reat! Let's start by **setting up the environment and installing all dependencies** step by step.

## **1️⃣ Backend Setup – ASP.NET Core API (with Dapper & Identity)**

### **a) Create the ASP.NET Core Project**

bash

CopyEdit

dotnet new webapi -n PES.WebAPI

cd PES.WebAPI

### **b) Install Dependencies**

**Dapper** (for database operations)  
  
 bash  
CopyEdit  
dotnet add package Dapper

**SQL Client** (to connect to MS SQL Server)  
  
 bash  
CopyEdit  
dotnet add package Microsoft.Data.SqlClient

**Swagger (API Documentation)** bash  
CopyEdit  
dotnet add package Swashbuckle.AspNetCore

**ASP.NET Core Identity (Authentication)** bash  
CopyEdit  
dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

**JWT (if you want token-based auth)** *(optional for now)* bash  
CopyEdit  
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

### **c) Verify Packages Installed**

bash

CopyEdit

dotnet list package

You should see: Dapper, SqlClient, Identity, EF Core, Swagger, etc.

## **2️⃣ Frontend Setup – React + Redux + Material UI**

### **a) Create React App**

bash

CopyEdit

npx create-react-app pes-react

cd pes-react

### **b) Install Dependencies**

**Material UI (UI components)** bash  
CopyEdit  
npm install @mui/material @emotion/react @emotion/styled

**Redux Toolkit (state management)** bash  
CopyEdit  
npm install @reduxjs/toolkit react-redux

**Axios (for API calls)** bash  
CopyEdit  
npm install axios

**React Router (for navigation)** bash  
CopyEdit  
npm install react-router-dom

### **c) Verify**

bash

CopyEdit

npm list --depth=0

You should see: React, Material UI, Redux Toolkit, Axios, React Router.

## **3️⃣ Summary of Tools Installed**

* **Backend:** Dapper, SqlClient, Identity, Swagger, EF Core
* **Frontend:** Material UI, Redux Toolkit, Axios, React Router

### **✅ Next Step:**

1. Setup **database connection & DapperContext (Singleton)** in the backend.
2. Add **CORS & Swagger** so React can talk to backend.
3. Create a **basic GET API** (e.g., fetch all Trustees) and test it.

👉 Do you want me to **show how to setup DapperContext and connect to MS SQL now**? Or do you first want the **folder structure & Program.cs configuration** for the backend?

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

Name of project and structure

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

Perfect! Let’s finalize the **name** and **project structure** for clarity before we dive deeper.

## **1️⃣ Project Name**

**PES.DataPortal** *(PES stands for your database context, DataPortal indicates a read-only data dashboard)*

We'll create two parts:

1. **Backend:** PES.DataPortal.API (ASP.NET Core Web API)
2. **Frontend:** PES.DataPortal.UI (React + Material UI)

## **2️⃣ Folder / Project Structure**

### **A) Backend: ASP.NET Core API (PES.DataPortal.API)**

pgsql

CopyEdit

PES.DataPortal.API/

├── Controllers/

│ ├── TrusteeController.cs

│ ├── AMLController.cs

│ ├── SPController.cs

│

├── Data/

│ ├── DapperContext.cs (Singleton for DB connection)

│ ├── Repositories/

│ ├── TrusteeRepository.cs

│ ├── AMLRepository.cs

│ ├── SPRepository.cs

│

├── Models/

│ ├── Trustee.cs

│ ├── AML.cs

│ ├── SP.cs

│

├── Services/

│ ├── IdentityService.cs (ASP.NET Identity login)

│

├── Properties/

│ ├── launchSettings.json

│

├── appsettings.json (DB connection string)

