
Este primer mensaje cubre:

1. **Requisitos Previos** (Node, npm, etc.)
2. **Estructura de Carpetas** (en laredu/frontend/)
3. **Explicación General** del proceso que seguiremos (React 19 + Vite + Tailwind 4).

Luego, en mensajes posteriores, mostraremos **código detallado** y **explicaciones** paso a paso.

Requisitos Previos

Node.js

- Versión estable LTS recomendada (ej. 18 o 20).
- Para verificar: `node -v`

npm

- Instalada junto a Node.js. Versión 8+ recomendada.
- Para verificar: npm -v
- Editor de código (Visual Studio Code, WebStorm, etc.)
- Conexión a nuestro backend de Laravel 11 corriendo en http://127.0.0.1:8000 (o el puerto que hayas configurado).

Estructura de Carpetas

Dado que en el backend tenemos la carpeta:

```
laredu/
├── backend/
```

Crearemos una segunda carpeta:

```
laredu/
├── backend/   # Código Laravel 11
└── frontend/  # Código React 19
```

Dentro de frontend/, ubicaremos todo el proyecto **React**

Qué vamos a Hacer

- ***Crear** un proyecto nuevo con **Vite**, usando **React** (versión 19) y **TypeScript**.*
- ***Instalar Tailwind CSS 4** siguiendo sus nuevas guías.*
- ***Configurar** un proyecto funcional, que sepa hacer llamadas al **API** de Laravel (Sanctum) y mostrar resultados en pantalla.*
- ***Revisar** el CORS para garantizar que el cliente React pueda acceder al backend en http://127.0.0.1:8000.*

Creación del Proyecto con Vite y React 19

Ahora configuraremos el entorno de **React 19** usando **Vite**, dentro de la carpeta laredu/frontend/. Siguiendo la documentación oficial de Vite y la guía de **React 19**:

Entrar a la carpeta frontend

En tu terminal, navega hasta la carpeta raíz del proyecto, luego entra a laredu/frontend/:

```
cd laredu/frontend
```

Inicializar el Proyecto con Vite

Ejecuta el comando para crear un nuevo proyecto con **Vite**. Debido a que la forma oficial actual omite npx para la última versión de Vite, podemos usar la siguiente sintaxis propuesta (o descargar un instalador según Vite Labs):

```
npm create vite@latest .
```

Cuando lo corras, te preguntará cosas como:

```
✓ Select a framework: › React
```

```
✓ Select a variant: › TypeScript
```

Elige React como framework y **TypeScript** como variante.

Nota: Si en el futuro Vite actualiza el comando, revisa la documentación oficial para la sintaxis correcta. De momento, npm create vite@latest . es lo más común.

Instalar Dependencias Iniciales

Vite creará un package.json con dependencias como react, react-dom, vite, etc.

Ejecuta: `npm install`

Para que se descarguen e instalen todos los módulos.

Estructura Inicial

Tras esto, tu carpeta frontend/ tendrá algo como:

```
frontend/
├── index.html
├── package.json
├── tsconfig.json
├── src/
│   ├── App.tsx
│   ├── main.tsx
│   └── vite-env.d.ts
├── vite.config.ts
└── node_modules/
```

Revisemos cada parte importante:

1. **index.html**: Punto de entrada de la app Vite.
2. **src/App.tsx**: Componente principal de React.
3. **src/main.tsx**: Renderiza <App /> en el DOM.
4. **vite.config.ts**: Configuración de Vite (puedes personalizar puertos, proxies, etc.).

Ejecutar el Servidor de Desarrollo

Para verificar que todo funciona correctamente:

```
npm run dev
```

Vite mostrará algo como: `VITE v4.4.0  ready in 350 ms  ➜  Local: http://127.0.0.1:5173/`

Preparar .gitignore y Estructura

- **Asegúrate** de que tengas node_modules/ en tu .gitignore (en laredu/frontend/.gitignore o en la raíz).
- **Mantén** la carpeta frontend/ separada para que no se confundan archivos de Laravel con React.

Resultado: Tienes un proyecto **React 19** corriendo en http://127.0.0.1:5173, listo para integrarle **Tailwind CSS 4** y conectarlo con el backend de **Laravel 11**.

Instalación y Configuración de Tailwind CSS 4

A continuación, configuraremos **Tailwind CSS 4** en nuestro proyecto de **React 19** con **Vite**, siguiendo la documentación oficial de Tailwind v4.

Instalar Tailwind 4 y Dependencias

Dentro del directorio de tu proyecto, instala Tailwind CSS junto con el plugin oficial para Vite:

```
npm install tailwindcss @tailwindcss/vite
```

Configura Vite para usar el plugin de Tailwind CSS

Abre (o crea) el archivo de configuración de Vite, normalmente llamado vite.config.ts o vite.config.js, y agrega el plugin de Tailwind CSS. Si ya usas el plugin de React, asegúrate de incluir ambos:

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react-swc'
import tailwindcss from '@tailwindcss/vite';

export default defineConfig({
  plugins: [
    react(),
    tailwindcss(),
  ],
});
```

Importa Tailwind CSS en tu archivo CSS principal

Crea (o edita) el archivo CSS que usarás en tu aplicación (por ejemplo, src/index.css) e incluye la siguiente línea para importar Tailwind:

```
/* src/index.css */
```

@import "tailwindcss";

Nota:
En Tailwind CSS v4 la configuración es aún más sencilla gracias a la "configuración CSS-first". Si necesitas personalizar valores (colores, spacing, etc.), puedes hacerlo directamente en este archivo utilizando la directiva @theme.

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

Ejecuta el servidor de desarrollo

Con todo configurado, inicia el servidor de desarrollo con el siguiente comando:

npm run dev

Usa las clases de Tailwind en tus componentes React

Por ejemplo, edita tu componente App.jsx para aplicar algunas clases de Tailwind:

```
// src/App.jsx
function App() {
  return (
    <div className="min-h-screen bg-gray-100 flex items-center justify-
center">
      <h1 className="text-3xl font-bold text-blue-600">
        ¡Hola, Tailwind CSS v4 en React!
      </h1>
    </div>
  );
}

export default App;
```

Peticiones al Backend (Laravel 11) y Uso de Tokens Sanctum en React 19

Ahora que tenemos **React 19 + Vite + Tailwind CSS 4** configurados, vamos a **consumir** la API de Laravel 11 que creamos anteriormente. El objetivo es hacer peticiones a rutas como POST /api/login, POST /api/register, GET /api/courses, etc., enviando y almacenando el **token de Sanctum**.

Revisar la Configuración de CORS en Laravel

En tu backend, asegúrate de que cors.php contenga algo como:

```
return [
    'paths' => ['api/*', 'sanctum/csrf-cookie'],
    'allowed_methods' => ['*'],
    'allowed_origins' => [
        'http://127.0.0.1:5173',
        'http://localhost:5173',
    ],
    'allowed_headers' => ['*'],
    'supports_credentials' => false,
    // ...
];
```

De esa forma, las solicitudes desde React en 127.0.0.1:5173 no serán bloqueadas.

Manejo de Login (Autenticación con Tokens)

1 Ejemplo de un Componente de Login

Supongamos que tenemos un **componente** llamado Login.tsx que hace un POST a /api/login. Usaremos fetch (podríamos usar axios también).

```tsx
// src/components/Login.tsx
import React, { useState } from "react";

interface LoginProps {
    onLoginSuccess: (token: string) => void;
    // onLoginSuccess recibirá el token
}

export default function Login({ onLoginSuccess }: LoginProps) {
    // Estados locales para email y password
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [error, setError] = useState("");

    const handleSubmit = (e: React.FormEvent) => {
        e.preventDefault();

        // Petición POST a /api/login
        fetch("http://127.0.0.1:8000/api/login", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ email, password }),
        })
            .then((res) => {
                if (!res.ok) {
                    throw new Error("Invalid credentials");
                }
                return res.json();
            })
            .then((data) => {
                // data.token y data.user vendrán del backend
                onLoginSuccess(data.token);
            })
            .catch((err) => {
                setError(err.message);
            });
```

```
    };


    return (
        <div className="max-w-sm mx-auto bg-white p-4 rounded shadow">
            <h2 className="text-xl font-bold mb-4">Login</h2>
            {error && <p className="text-red-500 mb-2">{error}</p>}
       <form onSubmit={handleSubmit} className="flex flex-col space-y-3">
                <input
                    type="email"
                    placeholder="Email"
                    className="border p-2 rounded"
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                />
                <input
                    type="password"
                    placeholder="Password"
                    className="border p-2 rounded"
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                />
                <button
                    type="submit"
         className="bg-blue-600 text-white p-2 rounded hover:bg-blue-500"
                >
                    Login
                </button>
            </form>
        </div>
    );
}
```

2 Almacenar el Token

En React, podemos almacenar el token en **localStorage** o en un **state management** (Redux, Zustand, etc.). Para este ejemplo simple, utilizaremos localStorage.

Ejemplo: en App.tsx:

```
// src/App.tsx
import { useState } from "react";
import Login from "./components/Login";

function App() {
  const [token, setToken] = useState<string | null>(null);

  const handleLoginSuccess = (receivedToken: string) => {
```

```
    localStorage.setItem("token", receivedToken);
    setToken(receivedToken);
  };

  return (
    <div className="min-h-screen bg-gray-100 p-4">
      {!token ? (
        <Login onLoginSuccess={handleLoginSuccess} />
      ) : (
        <p className="text-xl">You are logged in. Token: {token}</p>
      )}
    </div>
  );
}

export default App;
```

Así, cuando el login es exitoso, guardamos receivedToken tanto en localStorage como en el state de App. Más adelante, podremos usar este token para proteger rutas y hacer peticiones a la API.

3 Hacer Peticiones con Token

Una vez tengamos el token, para consumir endpoints protegidos como /api/courses, realizamos algo como:

```
fetch("http://127.0.0.1:8000/api/courses", {
  headers: {
    Authorization: "Bearer " + localStorage.getItem("token"),
  },
})
  .then((res) => res.json())
  .then((data) => {
    console.log("Courses:", data);
  })
  .catch((err) => console.error(err));
```

Recuerda que en **Laravel** usamos Route::middleware('auth:sanctum') para rutas protegidas. Si la cabecera Authorization: Bearer <TOKEN> no está presente, el servidor devolverá un **401** (Unauthorized).

4 Ejemplo de Lista de Cursos

Para que sea más ilustrativo, creamos un componente CoursesList.tsx:

```tsx
// src/components/CoursesList.tsx
import { useEffect, useState } from "react";

interface Course {
    id: number;
    name: string;
    description: string;
}

export default function CoursesList() {
    const [courses, setCourses] = useState<Course[]>([]);
    const [error, setError] = useState("");

    useEffect(() => {
        const token = localStorage.getItem("token");
        if (!token) {
            setError("No token found. Please log in.");
            return;
        }

        fetch("http://127.0.0.1:8000/api/courses", {
            headers: {
                Authorization: "Bearer " + token,
            },
        })
            .then((res) => {
                if (!res.ok) {
                    throw new Error("Failed to fetch courses");
                }
                return res.json();
            })
            .then((data: Course[]) => {
                setCourses(data);
            })
            .catch((err) => setError(err.message));
    }, []);

    if (error) {
```

```
        return <p className="text-red-500">{error}</p>;
    }



 return (
        <div className="mt-4">
            <h2 className="text-2xl font-bold mb-2">Courses</h2>
            <ul className="space-y-2">
                {courses.map((course) => (
                    <li key={course.id} className="p-2 border rounded bg-
white shadow">
                        <strong>{course.name}</strong> -
{course.description}
                    </li>
                ))}
            </ul>
        </div>
    );
}
```

Luego, en App.tsx, podemos **renderizar** CoursesList si hay un token:

// ...

```
{token ? (
  <div>
    <p className="text-xl">Token: {token}</p>
    <CoursesList />
  </div>
) : (
  <Login onLoginSuccess={handleLoginSuccess} />
)}
```

Logout

Para **cerrar sesión**, bastaría con un **botón** que:

- Haga un POST a /api/logout con la cabecera Authorization: Bearer <TOKEN>.
- Limpie el localStorage.
- Actualice el state en React.

Crea el componente **`LogoutButton`**

```
// src/components/LogoutButton.tsx

interface LogoutButtonProps {
```

```
    onLogout: () => void;
}



export default function LogoutButton({ onLogout }: LogoutButtonProps) {
    return (
        <button
            onClick={onLogout}
            className="bg-red-600 text-white p-2 rounded hover:bg-red-
500"
        >
            Logout
        </button>
    );
}
```

// Modifica el archivo ***src/App.tsx*** en varias partes del código:

```
import LogoutButton from "./components/LogoutButton";
```

```
// Define la función handleLogout en el mismo archivo

  const handleLogout = () => {
    const token = localStorage.getItem("token");
    if (!token) return;

    fetch("http://127.0.0.1:8000/api/logout", {
      method: "POST",
      headers: {
        Authorization: "Bearer " + token,
      },
    })
      .then(() => {
        localStorage.removeItem("token");
        setToken(null);
      })
      .catch((err) => console.error(err));
  };
...
```

```
{token ? (
        <div>
          <p className="text-xl">Token: {token}</p>
```

```
          {/* Usa el componente LogoutButton pasando la función
handleLogout */}
          <LogoutButton onLogout={handleLogout} />
          <CoursesList />
        </div>
      ) : ( . . .
```

Creación de Componentes: Registro, Lista de Asignaturas y Entregas de Tareas

Ahora que tenemos autenticación con **Sanctum** funcionando en el **frontend de Laredu**, vamos a continuar creando más componentes esenciales:

1. **Registro de usuarios (Register.tsx)**
2. **Lista de asignaturas (SubjectsList.tsx)**
3. **Gestión de tareas y entregas (AssignmentsList.tsx, SubmissionsList.tsx)**

Cada componente **consumirá la API de Laravel 11** y trabajará con el **token** de usuario autenticado.

1. Registro de Usuarios (Register.tsx)

Objetivo

Crear un formulario donde los usuarios puedan **registrarse** en la plataforma.

Petición API

La API en **Laravel 11** expone el endpoint:
POST /api/register

Datos esperados:

```
{
  "name": "Nuevo Usuario",
  "email": "nuevo@example.com",
  "password": "password123",
  "role": "student"
}
```

Código del Componente: src/components/Register.tsx:

```
// src/components/Register.tsx
import React, { useState } from "react";

export default function Register() {
    const [name, setName] = useState("");
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [role, setRole] = useState("student"); // Por defecto,
estudiante
    const [message, setMessage] = useState("");

    const handleSubmit = (e: React.FormEvent) => {
        e.preventDefault();

        fetch("http://127.0.0.1:8000/api/register", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ name, email, password, role }),
        })
            .then((res) => res.json())
            .then(() => setMessage("Usuario registrado con éxito"))
            .catch(() => setMessage("Error en el registro"));
    };

    return (
        <div className="max-w-sm mx-auto bg-white p-4 rounded shadow">
            <h2 className="text-xl font-bold mb-4">Registro</h2>
            {message && <p className="text-green-500">{message}</p>}
            <form onSubmit={handleSubmit} className="flex flex-col space-
y-3">
                <input
                    type="text"
                    placeholder="Nombre"
                    className="border p-2 rounded"
                    value={name}
                    onChange={(e) => setName(e.target.value)}
                />
                <input
                    type="email"
```

```
                    placeholder="Email"
                    className="border p-2 rounded"
                    value={email}
                    onChange={(e) => setEmail(e.target.value)}
                />


                <input
                    type="password"
                    placeholder="Contraseña"
                    className="border p-2 rounded"
                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                />
                <label htmlFor="role" className="sr-only">Role</label>
                <select
                    id="role"
                    className="border p-2 rounded"
                    value={role}
                    onChange={(e) => setRole(e.target.value)}
                >
                    <option value="student">Estudiante</option>
                    <option value="teacher">Profesor</option>
                </select>
                <button type="submit" className="bg-green-600 text-white
p-2 rounded">
                    Registrarse
                </button>
            </form>
        </div>
    );
}
```

2. Lista de Asignaturas (SubjectsList.tsx)

Objetivo

Mostrar todas las asignaturas disponibles en la plataforma.

Petición API

GET /api/subjects

Ejemplo de respuesta:

[

