I want to create a. Full stack app Using asp.net core MVC ORM: Dapper Database Server: ms sql APi: Bruno swagger Operation: Crud Language: C# Authentication : Identity Database : PES Table: AMl, Trustee,SP Ui: material Froentend - React Redux Design Pattern: Singleton Services: cors And we will create tables in ms sql PES database Manually No migration We will do step by step Suggest me the best project idea for this

###### **ChatGPT said:**

Got it! Since you want to build a **full-stack app** with ASP.NET Core MVC (Dapper + Identity) + React/Redux + Material UI and MS SQL (manual tables), here’s a **project idea that fits your stack and allows CRUD + Identity authentication**:

### **Project Idea: PES Portal - Compliance & Trustee Management System**

**Description:** Build a portal for the PES (Plan Enrollment System) database to manage **AML (Anti-Money Laundering) records, Trustees, and Service Providers (SP)** with proper user authentication and role-based access.

The system will allow admins and users to:

1. **Manage AML Data** (upload, view, edit, delete records)
2. **Manage Trustees** (details, plans assigned, compliance)
3. **Manage Service Providers** (basic details, services linked to plans)
4. **Generate Compliance Reports** (downloadable PDF/Excel)
5. **Role-Based Access** using Identity (Admin, User, Auditor)

### **Modules & Tables**

1. **Authentication (Identity)** – User login, role-based access
2. **AML Module**
   * Table: AML (store compliance records per plan)
   * CRUD: Add, update, soft delete AML records
3. **Trustee Module**
   * Table: Trustee (Trustee details, contact info, plans mapped)
   * CRUD: Manage trustees and link AML records
4. **Service Provider Module (SP)**
   * Table: SP (basic details, associated trustees/plans)
   * CRUD: Manage SP info and assign to trustees
5. **Reports & Dashboard**
   * Fetch aggregated data using **Dapper queries**
   * Show graphs/charts using Material UI (React)

### **Features**

1. **User Roles:** Admin (full CRUD), User (view + limited edit), Auditor (view only)
2. **Identity Authentication:** Secure login with password hashing, JWT tokens for APIs
3. **React + Material UI:**
   * Table view with pagination, filtering (DataGrid)
   * Forms with validation
   * Modal dialogs for CRUD
4. **Dapper for Data Access:** Use stored procedures & queries (with NOLOCK)
5. **Singleton Pattern:** Database context/service layer as Singleton
6. **CORS enabled APIs:** Connect React frontend to ASP.NET backend
7. **Bruno + Swagger:** API documentation & testing

### **Why this project is best?**

* Uses all your mentioned tables **AML, Trustee, SP** logically
* Covers full CRUD with role-based authentication
* Realistic (fits PES domain) and extendable
* React UI + Material gives a professional look
* You can showcase **Dapper performance + Singleton pattern**

👉 Do you want me to **start with database table structure (AML, Trustee, SP) in MS SQL (manual creation)**?  
 Or first **set up the ASP.NET Core MVC backend project with Dapper, Identity & Swagger**?

##### **You said:**

But tablename must be Deepanshu\_tablename And each with nolock

###### **ChatGPT said:**

Perfect! We'll stick to your naming convention and **always use WITH (NOLOCK)** in queries.

Here’s how we’ll proceed:

## **Step 1: Database Design (Manual Tables)**

We’ll create **3 tables** in the PES database with your convention:

### **1. AML Table**

sql

CopyEdit

CREATE TABLE Deepanshu\_AML (

AML\_ID INT IDENTITY(1,1) PRIMARY KEY,

Plan\_Number VARCHAR(20) NOT NULL,

Status VARCHAR(50) NOT NULL,

Compliance\_Date DATETIME NOT NULL,

Comments VARCHAR(255) NULL,

Created\_On DATETIME DEFAULT GETDATE()

);

### **2. Trustee Table**

sql

CopyEdit

CREATE TABLE Deepanshu\_Trustee (

Trustee\_ID INT IDENTITY(1,1) PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Email VARCHAR(100) NOT NULL,

Phone VARCHAR(20) NULL,

Plan\_Number VARCHAR(20) NOT NULL,

Created\_On DATETIME DEFAULT GETDATE()

);

### **3. Service Provider (SP) Table**

sql

CopyEdit

CREATE TABLE Deepanshu\_SP (

SP\_ID INT IDENTITY(1,1) PRIMARY KEY,

Name VARCHAR(100) NOT NULL,

Service\_Type VARCHAR(50) NOT NULL,

Trustee\_ID INT NOT NULL,

Created\_On DATETIME DEFAULT GETDATE(),

FOREIGN KEY (Trustee\_ID) REFERENCES Deepanshu\_Trustee(Trustee\_ID)

);

## **Step 2: Query Rule**

Whenever we fetch data, we’ll always use **WITH (NOLOCK)**:

sql

CopyEdit

SELECT \* FROM Deepanshu\_AML WITH (NOLOCK);

SELECT \* FROM Deepanshu\_Trustee WITH (NOLOCK);

SELECT \* FROM Deepanshu\_SP WITH (NOLOCK);

## **Next Steps**

1️⃣ **Do you want me to create full insert/select/update/delete queries (with NOLOCK)** for all tables?  
 2️⃣ Or should we now move to **Step 2: Backend Setup** (ASP.NET Core MVC + Dapper + Identity + Swagger + Singleton)?