├── Program.cs

### **B) Frontend: React App (PES.DataPortal.UI)**

pgsql

CopyEdit

PES.DataPortal.UI/

├── src/

│ ├── components/

│ ├── Dashboard.js

│ ├── TrusteeList.js

│ ├── AMLList.js

│ ├── SPList.js

│

│ ├── redux/

│ ├── store.js

│ ├── slices/

│ ├── trusteeSlice.js

│ ├── amlSlice.js

│ ├── spSlice.js

│

│ ├── pages/

│ ├── LoginPage.js

│ ├── DashboardPage.js

│

│ ├── App.js

│ ├── index.js

│

├── public/

│ ├── index.html

## **3️⃣ Architecture Overview**

* **Frontend:** React + Material UI → Displays dashboard, tables & charts. Uses Redux for state.
* **Backend:** ASP.NET Core API + Dapper → Fetches data from MS SQL (read-only).
* **Authentication:** ASP.NET Identity (optional JWT).
* **Communication:** REST APIs (GET only) + Swagger for documentation.
* **Database:** MS SQL Server with tables: AML, Trustee, SP.

## **4️⃣ Next Steps**

1. Setup **Backend base project** with:  
   * DapperContext (Singleton)
   * CORS and Swagger in Program.cs
   * Basic TrusteeController with a sample GET /api/trustee
2. Once backend is working, start **Frontend scaffolding**.

👉 **Do you want me to now write the Program.cs setup + DapperContext (Singleton) + DB connection first?** Or should I show you the **database connection string setup (appsettings.json)** with folder creation?

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

Yes

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

Great! Let’s now set up the **Backend (PES.DataPortal.API)** with Program.cs, appsettings.json, and DapperContext (Singleton) to connect to MS SQL.

## **1️⃣ appsettings.json – Database Connection**

Open appsettings.json and add your connection string:

json

CopyEdit

{

"ConnectionStrings": {

"DefaultConnection": "Server=YOUR\_SERVER\_NAME;Database=PES;User Id=YOUR\_DB\_USER;Password=YOUR\_PASSWORD;TrustServerCertificate=True;"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

⚡ Replace **YOUR\_SERVER\_NAME**, **YOUR\_DB\_USER**, and **YOUR\_PASSWORD** with your SQL Server details.

## **2️⃣ DapperContext.cs (Singleton)**

Create a folder Data/ and add **DapperContext.cs**:

csharp

CopyEdit

using System.Data;

using Microsoft.Data.SqlClient;

namespace PES.DataPortal.API.Data

{

public class DapperContext

{

private readonly IConfiguration \_configuration;

private readonly string \_connectionString;

public DapperContext(IConfiguration configuration)

{

\_configuration = configuration;

\_connectionString = \_configuration.GetConnectionString("DefaultConnection");

}

public IDbConnection CreateConnection()

=> new SqlConnection(\_connectionString);

}

}

## **3️⃣ Program.cs – Register Dependencies**

Edit **Program.cs** to register **DapperContext**, CORS, Swagger:

csharp

CopyEdit

using PES.DataPortal.API.Data;

var builder = WebApplication.CreateBuilder(args);

// Add Controllers

builder.Services.AddControllers();

// Register DapperContext as Singleton

builder.Services.AddSingleton<DapperContext>();

// Enable Swagger

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Enable CORS for React App

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowReactApp",

builder => builder.WithOrigins("http://localhost:3000")

.AllowAnyHeader()

.AllowAnyMethod());

});

var app = builder.Build();

// Enable Swagger in development

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

// Enable CORS

app.UseCors("AllowReactApp");

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

## **4️⃣ Create a Sample Controller (Trustee)**

Inside Controllers/TrusteeController.cs:

csharp

CopyEdit

using Microsoft.AspNetCore.Mvc;

using PES.DataPortal.API.Data;

using Dapper;

namespace PES.DataPortal.API.Controllers

{

[ApiController]

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

public class TrusteeController : ControllerBase

{

private readonly DapperContext \_context;

public TrusteeController(DapperContext context)

{

\_context = context;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var query = "SELECT TOP 10 \* FROM Trustee"; // Fetch first 10 rows

using var connection = \_context.CreateConnection();

var trustees = await connection.QueryAsync(query);

return Ok(trustees);

}

}

}

## **5️⃣ Run and Test**

Run backend:  
  
 bash  
CopyEdit  
dotnet run

1. Open browser:  
   * Swagger UI → [**https://localhost:5001/swagger**](https://localhost:5001/swagger)
   * Test endpoint → **GET** [**https://localhost:5001/api/trustee**](https://localhost:5001/api/trustee)

If everything is fine, you should see trustee data from MS SQL.