```
  { "id": 1, "name": "Matemáticas", "course_id": 1,
"teacher_id": 3 },
  { "id": 2, "name": "Historia", "course_id": 2, "teacher_id": 5
}
]
```

Código del Componente: ***src/components/SubjectsList.tsx:***

```tsx
// src/components/SubjectsList.tsx
import React, { useEffect, useState } from "react";

interface Subject {
    id: number;
    name: string;
    course_id: number;
    teacher_id: number;
}

export default function SubjectsList() {
    const [subjects, setSubjects] = useState<Subject[]>([]);
    const [error, setError] = useState("");

    useEffect(() => {
        fetch("http://127.0.0.1:8000/api/subjects", {
            headers: {
                Authorization: "Bearer " + localStorage.getItem("token"),
            },
        })
            .then((res) => res.json())
            .then((data) => setSubjects(data))
            .catch(() => setError("Error al obtener las asignaturas"));
    }, []);

    return (
        <div className="mt-4">
            <h2 className="text-2xl font-bold mb-2">Asignaturas</h2>
            {error && <p className="text-red-500">{error}</p>}
            <ul className="space-y-2">
                {subjects.map((subject) => (
                    <li key={subject.id} className="p-2 border rounded
bg-white shadow">
                        <strong>{subject.name}</strong> - ID Curso:
{subject.course_id}
                    </li>
                ))}
```

```
            </ul>
        </div>
    );
}
```

3. Lista de Tareas y Entregas (AssignmentsList.tsx)

Objetivo

Mostrar las tareas disponibles y permitir ver las entregas de los estudiantes.

Petición API

GET /api/assignments

Ejemplo de respuesta:

[

{ "id": 1, "title": "Ejercicio Algebra", "due_date": "2025-02-10", "subject_id": 1 },

{ "id": 2, "title": "Ensayo Revolución Francesa", "due_date": "2025-02-12", "subject_id": 2 }

]

Código del Componente *src/components/AssignmentsList.tsx:*

```tsx
// src/components/AssignmentsList.tsx
import React, { useEffect, useState } from "react";

interface Assignment {
    id: number;
    title: string;
    due_date: string;
    subject_id: number;
}

export default function AssignmentsList() {
    const [assignments, setAssignments] = useState<Assignment[]>([]);
    const [error, setError] = useState("");

    useEffect(() => {
        fetch("http://127.0.0.1:8000/api/assignments", {
            headers: {
                Authorization: "Bearer " + localStorage.getItem("token"),
            },
```

```
        })
            .then((res) => res.json())
            .then((data) => setAssignments(data))
            .catch(() => setError("Error al obtener las tareas"));
    }, []);




    return (
        <div className="mt-4">
            <h2 className="text-2xl font-bold mb-2">Tareas</h2>
            {error && <p className="text-red-500">{error}</p>}
            <ul className="space-y-2">
                {assignments.map((assignment) => (
                    <li key={assignment.id} className="p-2 border rounded
bg-white shadow">
                        <strong>{assignment.title}</strong> - Fecha de
entrega: {assignment.due_date}
                    </li>
                ))}
            </ul>
        </div>
    );
}
```

Implementación de Entrega de Tareas y Mensajería Interna

Ahora vamos a continuar con la funcionalidad de **entrega de tareas (submissions)** y la **mensajería interna** para que los usuarios puedan comunicarse dentro de la plataforma **Laredu**.

1. Componente de Entrega de Tareas (SubmissionsList.tsx)

Objetivo

Los estudiantes podrán **entregar** tareas y ver el estado de sus envíos.

Petición API

POST /api/submissions

Ejemplo de datos a enviar:

```
{
  "user_id": 2,
```

```
  "assignment_id": 1,
  "submitted_at": "2025-02-10",
  "grade": null
}
```

Código del Componente *src/components/SubmissionsList.tsx:*

```tsx
// src/components/SubmissionsList.tsx
import React, { useEffect, useState } from "react";

interface Submission {
    id: number;
    assignment_id: number;
    user_id: number;
    submitted_at: string;
    grade: number | null;
}
export default function SubmissionsList() {
    const [submissions, setSubmissions] = useState<Submission[]>([]);
    const [assignmentId, setAssignmentId] = useState("");
    const [message, setMessage] = useState("");

    useEffect(() => {
        fetch("http://127.0.0.1:8000/api/submissions", {
            headers: {
                Authorization: "Bearer " + localStorage.getItem("token"),
            },
        })
            .then((res) => res.json())
            .then((data) => setSubmissions(data))
            .catch(() => setMessage("Error al obtener entregas"));
    }, []);
    const handleSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        fetch("http://127.0.0.1:8000/api/submissions", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
                Authorization: "Bearer " + localStorage.getItem("token"),
            },
            body: JSON.stringify({
```