👉 **Which one should we do first?**

* **Complete SQL scripts (CRUD with NOLOCK)**
* **Setup the ASP.NET Core project (Dapper + Identity)**

##### **You said:**

Both

###### **ChatGPT said:**

Okay! Let's do **both: SQL CRUD (with NOLOCK)** and **Backend setup** step-by-step.

## **PART 1: SQL CRUD QUERIES (with NOLOCK)**

### **1. AML Table (Deepanshu\_AML)**

#### **Insert**

sql

CopyEdit

INSERT INTO Deepanshu\_AML (Plan\_Number, Status, Compliance\_Date, Comments)

VALUES ('PLAN123', 'Approved', GETDATE(), 'AML verified');

#### **Select (NOLOCK)**

sql

CopyEdit

SELECT \*

FROM Deepanshu\_AML WITH (NOLOCK);

#### **Update**

sql

CopyEdit

UPDATE Deepanshu\_AML

SET Status = 'Pending', Comments = 'Verification required'

WHERE AML\_ID = 1;

#### **Delete**

sql

CopyEdit

DELETE FROM Deepanshu\_AML

WHERE AML\_ID = 1;

### **2. Trustee Table (Deepanshu\_Trustee)**

#### **Insert**

sql

CopyEdit

INSERT INTO Deepanshu\_Trustee (Name, Email, Phone, Plan\_Number)

VALUES ('John Doe', 'john@example.com', '9876543210', 'PLAN123');

#### **Select (NOLOCK)**

sql

CopyEdit

SELECT \*

FROM Deepanshu\_Trustee WITH (NOLOCK);

#### **Update**

sql

CopyEdit

UPDATE Deepanshu\_Trustee

SET Email = 'john.doe@trustee.com'

WHERE Trustee\_ID = 1;

#### **Delete**

sql

CopyEdit

DELETE FROM Deepanshu\_Trustee

WHERE Trustee\_ID = 1;

### **3. SP Table (Deepanshu\_SP)**

#### **Insert**

sql

CopyEdit

INSERT INTO Deepanshu\_SP (Name, Service\_Type, Trustee\_ID)

VALUES ('ABC Services', 'Auditing', 1);

#### **Select (NOLOCK)**

sql

CopyEdit

SELECT \*

FROM Deepanshu\_SP WITH (NOLOCK);

#### **Update**

sql

CopyEdit

UPDATE Deepanshu\_SP

SET Service\_Type = 'Advisory'

WHERE SP\_ID = 1;

#### **Delete**

sql

CopyEdit

DELETE FROM Deepanshu\_SP

WHERE SP\_ID = 1;

## **PART 2: BACKEND SETUP (ASP.NET Core MVC + Dapper + Identity)**

### **Step 1: Project Structure**

1. Create a **new ASP.NET Core Web App (Model-View-Controller)** project
2. Add folders:  
   * Controllers
   * Models
   * Data (for Dapper Context)
   * Services

Install NuGet packages:  
  
 bash  
CopyEdit  
dotnet add package Dapper

dotnet add package Microsoft.Data.SqlClient

dotnet add package Swashbuckle.AspNetCore

dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore

### **Step 2: Dapper Singleton Context**

Data/DapperContext.cs

csharp

CopyEdit

using Microsoft.Data.SqlClient;

using System.Data;

public class DapperContext

{

private readonly string \_connectionString;

private IDbConnection \_connection;

public DapperContext(IConfiguration configuration)

{

\_connectionString = configuration.GetConnectionString("PESDb");

}

public IDbConnection CreateConnection()

{

if (\_connection == null)

{

\_connection = new SqlConnection(\_connectionString);

}

return \_connection;

}

}

Register as **Singleton** in Program.cs:

csharp

CopyEdit

builder.Services.AddSingleton<DapperContext>();

### **Step 3: Identity Authentication**

Add **Identity** in Program.cs  
  
 csharp  
CopyEdit  
builder.Services.AddIdentity<IdentityUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

1. Add login/register endpoints (will show in later steps).

### **Step 4: AML Controller with Dapper**

Controllers/AMLController.cs

csharp

CopyEdit

using Dapper;

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/[controller]")]

public class AMLController : ControllerBase

{

private readonly DapperContext \_context;

public AMLController(DapperContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var sql = "SELECT \* FROM Deepanshu\_AML WITH (NOLOCK)";

using var conn = \_context.CreateConnection();

var data = await conn.QueryAsync(sql);

return Ok(data);

}

}

### **Step 5: Swagger + CORS**

In Program.cs

csharp

CopyEdit

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowAll", builder =>

builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

});

var app = builder.Build();

app.UseCors("AllowAll");

app.UseSwagger();

app.UseSwaggerUI();

## **NEXT MOVE**

👉 Do you want me to:

1. **Complete all Controllers (AML, Trustee, SP) with Dapper CRUD + Identity authorization**?
2. Or move to **React + Redux frontend (Material UI) setup** and link APIs?

Which **step do you want to tackle first?**

##### **You said:**

Yes

###### **ChatGPT said:**

Great! We'll **complete all Controllers (AML, Trustee, SP) with full Dapper CRUD + Identity Authorization** first, then move to the React + Redux (Material UI) frontend.