## **Next Step**

👉 Once this base setup is working, we will:

1. **Create models** (Trustee, AML, SP).
2. Add **Repository pattern** for clean code.
3. Move to **Frontend React** (Material UI + Redux).

Do you want me to now **add models + repository (clean data access)** for Trustee, AML, and SP tables? Or first show you **how to test this API in Swagger with real DB data**?

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

Yes

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

Great! Let’s now **add Models and Repository pattern** for clean data access for all three tables (Trustee, AML, SP).

## **1️⃣ Models (Data Classes)**

Inside Models/ folder, create these files:

### **a) Trustee.cs**

csharp

CopyEdit

namespace PES.DataPortal.API.Models

{

public class Trustee

{

public int TrusteeID { get; set; }

public string Name { get; set; }

public string Email { get; set; }

public DateTime CreatedDate { get; set; }

}

}

### **b) AML.cs**

csharp

CopyEdit

namespace PES.DataPortal.API.Models

{

public class AML

{

public int AMLID { get; set; }

public int TrusteeID { get; set; }

public string DocumentPath { get; set; }

public string Status { get; set; }

public DateTime VerifiedDate { get; set; }

}

}

### **c) SP.cs (Service Providers)**

csharp

CopyEdit

namespace PES.DataPortal.API.Models

{

public class SP

{

public int SPID { get; set; }

public int TrusteeID { get; set; }

public string Name { get; set; }

public DateTime ContractStart { get; set; }

public DateTime ContractEnd { get; set; }

}

}

⚠️ Adjust column names & types if they are different in your DB.

## **2️⃣ Repositories (Dapper Queries)**

Inside Data/Repositories/ folder:

### **a) ITrusteeRepository.cs**

csharp

CopyEdit

using PES.DataPortal.API.Models;

namespace PES.DataPortal.API.Data.Repositories

{

public interface ITrusteeRepository

{

Task<IEnumerable<Trustee>> GetAllAsync();

}

}

### **b) TrusteeRepository.cs**

csharp

CopyEdit

using Dapper;

using PES.DataPortal.API.Models;

namespace PES.DataPortal.API.Data.Repositories

{

public class TrusteeRepository : ITrusteeRepository

{

private readonly DapperContext \_context;

public TrusteeRepository(DapperContext context)

{

\_context = context;

}

public async Task<IEnumerable<Trustee>> GetAllAsync()

{

var query = "SELECT TrusteeID, Name, Email, CreatedDate FROM Trustee";

using var connection = \_context.CreateConnection();

return await connection.QueryAsync<Trustee>(query);

}

}

}

### **c) AMLRepository.cs**

csharp

CopyEdit

using Dapper;

using PES.DataPortal.API.Models;

namespace PES.DataPortal.API.Data.Repositories

{

public class AMLRepository

{

private readonly DapperContext \_context;

public AMLRepository(DapperContext context)

{

\_context = context;

}

public async Task<IEnumerable<AML>> GetByTrusteeAsync(int trusteeId)

{

var query = "SELECT AMLID, TrusteeID, DocumentPath, Status, VerifiedDate FROM AML WHERE TrusteeID = @TrusteeID";

using var connection = \_context.CreateConnection();

return await connection.QueryAsync<AML>(query, new { TrusteeID = trusteeId });