```
                user_id: 2, // Esto debe cambiarse para tomar el ID del
usuario autenticado
                assignment_id: parseInt(assignmentId),
                submitted_at: new Date().toISOString(),
                grade: null,
            }),
        })
            .then((res) => res.json())
            .then(() => setMessage("Tarea entregada con éxito"))
            .catch(() => setMessage("Error al entregar tarea"));
    };

    return (
        <div className="mt-4">
            <h2 className="text-2xl font-bold mb-2">Entregas de
Tareas</h2>
            {message && <p className="text-green-500">{message}</p>}
            <form onSubmit={handleSubmit} className="mb-4 flex space-x-
2">
                <input
                    type="number"
                    placeholder="ID de la Tarea"
                    className="border p-2 rounded"
                    value={assignmentId}
                    onChange={(e) => setAssignmentId(e.target.value)}
                />
                <button type="submit" className="bg-blue-600 text-white
p-2 rounded">
                    Entregar Tarea
                </button>
            </form>
            <ul className="space-y-2">
                {submissions.map((submission) => (
                    <li key={submission.id} className="p-2 border rounded
bg-white shadow">
                        <strong>ID Tarea:
{submission.assignment_id}</strong> - Entregado el{" "}
                        {new
Date(submission.submitted_at).toLocaleDateString()} -{" "}
                        {submission.grade !== null ? `Nota:
${submission.grade}` : "Sin nota"}
                    </li>
                ))}
            </ul>
        </div>
    );
}
```

2. Componente de Mensajería (MessageList.tsx)

Objetivo

Los usuarios podrán **enviar y recibir mensajes** privados dentro de la plataforma.

Petición API

POST /api/messages

Ejemplo de datos a enviar:

```
{
  "receiver_id": 3,
  "content": "Hola, ¿cómo estás?"
}
```

Código del Componente: *src/components/MessageList.tsx:*

```tsx
// src/components/MessageList.tsx
import React, { useEffect, useState } from "react";

interface Message {
    id: number;
    sender_id: number;
    receiver_id: number;
    content: string;
    is_read: boolean;
    created_at: string;
}

export default function MessageList() {
    const [messages, setMessages] = useState<Message[]>([]);
    const [receiverId, setReceiverId] = useState("");
    const [content, setContent] = useState("");
    const [message, setMessage] = useState("");

    useEffect(() => {
        fetch("http://127.0.0.1:8000/api/messages", {
```

```
            headers: {
                Authorization: "Bearer " + localStorage.getItem("token"),
            },
        })
            .then((res) => res.json())
            .then((data) => setMessages(data))
            .catch(() => setMessage("Error al obtener mensajes"));
    }, []);



    const handleSendMessage = (e: React.FormEvent) => {
        e.preventDefault();
        fetch("http://127.0.0.1:8000/api/messages", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
                Authorization: "Bearer " + localStorage.getItem("token"),
            },
            body: JSON.stringify({
                receiver_id: parseInt(receiverId),
                content,
            }),
        })
            .then((res) => res.json())
            .then(() => setMessage("Mensaje enviado con éxito"))
            .catch(() => setMessage("Error al enviar mensaje"));
    };
    return (
        <div className="mt-4">
            <h2 className="text-2xl font-bold mb-2">Mensajería</h2>
            {message && <p className="text-green-500">{message}</p>}
      <form onSubmit={handleSendMessage} className="mb-4 flex space-x-2">
                <input
                    type="number"
                    placeholder="ID Destinatario"
                    className="border p-2 rounded"
                    value={receiverId}
                    onChange={(e) => setReceiverId(e.target.value)}
                />
                <input
                    type="text"
                    placeholder="Mensaje"
                    className="border p-2 rounded"
                    value={content}
                    onChange={(e) => setContent(e.target.value)}
                />
```

```
                <button type="submit" className="bg-green-600 text-white
p-2 rounded">
                    Enviar
                </button>
            </form>
            <ul className="space-y-2">
                {messages.map((msg) => (
                    <li key={msg.id} className="p-2 border rounded bg-
white shadow">
                        <strong>{msg.sender_id === 2 ? "Yo" : `Usuario
${msg.sender_id}`}</strong>:{" "}
                        {msg.content} - {new
Date(msg.created_at).toLocaleDateString()}
                    </li>
                ))}
            </ul>
        </div>
    );
}
```

3. Integración en App.tsx

Agregamos estos componentes a App.tsx:

```
// src/App.tsx
import { useState } from "react";
import Login from "./components/Login";
import Register from "./components/Register";
import CoursesList from "./components/CoursesList";
import SubjectsList from "./components/SubjectsList";
import AssignmentsList from "./components/AssignmentsList";
import SubmissionsList from "./components/SubmissionsList";
import MessageList from "./components/MessageList";
import LogoutButton from "./components/LogoutButton";

function App() {
  const [token, setToken] = useState<string |
null>(localStorage.getItem("token"));

  const handleLoginSuccess = (receivedToken: string) => {
    localStorage.setItem("token", receivedToken);
    setToken(receivedToken);
  };

  // Define la función handleLogout
  const handleLogout = () => {
    const token = localStorage.getItem("token");
    if (!token) return;
```

```
    fetch("http://127.0.0.1:8000/api/logout", {
      method: "POST",
      headers: {
        Authorization: "Bearer " + token,
      },
    })
      .then(() => {
        localStorage.removeItem("token");
        setToken(null);
      })
      .catch((err) => console.error(err));
  };

  return (
    <div className="min-h-screen bg-gray-100 p-4">
      {!token ? (
        <>
          <Login onLoginSuccess={handleLoginSuccess} />
          <Register />
        </>
      ) : (
        <div>
          <p className="text-xl">Bienvenido a Laredu</p>
          <LogoutButton onLogout={handleLogout} />