## **1. AML Controller (Deepanshu\_AML)**

Controllers/AMLController.cs

csharp

CopyEdit

using Dapper;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/[controller]")]

[Authorize] // Role-based auth can be added later

public class AMLController : ControllerBase

{

private readonly DapperContext \_context;

public AMLController(DapperContext context)

{

\_context = context;

}

// GET All

[HttpGet]

public async Task<IActionResult> GetAll()

{

var sql = "SELECT \* FROM Deepanshu\_AML WITH (NOLOCK)";

using var conn = \_context.CreateConnection();

var data = await conn.QueryAsync(sql);

return Ok(data);

}

// GET by ID

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id)

{

var sql = "SELECT \* FROM Deepanshu\_AML WITH (NOLOCK) WHERE AML\_ID = @id";

using var conn = \_context.CreateConnection();

var data = await conn.QuerySingleOrDefaultAsync(sql, new { id });

if (data == null) return NotFound();

return Ok(data);

}

// POST

[HttpPost]

public async Task<IActionResult> Create([FromBody] dynamic model)

{

var sql = @"INSERT INTO Deepanshu\_AML (Plan\_Number, Status, Compliance\_Date, Comments)

VALUES (@Plan\_Number, @Status, @Compliance\_Date, @Comments)";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, model);

return rows > 0 ? Ok("Inserted") : BadRequest();

}

// PUT

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, [FromBody] dynamic model)

{

var sql = @"UPDATE Deepanshu\_AML SET Plan\_Number=@Plan\_Number, Status=@Status,

Compliance\_Date=@Compliance\_Date, Comments=@Comments WHERE AML\_ID=@id";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, new

{

model.Plan\_Number,

model.Status,

model.Compliance\_Date,

model.Comments,

id

});

return rows > 0 ? Ok("Updated") : NotFound();

}

// DELETE

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var sql = "DELETE FROM Deepanshu\_AML WHERE AML\_ID=@id";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, new { id });

return rows > 0 ? Ok("Deleted") : NotFound();

}

}

## **2. Trustee Controller (Deepanshu\_Trustee)**

Controllers/TrusteeController.cs

csharp

CopyEdit

using Dapper;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/[controller]")]

[Authorize]

public class TrusteeController : ControllerBase

{

private readonly DapperContext \_context;

public TrusteeController(DapperContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var sql = "SELECT \* FROM Deepanshu\_Trustee WITH (NOLOCK)";

using var conn = \_context.CreateConnection();

var data = await conn.QueryAsync(sql);

return Ok(data);

}

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id)

{

var sql = "SELECT \* FROM Deepanshu\_Trustee WITH (NOLOCK) WHERE Trustee\_ID=@id";

using var conn = \_context.CreateConnection();

var data = await conn.QuerySingleOrDefaultAsync(sql, new { id });

if (data == null) return NotFound();

return Ok(data);

}

[HttpPost]

public async Task<IActionResult> Create([FromBody] dynamic model)

{

var sql = @"INSERT INTO Deepanshu\_Trustee (Name, Email, Phone, Plan\_Number)

VALUES (@Name, @Email, @Phone, @Plan\_Number)";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, model);

return rows > 0 ? Ok("Inserted") : BadRequest();

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, [FromBody] dynamic model)

{

var sql = @"UPDATE Deepanshu\_Trustee SET Name=@Name, Email=@Email,

Phone=@Phone, Plan\_Number=@Plan\_Number WHERE Trustee\_ID=@id";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, new

{

model.Name,

model.Email,

model.Phone,

model.Plan\_Number,

id

});

return rows > 0 ? Ok("Updated") : NotFound();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var sql = "DELETE FROM Deepanshu\_Trustee WHERE Trustee\_ID=@id";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, new { id });

return rows > 0 ? Ok("Deleted") : NotFound();

}

}

## **3. SP Controller (Deepanshu\_SP)**

Controllers/SPController.cs

csharp

CopyEdit

using Dapper;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

[Route("api/[controller]")]

[Authorize]

public class SPController : ControllerBase

{

private readonly DapperContext \_context;

public SPController(DapperContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var sql = "SELECT \* FROM Deepanshu\_SP WITH (NOLOCK)";

using var conn = \_context.CreateConnection();

var data = await conn.QueryAsync(sql);

return Ok(data);

}

[HttpGet("{id}")]

public async Task<IActionResult> GetById(int id)

{

var sql = "SELECT \* FROM Deepanshu\_SP WITH (NOLOCK) WHERE SP\_ID=@id";

using var conn = \_context.CreateConnection();

var data = await conn.QuerySingleOrDefaultAsync(sql, new { id });

if (data == null) return NotFound();

return Ok(data);

}

[HttpPost]

public async Task<IActionResult> Create([FromBody] dynamic model)

{

var sql = @"INSERT INTO Deepanshu\_SP (Name, Service\_Type, Trustee\_ID)

VALUES (@Name, @Service\_Type, @Trustee\_ID)";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, model);

return rows > 0 ? Ok("Inserted") : BadRequest();

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(int id, [FromBody] dynamic model)

{

var sql = @"UPDATE Deepanshu\_SP SET Name=@Name, Service\_Type=@Service\_Type,

Trustee\_ID=@Trustee\_ID WHERE SP\_ID=@id";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, new

{

model.Name,

model.Service\_Type,

model.Trustee\_ID,

id

});

return rows > 0 ? Ok("Updated") : NotFound();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var sql = "DELETE FROM Deepanshu\_SP WHERE SP\_ID=@id";

using var conn = \_context.CreateConnection();

var rows = await conn.ExecuteAsync(sql, new { id });

return rows > 0 ? Ok("Deleted") : NotFound();

}

}

## **Next Step: Connect to Identity + Roles**

1. Add **Admin/User roles** so only admins can POST/PUT/DELETE
2. Protect endpoints with [Authorize(Roles="Admin")]

👉 Do you want me to:

1. **Add Identity Register/Login + Roles setup** (so we can authenticate APIs from frontend)?
2. Or move directly to **React + Redux (Material UI) frontend setup** and consume these APIs?