}

}

}

### **d) SPRepository.cs**

csharp

CopyEdit

using Dapper;

using PES.DataPortal.API.Models;

namespace PES.DataPortal.API.Data.Repositories

{

public class SPRepository

{

private readonly DapperContext \_context;

public SPRepository(DapperContext context)

{

\_context = context;

}

public async Task<IEnumerable<SP>> GetByTrusteeAsync(int trusteeId)

{

var query = "SELECT SPID, TrusteeID, Name, ContractStart, ContractEnd FROM SP WHERE TrusteeID = @TrusteeID";

using var connection = \_context.CreateConnection();

return await connection.QueryAsync<SP>(query, new { TrusteeID = trusteeId });

}

}

}

## **3️⃣ Update Program.cs – Register Repositories**

Add these lines to **Program.cs**:

csharp

CopyEdit

builder.Services.AddScoped<ITrusteeRepository, TrusteeRepository>();

builder.Services.AddScoped<AMLRepository>();

builder.Services.AddScoped<SPRepository>();

## **4️⃣ Update Controller to Use Repository**

Modify TrusteeController.cs:

csharp

CopyEdit

using Microsoft.AspNetCore.Mvc;

using PES.DataPortal.API.Data.Repositories;

namespace PES.DataPortal.API.Controllers

{

[ApiController]

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

public class TrusteeController : ControllerBase

{

private readonly ITrusteeRepository \_trusteeRepository;

public TrusteeController(ITrusteeRepository trusteeRepository)

{

\_trusteeRepository = trusteeRepository;

}

[HttpGet]

public async Task<IActionResult> GetAll()

{

var trustees = await \_trusteeRepository.GetAllAsync();

return Ok(trustees);

}

}

}

## **5️⃣ Add AML and SP Controllers**

**AMLController.cs**

csharp

CopyEdit

using Microsoft.AspNetCore.Mvc;

using PES.DataPortal.API.Data.Repositories;

namespace PES.DataPortal.API.Controllers

{

[ApiController]

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

public class AMLController : ControllerBase

{

private readonly AMLRepository \_amlRepository;

public AMLController(AMLRepository amlRepository)

{

\_amlRepository = amlRepository;

}

[HttpGet("{trusteeId}")]

public async Task<IActionResult> GetAMLByTrustee(int trusteeId)

{

var aml = await \_amlRepository.GetByTrusteeAsync(trusteeId);

return Ok(aml);

}

}

}

**SPController.cs**

csharp

CopyEdit

using Microsoft.AspNetCore.Mvc;

using PES.DataPortal.API.Data.Repositories;

namespace PES.DataPortal.API.Controllers

{

[ApiController]

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

public class SPController : ControllerBase

{

private readonly SPRepository \_spRepository;

public SPController(SPRepository spRepository)

{

\_spRepository = spRepository;

}

[HttpGet("{trusteeId}")]

public async Task<IActionResult> GetSPByTrustee(int trusteeId)

{

var sp = await \_spRepository.GetByTrusteeAsync(trusteeId);

return Ok(sp);

}

}

}

## **6️⃣ Test in Swagger**

Run the project:

bash

CopyEdit

dotnet run

Swagger endpoints:

* **GET** /api/trustee → list of Trustees
* **GET** /api/aml/{trusteeId} → AML records for Trustee
* **GET** /api/sp/{trusteeId} → SP records for Trustee

## **✅ Next Step:**

Now that backend is ready:

1. **Do you want me to add Authentication (Identity login) now?**
2. Or **start setting up React + Material UI frontend** (to call these APIs and display dashboards)?

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

Both

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

Great! We'll tackle **Authentication (Identity)** and **Frontend (React + Material UI)** in **parallel**. Here's the roadmap so you can progress step-by-step without confusion.

## **1️⃣ Authentication Setup (ASP.NET Core Identity)**

We will enable **login/logout** for Admin and Trustee roles.

