Para desarrollar un Sistema de Registro y Control de Encomiendas en el Área de Almacén utilizando Visual Studio y React, seguirás un enfoque basado en la arquitectura moderna de aplicaciones web. A continuación, te proporcionaré una guía paso a paso para construir este sistema.

1. Configuración del Entorno de Desarrollo

- **Visual Studio**: Utiliza Visual Studio como tu IDE principal para gestionar tanto el backend como el frontend del proyecto.
- **Node.js**: Asegúrate de tener instalado Node.js, ya que React depende de él.
- **Create React App**: Utiliza Create React App para configurar rápidamente el entorno React.
- **.NET Core**: Utiliza .NET Core para desarrollar el backend del sistema, que se encargará de manejar la lógica de negocio y la conexión a la base de datos.

2. Diseño de la Arquitectura

- **Frontend (React)**: El frontend será una aplicación SPA (Single Page Application) desarrollada con React. Se encargará de la interacción del usuario, el registro de encomiendas, la visualización del estado de las encomiendas y otras funcionalidades necesarias.
- **Backend (.NET Core)**: El backend se encargará de gestionar las operaciones CRUD (Create, Read, Update, Delete) en la base de datos. Utiliza Web API de .NET Core para exponer endpoints que puedan ser consumidos por la aplicación React.
- **Base de Datos**: Puedes optar por SQL Server o cualquier otra base de datos compatible con .NET Core para almacenar la información de las encomiendas.

3. Implementación del Backend

1. **Crear un nuevo proyecto en Visual Studio**:
    - Abre Visual Studio y crea un nuevo proyecto ASP.NET Core Web API.
    - Configura la conexión a la base de datos (SQL Server, por ejemplo).
2. **Modelos y Controladores**:
    - Define los modelos que representarán las entidades del sistema, como `Encomienda`, `Cliente`, `Estado`, `Almacén`, etc.
    - Crea controladores para manejar las operaciones de la API (GET, POST, PUT, DELETE). Estos controladores interactuarán con la base de datos para realizar las operaciones necesarias.
3. **Servicios y Repositorios**:
    - Implementa servicios que manejen la lógica de negocio.
    - Utiliza el patrón de repositorio para interactuar con la base de datos.
4. **Endpoints API**:
    - Define endpoints para las operaciones que el frontend necesitará, como el registro de nuevas encomiendas, la actualización del estado, la consulta de encomiendas por ID, etc.

4. Implementación del Frontend

1. **Crear un proyecto React**:
    - Utiliza Create React App para iniciar un nuevo proyecto de React.
    - Configura las rutas y la estructura de carpetas de tu aplicación.
2. **Componentes de React**:
    - Crea componentes reutilizables como formularios de registro, tablas de visualización de encomiendas, y botones para actualizar o eliminar registros.
    - Implementa componentes de estado utilizando `useState` y `useEffect` para manejar la lógica interna y las llamadas a la API.
3. **Integración con el Backend**:
    - Utiliza `fetch` o Axios para hacer solicitudes HTTP al backend .NET Core.
    - Implementa autenticación y manejo de errores para garantizar que solo usuarios autorizados puedan acceder a ciertas funcionalidades.
4. **Diseño de la Interfaz**:
    - Usa CSS o frameworks como Bootstrap para estilizar la aplicación.
    - Asegúrate de que la interfaz sea intuitiva y fácil de usar.

5. Pruebas y Despliegue

1. **Pruebas**:
    - Realiza pruebas unitarias y de integración para asegurarte de que todas las funcionalidades funcionan correctamente.
    - Prueba el sistema en diferentes navegadores y dispositivos para garantizar la compatibilidad.
2. **Despliegue**:
    - Puedes desplegar el backend en un servidor web compatible con .NET Core (como Azure o AWS).
    - El frontend puede ser desplegado en un servicio de hosting estático o en el mismo servidor donde resides el backend.

6. Optimización y Mantenimiento

- **Monitoreo**: Implementa herramientas de monitoreo para rastrear el rendimiento y los posibles errores en tiempo real.
- **Actualizaciones**: Asegúrate de mantener el sistema actualizado con las últimas versiones de React y .NET Core para beneficiarte de mejoras de seguridad y rendimiento.

Desarrollo

1. Backend: .NET Core API

a. Crear un Proyecto en Visual Studio

1. Abre Visual Studio y selecciona **Create a new project**.
2. Selecciona **ASP.NET Core Web API** y da un nombre al proyecto.
3. Configura el proyecto para utilizar **.NET 6 (o la versión más reciente disponible)**.

b. Definir Modelos

Crea un modelo para `Encomienda` en el directorio `Models`:

csharp
Copiar código

```
using System;

namespace TromeExpressCargoAPI.Models
{
    public class Encomienda
    {
        public int Id { get; set; }
        public string Codigo { get; set; }
        public string Descripcion { get; set; }
        public string Destinatario { get; set; }
        public string Estado { get; set; }
        public DateTime FechaRegistro { get; set; }
    }
}
```

c. Configurar el Contexto de Base de Datos

En el archivo `Data/TromeExpressContext.cs`:

csharp
Copiar código

```
using Microsoft.EntityFrameworkCore;
using TromeExpressCargoAPI.Models;

namespace TromeExpressCargoAPI.Data
{
    public class TromeExpressContext : DbContext
    {
```

```
        public
TromeExpressContext(DbContextOptions<TromeExpressContext> options) :
base(options) { }

        public DbSet<Encomienda> Encomiendas { get; set; }
    }
}
```

d. Crear el Controlador

En el directorio `Controllers`, crea un controlador `EncomiendasController.cs`:

csharp
Copiar código

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using TromeExpressCargoAPI.Data;
using TromeExpressCargoAPI.Models;

namespace TromeExpressCargoAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class EncomiendasController : ControllerBase
    {
        private readonly TromeExpressContext _context;

        public EncomiendasController(TromeExpressContext context)
        {
            _context = context;
        }