          {/* Módulos del sistema */}
          <CoursesList />
          <SubjectsList />
          <AssignmentsList />
          <SubmissionsList />
          <MessageList />
        </div>
      )}
    </div>
  );
}

export default App;
```

Implementación de Navegación con React Router en Laredu

Ahora vamos a estructurar la aplicación **Laredu** utilizando **React Router** para gestionar la navegación entre las distintas secciones:

- **Login y Registro**

- **Lista de Cursos**
- **Lista de Asignaturas**
- **Lista de Tareas**
- **Entrega de Tareas**
- **Mensajería**
- **Logout**

Beneficios de usar React Router:

✓ Organización clara de la app.
✓ URLs amigables para cada módulo.
✓ Navegación fluida sin recargar la página.

1 Instalar React Router

Ejecuta el siguiente comando en la carpeta del frontend:

```
npm install react-router-dom
```

2 Configurar las Rutas en App.tsx

Ahora estructuramos el archivo **App.tsx** para manejar rutas y proteger contenido.

```
// src/App.tsx

import { useState } from "react";
import { BrowserRouter as Router, Routes, Route, Navigate, Link } from
"react-router-dom";
import Login from "./components/Login";
import Register from "./components/Register";
import CoursesList from "./components/CoursesList";
import SubjectsList from "./components/SubjectsList";
import AssignmentsList from "./components/AssignmentsList";
import SubmissionsList from "./components/SubmissionsList";
import MessageList from "./components/MessageList";
import LogoutButton from "./components/LogoutButton";

export default function App() {
  const [token, setToken] = useState<string |
null>(localStorage.getItem("token"));

  const handleLoginSuccess = (receivedToken: string) => {
    localStorage.setItem("token", receivedToken);
    setToken(receivedToken);
  };

  const handleLogout = () => {
```

```
    const token = localStorage.getItem("token");
    if (!token) return;

    fetch("http://127.0.0.1:8000/api/logout", {
      method: "POST",
      headers: { Authorization: "Bearer " + token },
    })
      .then(() => {
        localStorage.removeItem("token");
        setToken(null);
      })
      .catch((err) => console.error(err));
  };

  return (
    <Router>
      <div className="min-h-screen bg-gray-100 p-4">
        {!token ? (
          <Routes>
            <Route path="/" element={<Login
onLoginSuccess={handleLoginSuccess} />} />
            <Route path="/register" element={<Register />} />
            <Route path="*" element={<Navigate to="/" />} />
          </Routes>
        ) : (
          <>
            {/* Navbar de navegación */}
            <nav className="bg-blue-600 text-white p-4 flex justify-
between">
              <div className="flex space-x-4">
                <Link to="/courses">Cursos</Link>
                <Link to="/subjects">Asignaturas</Link>
                <Link to="/assignments">Tareas</Link>
                <Link to="/submissions">Entregas</Link>
                <Link to="/messages">Mensajes</Link>
              </div>
              <LogoutButton onLogout={handleLogout} />
            </nav>

            <div className="p-4">
              <Routes>
                <Route path="/courses" element={<CoursesList />} />
                <Route path="/subjects" element={<SubjectsList />} />
                <Route path="/assignments" element={<AssignmentsList />}
/>
                <Route path="/submissions" element={<SubmissionsList />}
/>
                <Route path="/messages" element={<MessageList />} />
```

```
                <Route path="*" element={<Navigate to="/courses" />} />
              </Routes>
            </div>
          </>
        )}
      </div>
    </Router>
  );
}
```

3 Explicación de App.tsx con React Router

✓ Si el usuario NO está autenticado:

- Solo puede ver / (Login) y /register (Registro).
- Cualquier otra URL lo redirige a /.

✓ Si el usuario SÍ está autenticado:

- Aparece un **Navbar** con enlaces a cada sección.
- Puede navegar entre /courses, /subjects, /assignments, etc.
- Cualquier URL no válida lo redirige a /courses.

✓ Logout:

- Al cerrar sesión, se borra el token y se redirige al Login.

4 Mejorar la Barra de Navegación (Navbar.tsx)

Podemos extraer la barra de navegación en un componente **Navbar.tsx** para mejorar la estructura.

```
// src/components/Navbar.tsx

import { Link } from "react-router-dom";
import LogoutButton from "./LogoutButton";

interface NavbarProps {
    onLogout: () => void;
}

export default function Navbar({ onLogout }: NavbarProps) {
```

```
    return (
        <nav className="bg-blue-600 text-white p-4 flex justify-between">
            <div className="flex space-x-4">
                <Link to="/courses">Cursos</Link>
                <Link to="/subjects">Asignaturas</Link>
                <Link to="/assignments">Tareas</Link>
                <Link to="/submissions">Entregas</Link>
                <Link to="/messages">Mensajes</Link>
            </div>
            <LogoutButton onLogout={onLogout} />
        </nav>
    );
}
```

Ahora **lo importamos en App.tsx** y lo usamos:

```
import Navbar from "./components/Navbar";

// ...