### **Step 1: Add EF Core Context for Identity**

1. Create Data/ApplicationDbContext.cs:

csharp

CopyEdit

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace PES.DataPortal.API.Data

{

public class ApplicationDbContext : IdentityDbContext

{

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

}

}

### **Step 2: Register Identity in Program.cs**

csharp

CopyEdit

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Identity;

using PES.DataPortal.API.Data;

var builder = WebApplication.CreateBuilder(args);

// Add Identity + EF Core

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

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

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

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

// Add Authentication Middleware

builder.Services.AddAuthentication();

### **Step 3: Create AuthController**

Controllers/AuthController.cs

csharp

CopyEdit

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

namespace PES.DataPortal.API.Controllers

{

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly SignInManager<IdentityUser> \_signInManager;

public AuthController(UserManager<IdentityUser> userManager, SignInManager<IdentityUser> signInManager)

{

\_userManager = userManager;

\_signInManager = signInManager;

}

[HttpPost("register")]

public async Task<IActionResult> Register(string email, string password)

{

var user = new IdentityUser { UserName = email, Email = email };

var result = await \_userManager.CreateAsync(user, password);

if (result.Succeeded)

return Ok(new { message = "User Registered" });

return BadRequest(result.Errors);

}

[HttpPost("login")]

public async Task<IActionResult> Login(string email, string password)

{

var result = await \_signInManager.PasswordSignInAsync(email, password, false, false);

if (result.Succeeded)

return Ok(new { message = "Login Successful" });

return Unauthorized(new { message = "Invalid Credentials" });

}

}

}

### **Step 4: Update Middleware**

In Program.cs add:

csharp

CopyEdit

app.UseAuthentication();

app.UseAuthorization();

### **Step 5: Create Database & Seed User**

Run migrations:

bash

CopyEdit

dotnet ef migrations add InitIdentity

dotnet ef database update

Now you can register users using Swagger:

* POST /api/auth/register?email=admin@test.com&password=Test@123
* Then login using POST /api/auth/login

## **2️⃣ Frontend Setup (React + Material UI + Redux)**

### **Step 1: Initialize React (if not done)**

bash

CopyEdit

npx create-react-app pes-dataportal-ui

cd pes-dataportal-ui

Install dependencies:

bash

CopyEdit

npm install @mui/material @emotion/react @emotion/styled

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

### **Step 2: Setup Redux Store**

src/redux/store.js

javascript

CopyEdit

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

import trusteeReducer from './slices/trusteeSlice';

import authReducer from './slices/authSlice';

export const store = configureStore({

reducer: {

trustees: trusteeReducer,

auth: authReducer

},

});

### **Step 3: Auth Slice**

src/redux/slices/authSlice.js

javascript

CopyEdit

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import axios from 'axios';

export const login = createAsyncThunk('auth/login', async ({ email, password }) => {

const res = await axios.post(`https://localhost:5001/api/auth/login?email=${email}&password=${password}`);

return res.data;

});

const authSlice = createSlice({

name: 'auth',

initialState: { user: null, status: null },

extraReducers: (builder) => {

builder.addCase(login.fulfilled, (state, action) => {

state.user = action.payload;

state.status = 'logged\_in';

});

},

});

export default authSlice.reducer;

### **Step 4: Login Page**

src/pages/LoginPage.js

javascript

CopyEdit

import React, { useState } from 'react';

import { useDispatch } from 'react-redux';

import { login } from '../redux/slices/authSlice';

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

const LoginPage = () => {

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

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

const dispatch = useDispatch();

const handleSubmit = (e) => {

e.preventDefault();

dispatch(login({ email, password }));

};

return (

<Container maxWidth="xs">

<h2>Login</h2>

<form onSubmit={handleSubmit}>

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

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

<Button type="submit" variant="contained" sx={{ mt: 2 }}>Login</Button>

</form>

</Container>

);

};

export default LoginPage;