Which one should we do now?

##### **You said:**

Both

###### **ChatGPT said:**

Awesome! Let’s **do both: Identity (Register/Login + Roles)** **AND** React + Redux + Material UI setup **together in proper flow**.

## **PART 1: Identity Setup (Register/Login + Roles)**

We’ll integrate **ASP.NET Core Identity** into the same backend project so APIs are protected.

### **1. Create Identity DbContext**

Create a new file: Data/ApplicationDbContext.cs

csharp

CopyEdit

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : IdentityDbContext<IdentityUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

}

### **2. Configure in Program.cs**

csharp

CopyEdit

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Identity;

var builder = WebApplication.CreateBuilder(args);

// Connection for Identity

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("PESIdentityDb")));

// Add Identity

builder.Services.AddIdentity<IdentityUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

// Add Authentication & JWT

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = "JwtBearer";

options.DefaultChallengeScheme = "JwtBearer";

})

.AddJwtBearer("JwtBearer", options =>

{

options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters

{

ValidateIssuer = false,

ValidateAudience = false,

ValidateIssuerSigningKey = true,

IssuerSigningKey = new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes("SuperSecretKey@345"))

};

});

// Services

builder.Services.AddControllers();

builder.Services.AddSingleton<DapperContext>();

builder.Services.AddCors(options => options.AddPolicy("AllowAll", p => p.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader()));

builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseCors("AllowAll");

app.UseAuthentication();

app.UseAuthorization();

app.UseSwagger();

app.UseSwaggerUI();

app.MapControllers();

app.Run();

### **3. AuthController: Register & Login**

Create Controllers/AuthController.cs

csharp

CopyEdit

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using System.IdentityModel.Tokens.Jwt;

using Microsoft.IdentityModel.Tokens;

using System.Text;

using System.Security.Claims;

[ApiController]

[Route("api/[controller]")]

public class AuthController : ControllerBase

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

public AuthController(UserManager<IdentityUser> userManager, RoleManager<IdentityRole> roleManager)

{

\_userManager = userManager;

\_roleManager = roleManager;

}

[HttpPost("register")]

public async Task<IActionResult> Register([FromBody] dynamic model)

{

var user = new IdentityUser { UserName = (string)model.Username, Email = (string)model.Email };

var result = await \_userManager.CreateAsync(user, (string)model.Password);

if (result.Succeeded)

{

if (!await \_roleManager.RoleExistsAsync("Admin"))

await \_roleManager.CreateAsync(new IdentityRole("Admin"));

await \_userManager.AddToRoleAsync(user, "Admin");

return Ok("User Registered & Assigned Admin Role");

}

return BadRequest(result.Errors);

}

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] dynamic model)

{

var user = await \_userManager.FindByNameAsync((string)model.Username);

if (user != null && await \_userManager.CheckPasswordAsync(user, (string)model.Password))

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.UTF8.GetBytes("SuperSecretKey@345");

var roles = await \_userManager.GetRolesAsync(user);

var claims = new List<Claim> { new Claim(ClaimTypes.Name, user.UserName) };

foreach (var role in roles)

claims.Add(new Claim(ClaimTypes.Role, role));

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

Expires = DateTime.UtcNow.AddHours(3),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return Ok(new { token = tokenHandler.WriteToken(token) });

}

return Unauthorized("Invalid login");

}

}

### **4. Secure Controllers with Roles**

Modify AML, Trustee & SP controllers:

csharp

CopyEdit

[Authorize(Roles = "Admin")]

This will ensure only **Admin** can do CRUD.

## **PART 2: React + Redux + Material UI Frontend**

We will now create a **React app** that consumes the above APIs.

### **1. React Project Setup**

bash

CopyEdit

npx create-react-app pes-portal

cd pes-portal

npm install @mui/material @mui/icons-material axios react-redux @reduxjs/toolkit react-router-dom

### **2. Redux Store Setup**

src/store.js

javascript

CopyEdit

import { configureStore } from "@reduxjs/toolkit";

import authReducer from "./slices/authSlice";

export const store = configureStore({

reducer: {

auth: authReducer

}

});

src/slices/authSlice.js

javascript

CopyEdit

import { createSlice } from "@reduxjs/toolkit";

const authSlice = createSlice({

name: "auth",

initialState: { token: null },

reducers: {

setToken: (state, action) => {

state.token = action.payload;

},

logout: (state) => {

state.token = null;

}

}

});

export const { setToken, logout } = authSlice.actions;

export default authSlice.reducer;

### **3. Axios Instance**

src/api.js

javascript

CopyEdit

import axios from "axios";

const api = axios.create({

baseURL: "http://localhost:5000/api"

});

api.interceptors.request.use((config) => {

const token = localStorage.getItem("token");

if (token) config.headers.Authorization = `Bearer ${token}`;

return config;

});

export default api;