{token && <Navbar onLogout={handleLogout} />}
```

5 Prueba la Navegación

1 Inicia la app con: ***npm run dev***

2 Regístrate o inicia sesión.
3 ***Navega entre cursos, asignaturas, tareas, entregas y mensajes****.*
4 ***Haz logout*** *y revisa que redirige al login correctamente.*

Mejorando el Diseño con TailwindCSS en Laredu

Ahora que la estructura de navegación está funcionando con **React Router**, vamos a mejorar la **interfaz de usuario** utilizando **TailwindCSS**.

📌 **Objetivos:**
✓ Hacer que la UI sea más atractiva y moderna.
✓ Agregar una **página de inicio/dashboard**.
✓ Mejorar la disposición de los componentes.

Mejorar la Barra de Navegación (Navbar.tsx)

📌 **Antes**: La barra de navegación era simple.
📌 **Ahora**: Agregamos **estilos mejorados, un logo y un diseño más intuitivo**.

```
// src/components/Navbar.tsx
import { Link } from "react-router-dom";
import LogoutButton from "./LogoutButton";

interface NavbarProps {
    onLogout: () => void;
}

export default function Navbar({ onLogout }: NavbarProps) {
    return (
        <nav className="bg-blue-600 text-white p-4 flex justify-between
items-center shadow-lg">
            <div className="flex items-center space-x-4">
                <span className="text-2xl font-bold">📚 Laredu</span>
                <Link className="hover:text-gray-300"
to="/courses">Cursos</Link>
                <Link className="hover:text-gray-300"
to="/subjects">Asignaturas</Link>
                <Link className="hover:text-gray-300"
to="/assignments">Tareas</Link>
                <Link className="hover:text-gray-300"
to="/submissions">Entregas</Link>
                <Link className="hover:text-gray-300"
to="/messages">Mensajes</Link>
            </div>
            <LogoutButton onLogout={onLogout} />
        </nav>
    );
}
```

✓ Ahora la barra de navegación es más intuitiva y atractiva.
✓ Se ha agregado un título y separación entre enlaces.

Creación del Dashboard (Dashboard.tsx)

Objetivo: Mostrar un resumen de las secciones principales.

```
// src/components/Dashboard.tsx
import { Link } from "react-router-dom";

export default function Dashboard() {
    return (
        <div className="p-6">
            <h1 className="text-3xl font-bold mb-6">Bienvenido a
Laredu</h1>
```

```
            <div className="grid grid-cols-2 gap-4">
                <Link to="/courses" className="p-4 bg-blue-500 text-white
rounded shadow hover:bg-blue-400">
                    📖 Ver Cursos
                </Link>
                <Link to="/subjects" className="p-4 bg-green-500 text-
white rounded shadow hover:bg-green-400">
                    📚 Ver Asignaturas
                </Link>
                <Link to="/assignments" className="p-4 bg-purple-500
text-white rounded shadow hover:bg-purple-400">
                    📝 Ver Tareas
                </Link>
                <Link to="/submissions" className="p-4 bg-yellow-500
text-white rounded shadow hover:bg-yellow-400">
                    📤 Ver Entregas
                </Link>
                <Link to="/messages" className="p-4 bg-red-500 text-white
rounded shadow hover:bg-red-400">
                    💬 Ver Mensajes
                </Link>
            </div>
        </div>
    );
}
```

✓ **Página de inicio más organizada con botones llamativos.**
✓ **Cada botón dirige a una sección del sistema.**

Integrar el Dashboard en App.tsx

Modificamos App.tsx para que el Dashboard sea la página principal tras iniciar sesión.

```
// src/App.tsx

import { useState } from "react";
import { BrowserRouter as Router, Routes, Route, Navigate } from "react-
router-dom";
import Login from "./components/Login";
import Register from "./components/Register";
import Navbar from "./components/Navbar";
import Dashboard from "./components/Dashboard";
import CoursesList from "./components/CoursesList";
import SubjectsList from "./components/SubjectsList";
import AssignmentsList from "./components/AssignmentsList";
import SubmissionsList from "./components/SubmissionsList";
```

```
import MessageList from "./components/MessageList";

export default function App() {
  const [token, setToken] = useState<string |
null>(localStorage.getItem("token"));
  const handleLoginSuccess = (receivedToken: string) => {
    localStorage.setItem("token", receivedToken);
    setToken(receivedToken);
  };
  return (
    <Router>
      <div className="min-h-screen bg-gray-100">
        {!token ? (
          <Routes>
            <Route path="/" element={<Login
onLoginSuccess={handleLoginSuccess} />} />
            <Route path="/register" element={<Register />} />
            <Route path="*" element={<Navigate to="/" />} />
          </Routes>
        ) : (
          <>
            <Navbar onLogout={() => { localStorage.removeItem("token");
setToken(null); }} />
            <div className="p-6">
              <Routes>
                <Route path="/" element={<Dashboard />} />
                <Route path="/courses" element={<CoursesList />} />
                <Route path="/subjects" element={<SubjectsList />} />
                <Route path="/assignments" element={<AssignmentsList />}
/>
                <Route path="/submissions" element={<SubmissionsList />}
/>
                <Route path="/messages" element={<MessageList />} />
                <Route path="*" element={<Navigate to="/" />} />
              </Routes>
            </div>
          </>
        )}
      </div>
    </Router>
  );
}
```

✓ **El Dashboard es la nueva página principal tras iniciar sesión.**
✓ **El usuario puede navegar fácilmente entre secciones.**

Estilizar Botón de Logout (LogoutButton.tsx)

Para que el botón de Logout se vea mejor, lo estilizamos con TailwindCSS.