### **Step 5: Trustee Dashboard (After Login)**

src/components/TrusteeList.js

javascript

CopyEdit

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

import axios from 'axios';

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

const TrusteeList = () => {

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

useEffect(() => {

axios.get('https://localhost:5001/api/trustee').then(res => setTrustees(res.data));

}, []);

return (

<Container>

<h2>Trustee Data</h2>

<Table>

<TableHead>

<TableRow>

<TableCell>Name</TableCell>

<TableCell>Email</TableCell>

<TableCell>Created</TableCell>

</TableRow>

</TableHead>

<TableBody>

{trustees.map(t => (

<TableRow key={t.trusteeID}>

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

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

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

</TableRow>

))}

</TableBody>

</Table>

</Container>

);

};

export default TrusteeList;

## **3️⃣ Next Step**

Now we have:

* **Backend:** Identity auth + APIs ready
* **Frontend:** Login page + Trustee data table

👉 **Do you want me to connect login flow → restrict Trustee dashboard only after login?** Or **should I now add AML & SP data tables + a full dashboard page with Material UI cards & charts?**

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

Yes

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

Perfect! Let's now **connect the login flow with access control** and build a **Dashboard page** (Trustees + AML + SP data with Material UI cards & charts).

## **1️⃣ Protect Trustee Dashboard (Only after Login)**

### **Step 1: Update authSlice.js for state & logout**

javascript

CopyEdit

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

import axios from 'axios';

export const login = createAsyncThunk('auth/login', async ({ email, password }) => {

const res = await axios.post(`https://localhost:5001/api/auth/login?email=${email}&password=${password}`);

return { email, ...res.data };

});

const authSlice = createSlice({

name: 'auth',

initialState: { user: null, status: null },

reducers: {

logout: (state) => {

state.user = null;

state.status = null;

}

},

extraReducers: (builder) => {

builder.addCase(login.fulfilled, (state, action) => {

state.user = action.payload;

state.status = 'logged\_in';

});

},

});

export const { logout } = authSlice.actions;

export default authSlice.reducer;

### **Step 2: Add Protected Route in App.js**

src/App.js

javascript

CopyEdit

import React from 'react';

import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';

import { useSelector } from 'react-redux';

import LoginPage from './pages/LoginPage';

import DashboardPage from './pages/DashboardPage';

function App() {

const { user } = useSelector(state => state.auth);

return (

<BrowserRouter>

<Routes>

<Route path="/login" element={<LoginPage />} />

<Route

path="/dashboard"

element={user ? <DashboardPage /> : <Navigate to="/login" />}

/>

<Route path="\*" element={<Navigate to="/login" />} />

</Routes>

</BrowserRouter>

);

}

export default App;

## **2️⃣ Dashboard Page (Cards + Trustee Table + AML/SP Links)**

### **Step 1: DashboardPage.js**

src/pages/DashboardPage.js

javascript

CopyEdit

import React from 'react';

import { Container, Grid, Card, CardContent, Typography, Button } from '@mui/material';

import TrusteeList from '../components/TrusteeList';

import { useDispatch } from 'react-redux';

import { logout } from '../redux/slices/authSlice';

const DashboardPage = () => {

const dispatch = useDispatch();

return (

<Container>

<Grid container justifyContent="space-between" alignItems="center" sx={{ mb: 3 }}>

<Typography variant="h4">PES Data Dashboard</Typography>

<Button variant="outlined" color="error" onClick={() => dispatch(logout())}>

Logout

</Button>

</Grid>

<Grid container spacing={2} sx={{ mb: 3 }}>

<Grid item xs={12} md={4}>

<Card>

<CardContent>

<Typography variant="h6">Total Trustees</Typography>

<Typography variant="h5" color="primary">12</Typography> {/\* Hardcoded for now \*/}