### **4. UI Components (Material UI)**

**Login Page:**

javascript

CopyEdit

import React, { useState } from "react";

import { Button, TextField, Container, Typography } from "@mui/material";

import api from "../api";

import { useDispatch } from "react-redux";

import { setToken } from "../slices/authSlice";

export default function Login() {

const [username, setUsername] = useState("");

const [password, setPassword] = useState("");

const dispatch = useDispatch();

const handleLogin = async () => {

const res = await api.post("/auth/login", { username, password });

localStorage.setItem("token", res.data.token);

dispatch(setToken(res.data.token));

};

return (

<Container maxWidth="sm">

<Typography variant="h4">Login</Typography>

<TextField label="Username" fullWidth margin="normal" value={username} onChange={(e) => setUsername(e.target.value)} />

<TextField label="Password" type="password" fullWidth margin="normal" value={password} onChange={(e) => setPassword(e.target.value)} />

<Button variant="contained" fullWidth onClick={handleLogin}>Login</Button>

</Container>

);

}

### **5. AML List Component (CRUD)**

**AMLList.jsx**

javascript

CopyEdit

import React, { useEffect, useState } from "react";

import api from "../api";

import { Button, Table, TableHead, TableRow, TableCell, TableBody, Container } from "@mui/material";

export default function AMLList() {

const [amlList, setAmlList] = useState([]);

const fetchAML = async () => {

const res = await api.get("/aml");

setAmlList(res.data);

};

const deleteAML = async (id) => {

await api.delete(`/aml/${id}`);

fetchAML();

};

useEffect(() => {

fetchAML();

}, []);

return (

<Container>

<h2>AML Records</h2>

<Table>

<TableHead>

<TableRow>

<TableCell>AML\_ID</TableCell>

<TableCell>Plan Number</TableCell>

<TableCell>Status</TableCell>

<TableCell>Actions</TableCell>

</TableRow>

</TableHead>

<TableBody>

{amlList.map((aml) => (

<TableRow key={aml.AML\_ID}>

<TableCell>{aml.AML\_ID}</TableCell>

<TableCell>{aml.Plan\_Number}</TableCell>

<TableCell>{aml.Status}</TableCell>

<TableCell>

<Button color="error" onClick={() => deleteAML(aml.AML\_ID)}>Delete</Button>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</Container>

);

}

## **NEXT STEPS**

1️⃣ Add **Trustee & SP Components (CRUD)** similar to AML.  
 2️⃣ Add **React Router** for navigation (Login → AMLList → TrusteeList → SPList).  
 3️⃣ Deploy both backend & frontend.

👉 Do you want me to now **build Trustee & SP React Components with CRUD (Material UI)** **and link everything with routing**?  
 Or should I first **add Register & Role-based UI (Admin/User)**?

##### **You said:**

Which suits easy and best

###### **ChatGPT said:**

Good question! Here's what will make the project **easiest to build and best to showcase**:

## **Best & Easiest Flow**

### **1. Build full CRUD UI (Trustee & SP React Components + Routing)**

* **Why?**
  + This gives you a working full-stack app (Login → List → Add/Edit/Delete).
  + You can demo the app end-to-end quickly.
  + React components are similar to AML (just copy, adjust columns, and done).

### **2. Add Role-based UI (Admin/User)**

* **Why?**
  + Role-based UI can be easily added later with small condition checks (if role === 'Admin').
  + It’s just hiding buttons like **Add/Delete** for normal users.
  + Doesn’t block you from testing CRUD features.

## **Recommendation:**

➡️ **Step 1:** Build **Trustee & SP React Components** with Material UI CRUD + Add Router navigation.  
 ➡️ **Step 2:** Add **Role-based UI + Register page** as an enhancement.

This way you’ll have a **fully functional MVP app fast** and then enhance it.

👉 Do you want me to now **write the full Trustee & SP React Components (CRUD) + React Router setup** so you can navigate between pages?  
 *(This will complete your MVP app)*

##### **You said:**

1.

###### **ChatGPT said:**

Great! Let's build **Trustee & SP React Components with CRUD + React Router navigation**.

## **1. Install React Router**

In your React project:

bash

CopyEdit

npm install react-router-dom

## **2. Setup React Router**

Edit src/index.js:

javascript

CopyEdit

import React from "react";

import ReactDOM from "react-dom/client";

import { BrowserRouter } from "react-router-dom";

import { Provider } from "react-redux";

import { store } from "./store";

import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(

<Provider store={store}>

<BrowserRouter>

<App />

</BrowserRouter>

</Provider>

);

## **3. App.js with Routes**

src/App.js

javascript

CopyEdit

import React from "react";

import { Routes, Route, Link } from "react-router-dom";

import Login from "./components/Login";

import AMLList from "./components/AMLList";

import TrusteeList from "./components/TrusteeList";

import SPList from "./components/SPList";

export default function App() {

return (

<div>

<nav style={{ padding: "10px", background: "#f5f5f5" }}>

<Link to="/aml" style={{ marginRight: "15px" }}>AML</Link>

<Link to="/trustee" style={{ marginRight: "15px" }}>Trustee</Link>

<Link to="/sp">Service Providers</Link>

</nav>

<Routes>

<Route path="/" element={<Login />} />

<Route path="/aml" element={<AMLList />} />

<Route path="/trustee" element={<TrusteeList />} />

<Route path="/sp" element={<SPList />} />

</Routes>

</div>

);