```tsx
// src/components/LogoutButton.tsx
interface LogoutProps {
    onLogout: () => void;
}
export default function LogoutButton({ onLogout }: LogoutProps) {
    return (
        <button
            onClick={onLogout}
      className="mt-4 bg-red-600 text-white p-2 rounded hover:bg-red-500"
        >
            Cerrar Sesión
        </button>
    );
}
```

✓ **Mejor diseño y efecto visual al pasar el mouse.**

Protección de Rutas en Laredu con React Router

*Ahora aseguraremos que solo **usuarios autenticados** puedan acceder a las rutas protegidas de la aplicación, evitando que alguien sin sesión iniciada acceda a cursos, tareas, mensajes, etc.*

Objetivos:

✓***Bloquear acceso no autorizado** a rutas internas.*
✓***Redirigir al login si el usuario no tiene un token válido.***
✓***Mantener la sesión incluso al recargar la página.***

Crear un ProtectedRoute (Ruta Protegida)

Este componente verificará si el usuario está autenticado antes de mostrar la página.

```tsx
// src/components/ProtectedRoute.tsx
import { Navigate, Outlet } from "react-router-dom";

export default function ProtectedRoute() {
    const token = localStorage.getItem("token");

    return token ? <Outlet /> : <Navigate to="/" replace />;
}
```

✓***Si hay un token, muestra la página.***
✓***Si no hay un token, redirige al login (/).***
✓***Utiliza <Outlet /> para renderizar los componentes protegidos.***

Aplicar ProtectedRoute en App.tsx

Modificamos App.tsx para envolver las rutas protegidas dentro de ProtectedRoute.

```
import { BrowserRouter as Router, Routes, Route, Navigate } from "react-
router-dom";
import { useState, useEffect } from "react";
import Login from "./components/Login";
import Register from "./components/Register";
import Navbar from "./components/Navbar";
import Dashboard from "./components/Dashboard";
import CoursesList from "./components/CoursesList";
import SubjectsList from "./components/SubjectsList";
import AssignmentsList from "./components/AssignmentsList";
import SubmissionsList from "./components/SubmissionsList";
import MessageList from "./components/MessageList";

export default function App() {
  const [token, setToken] = useState<string |
null>(localStorage.getItem("token"));
  useEffect(() => {
    setToken(localStorage.getItem("token")); // Asegura que el estado
refleje cambios en localStorage
  }, []);
  const handleLoginSuccess = (receivedToken: string) => {
    localStorage.setItem("token", receivedToken);
    setToken(receivedToken);
  };
  const handleLogout = () => {
    localStorage.removeItem("token");
    setToken(null);
  };
  return (
    <Router>
      <div className="min-h-screen bg-gray-100">
        {!token ? (
          <Routes>
            <Route path="/" element={<Login
onLoginSuccess={handleLoginSuccess} />} />
            <Route path="/register" element={<Register />} />
            <Route path="*" element={<Navigate to="/" />} />
          </Routes>
        ) : (
          <>
            <Navbar onLogout={handleLogout} />
            <div className="p-6">
              <Routes>
```

```
                <Route path="/" element={<Dashboard />} />
                <Route path="/courses" element={<CoursesList />} />
                <Route path="/subjects" element={<SubjectsList />} />
                <Route path="/assignments" element={<AssignmentsList />}
/>
                <Route path="/submissions" element={<SubmissionsList />}
/>
                <Route path="/messages" element={<MessageList />} />
                <Route path="*" element={<Navigate to="/" />} />
              </Routes>
            </div>
          </>
        )}
      </div>
    </Router>
  );
}
```

✓El usuario NO autenticado solo puede ver / (Login) y /register (Registro).
✓Si intenta acceder a /courses, /subjects, etc., será redirigido al Login.
✓Si está autenticado, verá la barra de navegación y el dashboard.

Pruebas de Seguridad en la Navegación

Verificar que la protección funciona correctamente *con los siguientes casos de prueba.*

Caso 1: Usuario NO autenticado intenta acceder a una ruta protegida

✅ ***Pasos:***

1. *Abre una pestaña en modo incógnito.*
2. *Ve a http://localhost:5173/courses (o cualquier otra ruta protegida).*
3. ***Resultado esperado:*** *Redirección automática a http://localhost:5173/.*

Caso 2: Usuario inicia sesión y accede a rutas protegidas

✅ ***Pasos:***

1. *Ingresa credenciales en la página de login.*
2. ***Resultado esperado:*** *Aparece el* ***dashboard*** *con todas las opciones del sistema.*

Caso 3: Usuario cierra sesión y prueba acceder a rutas protegidas

✅ ***Pasos:***

1. *Haz clic en el botón de* ***Cerrar Sesión****.*
2. *Intenta ingresar a http://localhost:5173/courses.*
3. ***Resultado esperado:*** *Redirección automática al Login.*

*El error **"token is assigned a value but never used"** ocurre porque **token** está declarado en el estado, pero no se está utilizando en el código.*
*Esto sucede en **App.tsx**, donde hemos definido:*

```
const [token, setToken] = useState<string |
null>(localStorage.getItem("token"));
```

*Pero luego **nunca usamos token directamente** dentro del return.*

✅ *Solución 1: Usar token en la Lógica de Navegación*

*En lugar de depender de ProtectedRoute, podemos usar **token directamente** en App.tsx para determinar qué mostrar.*

Código actualizado en App.tsx

```
// eslint-disable-next-line @typescript-eslint/no-unused-vars
const [token, setToken] = useState<string |
null>(localStorage.getItem("token"));
```

✓ Esto soluciona el error, pero no es la mejor práctica.
✓ Recomiendo la primera solución para una mejor gestión de estado.