        // GET: api/Encomiendas
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Encomienda>>>
GetEncomiendas()
        {
            return await _context.Encomiendas.ToListAsync();
        }
```

```
        // GET: api/Encomiendas/5
        [HttpGet("{id}")]
        public async Task<ActionResult<Encomienda>>
GetEncomienda(int id)
        {
            var encomienda = await
_context.Encomiendas.FindAsync(id);

            if (encomienda == null)
            {
                return NotFound();
            }

            return encomienda;
        }

        // POST: api/Encomiendas
        [HttpPost]
        public async Task<ActionResult<Encomienda>>
PostEncomienda(Encomienda encomienda)
        {
            _context.Encomiendas.Add(encomienda);
            await _context.SaveChangesAsync();

            return CreatedAtAction(nameof(GetEncomienda), new { id =
encomienda.Id }, encomienda);
        }

        // PUT: api/Encomiendas/5
        [HttpPut("{id}")]
        public async Task<IActionResult> PutEncomienda(int id,
Encomienda encomienda)
        {
            if (id != encomienda.Id)
            {
                return BadRequest();
            }

            _context.Entry(encomienda).State = EntityState.Modified;
```

```
            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!EncomiendaExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // DELETE: api/Encomiendas/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteEncomienda(int id)
        {
            var encomienda = await
_context.Encomiendas.FindAsync(id);
            if (encomienda == null)
            {
                return NotFound();
            }

            _context.Encomiendas.Remove(encomienda);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool EncomiendaExists(int id)
        {
            return _context.Encomiendas.Any(e => e.Id == id);
        }
    }
```

```
}
```

e. Configurar el Startup

Configura la conexión a la base de datos en `appsettings.json` y `Program.cs`.

`appsettings.json`:

json
Copiar código

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=TromeExpressCargoDB;Trusted
_Connection=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

`Program.cs`:

csharp
Copiar código

```
using Microsoft.EntityFrameworkCore;
using TromeExpressCargoAPI.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddDbContext<TromeExpressContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("Defa
ultConnection")));

builder.Services.AddControllers();
```

```
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}

app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

2. Frontend: React

a. Crear Proyecto React

En la línea de comandos:

bash
Copiar código
```
npx create-react-app trome-express-cargo
cd trome-express-cargo
```

b. Configurar Axios para las Llamadas API

Instala Axios:

bash
Copiar código
```
npm install axios
```

Crea un archivo `src/services/EncomiendaService.js` para manejar las llamadas API:

javascript
Copiar código
```
import axios from 'axios';

const API_URL = "https://localhost:5001/api/encomiendas";

export const getEncomiendas = async () => {
    return await axios.get(API_URL);
```

```
};

export const getEncomiendaById = async (id) => {
    return await axios.get(`${API_URL}/${id}`);
};

export const createEncomienda = async (data) => {
    return await axios.post(API_URL, data);
};

export const updateEncomienda = async (id, data) => {
    return await axios.put(`${API_URL}/${id}`, data);
};

export const deleteEncomienda = async (id) => {
    return await axios.delete(`${API_URL}/${id}`);
};
```

c. Crear Componentes React

Crea un componente simple para listar encomiendas `src/components/EncomiendaList.js`:

javascript
Copiar código

```
import React, { useState, useEffect } from 'react';
import { getEncomiendas } from '../services/EncomiendaService';

const EncomiendaList = () => {
    const [encomiendas, setEncomiendas] = useState([]);

    useEffect(() => {
        const fetchEncomiendas = async () => {
            const response = await getEncomiendas();
            setEncomiendas(response.data);
        };
        fetchEncomiendas();
    }, []);

    return (
        <div>
            <h2>Lista de Encomiendas</h2>
```

```
            <ul>
                {encomiendas.map((encomienda) => (
                    <li key={encomienda.id}>
                        {encomienda.codigo} -
{encomienda.destinatario} - {encomienda.estado}
                    </li>
                ))}
            </ul>
        </div>
    );
};

export default EncomiendaList;
```

d. Renderizar en `App.js`

javascript
Copiar código

```
import React from 'react';
import './App.css';
import EncomiendaList from './components/EncomiendaList';

function App() {
    return (
        <div className="App">
            <header className="App-header">
                <h1>Trome Express Cargo</h1>
                <EncomiendaList />
            </header>
        </div>
    );
}

export default App;
```

3. Ejecutar y Probar

1. **Backend**: Inicia el backend desde Visual Studio.

Frontend: En la línea de comandos, navega a la carpeta del proyecto React y ejecuta:
bash

Copiar código

```
npm start
```

2.

El frontend se ejecutará en `http://localhost:3000`, mientras que la API estará en `https://localhost:5001`.