}

## **4. TrusteeList Component (CRUD)**

src/components/TrusteeList.jsx

javascript

CopyEdit

import React, { useEffect, useState } from "react";

import api from "../api";

import { Button, Table, TableHead, TableRow, TableCell, TableBody, Container, TextField, Dialog, DialogActions, DialogContent, DialogTitle } from "@mui/material";

export default function TrusteeList() {

const [trustees, setTrustees] = useState([]);

const [open, setOpen] = useState(false);

const [form, setForm] = useState({ name: "", email: "", phone: "", plan\_Number: "" });

const [editId, setEditId] = useState(null);

const fetchTrustees = async () => {

const res = await api.get("/trustee");

setTrustees(res.data);

};

useEffect(() => {

fetchTrustees();

}, []);

const handleSave = async () => {

if (editId) {

await api.put(`/trustee/${editId}`, form);

} else {

await api.post("/trustee", form);

}

fetchTrustees();

setOpen(false);

setForm({ name: "", email: "", phone: "", plan\_Number: "" });

setEditId(null);

};

const handleEdit = (t) => {

setForm({ name: t.Name, email: t.Email, phone: t.Phone, plan\_Number: t.Plan\_Number });

setEditId(t.Trustee\_ID);

setOpen(true);

};

const handleDelete = async (id) => {

await api.delete(`/trustee/${id}`);

fetchTrustees();

};

return (

<Container>

<h2>Trustee Records</h2>

<Button variant="contained" onClick={() => setOpen(true)}>Add Trustee</Button>

<Table>

<TableHead>

<TableRow>

<TableCell>Trustee\_ID</TableCell>

<TableCell>Name</TableCell>

<TableCell>Email</TableCell>

<TableCell>Phone</TableCell>

<TableCell>Plan Number</TableCell>

<TableCell>Actions</TableCell>

</TableRow>

</TableHead>

<TableBody>

{trustees.map((t) => (

<TableRow key={t.Trustee\_ID}>

<TableCell>{t.Trustee\_ID}</TableCell>

<TableCell>{t.Name}</TableCell>

<TableCell>{t.Email}</TableCell>

<TableCell>{t.Phone}</TableCell>

<TableCell>{t.Plan\_Number}</TableCell>

<TableCell>

<Button onClick={() => handleEdit(t)}>Edit</Button>

<Button color="error" onClick={() => handleDelete(t.Trustee\_ID)}>Delete</Button>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

{/\* Modal Form \*/}

<Dialog open={open} onClose={() => setOpen(false)}>

<DialogTitle>{editId ? "Edit Trustee" : "Add Trustee"}</DialogTitle>

<DialogContent>

<TextField fullWidth label="Name" margin="normal" value={form.name} onChange={(e) => setForm({ ...form, name: e.target.value })} />

<TextField fullWidth label="Email" margin="normal" value={form.email} onChange={(e) => setForm({ ...form, email: e.target.value })} />

<TextField fullWidth label="Phone" margin="normal" value={form.phone} onChange={(e) => setForm({ ...form, phone: e.target.value })} />

<TextField fullWidth label="Plan Number" margin="normal" value={form.plan\_Number} onChange={(e) => setForm({ ...form, plan\_Number: e.target.value })} />

</DialogContent>

<DialogActions>

<Button onClick={() => setOpen(false)}>Cancel</Button>

<Button variant="contained" onClick={handleSave}>Save</Button>

</DialogActions>

</Dialog>

</Container>

);

}

## **5. SPList Component (CRUD)**

src/components/SPList.jsx

javascript

CopyEdit

import React, { useEffect, useState } from "react";

import api from "../api";

import { Button, Table, TableHead, TableRow, TableCell, TableBody, Container, TextField, Dialog, DialogActions, DialogContent, DialogTitle } from "@mui/material";

export default function SPList() {

const [spList, setSpList] = useState([]);

const [open, setOpen] = useState(false);

const [form, setForm] = useState({ name: "", service\_Type: "", trustee\_ID: "" });

const [editId, setEditId] = useState(null);

const fetchSP = async () => {

const res = await api.get("/sp");

setSpList(res.data);

};

useEffect(() => {

fetchSP();

}, []);

const handleSave = async () => {

if (editId) {

await api.put(`/sp/${editId}`, form);

} else {

await api.post("/sp", form);

}

fetchSP();

setOpen(false);

setForm({ name: "", service\_Type: "", trustee\_ID: "" });

setEditId(null);

};

const handleEdit = (sp) => {

setForm({ name: sp.Name, service\_Type: sp.Service\_Type, trustee\_ID: sp.Trustee\_ID });

setEditId(sp.SP\_ID);

setOpen(true);

};

const handleDelete = async (id) => {

await api.delete(`/sp/${id}`);

fetchSP();

};

return (

<Container>

<h2>Service Provider Records</h2>

<Button variant="contained" onClick={() => setOpen(true)}>Add SP</Button>

<Table>

<TableHead>

<TableRow>

<TableCell>SP\_ID</TableCell>

<TableCell>Name</TableCell>

<TableCell>Service Type</TableCell>

<TableCell>Trustee ID</TableCell>

<TableCell>Actions</TableCell>

</TableRow>

</TableHead>

<TableBody>

{spList.map((sp) => (

<TableRow key={sp.SP\_ID}>

<TableCell>{sp.SP\_ID}</TableCell>

<TableCell>{sp.Name}</TableCell>

<TableCell>{sp.Service\_Type}</TableCell>

<TableCell>{sp.Trustee\_ID}</TableCell>

<TableCell>

<Button onClick={() => handleEdit(sp)}>Edit</Button>

<Button color="error" onClick={() => handleDelete(sp.SP\_ID)}>Delete</Button>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

{/\* Modal Form \*/}

<Dialog open={open} onClose={() => setOpen(false)}>

<DialogTitle>{editId ? "Edit SP" : "Add SP"}</DialogTitle>

<DialogContent>

<TextField fullWidth label="Name" margin="normal" value={form.name} onChange={(e) => setForm({ ...form, name: e.target.value })} />

<TextField fullWidth label="Service Type" margin="normal" value={form.service\_Type} onChange={(e) => setForm({ ...form, service\_Type: e.target.value })} />

<TextField fullWidth label="Trustee ID" margin="normal" value={form.trustee\_ID} onChange={(e) => setForm({ ...form, trustee\_ID: e.target.value })} />

</DialogContent>

<DialogActions>

<Button onClick={() => setOpen(false)}>Cancel</Button>

<Button variant="contained" onClick={handleSave}>Save</Button>

</DialogActions>

</Dialog>

</Container>

);

}

## **Now your MVP app is ready:**

1. **Login** using /auth/login
2. Navigate **AML → Trustee → SP** using navbar
3. Perform full **CRUD** operations

👉 Next, we can **add role-based UI (hide add/delete buttons for non-admins)** + a **Register Page** so you can create users.

Do you want me to:

1. Add **Role-based UI + Register Component** (next level)
2. Or show you how to **test the full app with Bruno/Swagger + React** first?

##### **You said:**

Both

###### **ChatGPT said:**

Great! We'll do **both:** 1️⃣ **Role-based UI + Register Component** 2️⃣ **Testing APIs with Swagger/Bruno + React flow**

## **PART 1: Role-Based UI + Register Component**

We already have **roles in Identity** (Admin/User). Now we’ll use them in **React UI**.

### **1. Modify Backend to Return Roles at Login**

Update AuthController.cs (Login endpoint):

csharp

CopyEdit

[HttpPost("login")]

public async Task<IActionResult> Login([FromBody] dynamic model)

{

var user = await \_userManager.FindByNameAsync((string)model.Username);

if (user != null && await \_userManager.CheckPasswordAsync(user, (string)model.Password))

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.UTF8.GetBytes("SuperSecretKey@345");

var roles = await \_userManager.GetRolesAsync(user);

var claims = new List<Claim> { new Claim(ClaimTypes.Name, user.UserName) };

foreach (var role in roles)

claims.Add(new Claim(ClaimTypes.Role, role));

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

Expires = DateTime.UtcNow.AddHours(3),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return Ok(new { token = tokenHandler.WriteToken(token), roles });

}

return Unauthorized("Invalid login");

}

### **2. Update React Auth Slice for Roles**

Update authSlice.js:

javascript

CopyEdit

import { createSlice } from "@reduxjs/toolkit";

const authSlice = createSlice({

name: "auth",

initialState: { token: null, roles: [] },

reducers: {

setAuth: (state, action) => {

state.token = action.payload.token;

state.roles = action.payload.roles;

},

logout: (state) => {

state.token = null;

state.roles = [];

}

}

});

export const { setAuth, logout } = authSlice.actions;

export default authSlice.reducer;

### **3. Modify Login Component**

Update Login.jsx:

javascript

CopyEdit

const handleLogin = async () => {

const res = await api.post("/auth/login", { username, password });

localStorage.setItem("token", res.data.token);

localStorage.setItem("roles", JSON.stringify(res.data.roles));

dispatch(setAuth({ token: res.data.token, roles: res.data.roles }));

};

### **4. Role-Based UI (Hide Buttons for Non-Admins)**

Example in TrusteeList.jsx & SPList.jsx:

javascript

CopyEdit

import { useSelector } from "react-redux";

const roles = useSelector((state) => state.auth.roles);

const isAdmin = roles.includes("Admin");

// Hide add/delete if not admin

{isAdmin && <Button variant="contained" onClick={() => setOpen(true)}>Add Trustee</Button>}

<Button color="error" disabled={!isAdmin} onClick={() => handleDelete(t.Trustee\_ID)}>Delete</Button>

### **5. Register Component (Only for Admin)**

Create Register.jsx:

javascript

CopyEdit

import React, { useState } from "react";

import { Button, TextField, Container, Typography } from "@mui/material";

import api from "../api";

export default function Register() {

const [username, setUsername] = useState("");

const [email, setEmail] = useState("");

const [password, setPassword] = useState("");

const handleRegister = async () => {

await api.post("/auth/register", { username, email, password });

alert("User Registered Successfully");

};

return (

<Container maxWidth="sm">

<Typography variant="h4">Register</Typography>

<TextField label="Username" fullWidth margin="normal" value={username} onChange={(e) => setUsername(e.target.value)} />

<TextField label="Email" fullWidth margin="normal" value={email} onChange={(e) => setEmail(e.target.value)} />

<TextField label="Password" type="password" fullWidth margin="normal" value={password} onChange={(e) => setPassword(e.target.value)} />

<Button variant="contained" fullWidth onClick={handleRegister}>Register</Button>

</Container>

);

}

Add route in App.js:

jsx

CopyEdit

import Register from "./components/Register";

<Route path="/register" element={<Register />} />

**Show link only if Admin:**

jsx

CopyEdit

{isAdmin && <Link to="/register" style={{ marginLeft: "15px" }}>Register User</Link>}

## **PART 2: Testing APIs (Swagger + Bruno + React)**

### **1. Swagger (Backend)**

Run backend → open **Swagger UI**:  
  
 bash  
CopyEdit  
https://localhost:5001/swagger/index.html

* Test endpoints:  
  + **/api/auth/register** → create user
  + **/api/auth/login** → get JWT
  + **Authorize button (Swagger)** → paste JWT token
  + Test **AML/Trustee/SP CRUD**

### **2. Bruno**

Install **Bruno API Client** (alternative to Postman):  
  
 arduino  
CopyEdit  
https://www.usebruno.com/download

* Create a collection:  
  + POST /auth/login → save token as environment variable
  + Add **Authorization: Bearer {{token}}** in CRUD requests
  + Test **AML/Trustee/SP endpoints**

### **3. React Testing**

Run frontend:  
  
 bash  
CopyEdit  
npm start

* Steps:  
  1. Login with Admin → Get JWT saved
  2. Navigate to AML / Trustee / SP → Perform CRUD
  3. Register new User
  4. Login as new User → Add/Delete buttons will be hidden

## **Your Full App Is Now:**

1. **Authentication + Role-based Authorization (Admin/User)**
2. **Material UI + React Router Navigation**
3. **Complete CRUD for AML, Trustee, SP tables**
4. **Swagger + Bruno for API testing**

👉 Do you want me to now **draw the full architecture diagram (Backend + Frontend + DB)** and also **package the project for deployment (IIS + Vercel)**?  
 Or should I give you a **Git-style folder structure with all final files**?

##### **You said:**

Yes

###### **ChatGPT said:**

Perfect! Let's finalize everything with **Architecture Diagram + Deployment Steps + Full Folder Structure**

## **1. Architecture Diagram**

Here’s the **end-to-end flow**:

scss

CopyEdit

┌─────────────────────────────┐

│ React Frontend │

│ (Material UI + Redux + JWT) │

└──────────────┬──────────────┘

│ HTTP (CORS)

▼

┌─────────────────────────────┐

│ ASP.NET Core MVC API │

│ Controllers (AML, Trustee, SP)

│ Auth (Identity + JWT) │

│ Dapper ORM │

└──────────────┬──────────────┘

│

┌─────────────▼─────────────┐

│ SQL Server │

│ PES Database (manual) │

│ Tables: Deepanshu\_AML, │

│ Deepanshu\_Trustee, SP │

└────────────────────────────┘

### **Key Points**

1. React uses axios with JWT token for all CRUD requests.
2. Backend uses **Dapper ORM (NOLOCK)** and **ASP.NET Core Identity** for security.
3. SQL Server holds all tables, created manually (no EF migrations).

## **2. Folder Structure (Final)**

css

CopyEdit

📦 FullStackApp/

├── backend/

│ ├── Controllers/

│ │ ├── AMLController.cs

│ │ ├── TrusteeController.cs

│ │ ├── SPController.cs

│ │ └── AuthController.cs

│ ├── Data/

│ │ ├── DapperContext.cs

│ │ └── ApplicationDbContext.cs

│ ├── Models/ (if needed for mapping)

│ ├── Program.cs

│ ├── appsettings.json

│ └── ... (Identity, JWT configs)

│

└── frontend/

├── src/

│ ├── api.js

│ ├── store.js

│ ├── slices/

│ │ └── authSlice.js

│ ├── components/

│ │ ├── Login.jsx

│ │ ├── Register.jsx

│ │ ├── AMLList.jsx

│ │ ├── TrusteeList.jsx

│ │ └── SPList.jsx

│ └── App.js

└── package.json

## **3. Deployment Steps**

### **A. Backend (IIS / Azure App Service)**

1. Open backend project in Visual Studio
2. dotnet publish -c Release
3. Deploy to **IIS** or **Azure App Service**

Ensure **CORS is enabled** in Program.cs:  
  
 csharp  
CopyEdit  
app.UseCors("AllowAll");

1. Connection string in **appsettings.Production.json** should point to production SQL Server

### **B. Frontend (Vercel / Netlify)**

In frontend/package.json, set API base URL in api.js to the deployed backend URL  
  
 javascript  
CopyEdit  
const api = axios.create({

baseURL: "https://your-backend-url/api"

});

Build:  
  
 bash  
CopyEdit  
npm run build

1. Deploy build/ folder to:  
   * **Vercel:** vercel deploy
   * **Netlify:** netlify deploy

### **C. Bruno / Swagger Testing in Prod**

Swagger will be available at:  
  
 arduino  
CopyEdit  
https://your-backend-url/swagger/index.html

* Bruno: Update base URL → test login & CRUD APIs

## **4. Workflow After Deployment**

1. **Admin Logs In** (gets JWT + Role)
2. Navigates to:  
   * AML (CRUD)
   * Trustee (CRUD)
   * SP (CRUD)
3. **Registers Users** → New users can login but won't see Add/Delete buttons
4. React app consumes deployed APIs directly

## **5. NEXT?**

👉 Do you want me to:

1. **Prepare SQL scripts** for Deepanshu\_AML, Deepanshu\_Trustee, Deepanshu\_SP tables with correct PK/FK relations (for production)?
2. Or create **Docker setup** (SQL + backend + frontend) for full containerized deployment?
3. Or do you want a **sample GitHub repo structure with all code ready**?

Tools

ChatGPT can make mistakes. Check important info. See Cookie Preferences.