</CardContent>

</Card>

</Grid>

<Grid item xs={12} md={4}>

<Card>

<CardContent>

<Typography variant="h6">AML Pending</Typography>

<Typography variant="h5" color="secondary">4</Typography>

</CardContent>

</Card>

</Grid>

<Grid item xs={12} md={4}>

<Card>

<CardContent>

<Typography variant="h6">SP Expiring Soon</Typography>

<Typography variant="h5" color="error">2</Typography>

</CardContent>

</Card>

</Grid>

</Grid>

<TrusteeList />

</Container>

);

};

export default DashboardPage;

## **3️⃣ Link AML & SP Details (Drill-down)**

### **Step 1: Update TrusteeList.js**

Add clickable row for details:

javascript

CopyEdit

<TableBody>

{trustees.map(t => (

<TableRow

key={t.trusteeID}

onClick={() => window.location = `/dashboard/${t.trusteeID}`}

style={{ cursor: 'pointer' }}

>

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

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

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

</TableRow>

))}

</TableBody>

### **Step 2: Create TrusteeDetailsPage.js**

src/pages/TrusteeDetailsPage.js

javascript

CopyEdit

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

import { useParams, Link } from 'react-router-dom';

import axios from 'axios';

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

const TrusteeDetailsPage = () => {

const { trusteeId } = useParams();

const [aml, setAML] = useState([]);

const [sp, setSP] = useState([]);

useEffect(() => {

axios.get(`https://localhost:5001/api/aml/${trusteeId}`).then(res => setAML(res.data));

axios.get(`https://localhost:5001/api/sp/${trusteeId}`).then(res => setSP(res.data));

}, [trusteeId]);

return (

<Container>

<Link to="/dashboard">

<Button variant="outlined" sx={{ mt: 2 }}>⬅ Back</Button>

</Link>

<Typography variant="h4" sx={{ my: 2 }}>Trustee Details</Typography>

<Typography variant="h6">AML Records</Typography>

<Table>

<TableHead>

<TableRow>

<TableCell>Document</TableCell>

<TableCell>Status</TableCell>

<TableCell>Verified Date</TableCell>

</TableRow>

</TableHead>

<TableBody>

{aml.map(a => (

<TableRow key={a.amlid}>

<TableCell><a href={a.documentPath}>View</a></TableCell>

<TableCell>{a.status}</TableCell>

<TableCell>{a.verifiedDate}</TableCell>

</TableRow>

))}

</TableBody>

</Table>

<Typography variant="h6" sx={{ mt: 4 }}>Service Providers</Typography>

<Table>

<TableHead>

<TableRow>

<TableCell>Name</TableCell>

<TableCell>Contract Start</TableCell>

<TableCell>Contract End</TableCell>

</TableRow>

</TableHead>

<TableBody>

{sp.map(s => (

<TableRow key={s.spid}>

<TableCell>{s.name}</TableCell>

<TableCell>{s.contractStart}</TableCell>

<TableCell>{s.contractEnd}</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</Container>

);

};

export default TrusteeDetailsPage;

### **Step 3: Add Route in App.js**

javascript

CopyEdit

import TrusteeDetailsPage from './pages/TrusteeDetailsPage';

<Route

path="/dashboard/:trusteeId"

element={user ? <TrusteeDetailsPage /> : <Navigate to="/login" />}

/>

## **4️⃣ Final Flow**

1. User goes to **/login**, logs in using Identity API.
2. Redirected to **/dashboard**:  
   * See **cards** (total Trustees, AML pending, etc.)
   * **Trustee Table** (click row → details)
3. **Trustee Details Page** shows AML + SP tables.

## **5️⃣ Next Step**

👉 Do you want me to **connect Dashboard cards (counts) with live API data**?  
 Or should I now **add Material UI charts (AML status % graph, expiring SP graph) to Dashboard**?

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

All done or remaining something?

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

We are **90% done**, but a few **polishing & production-ready features** are remaining to make the project complete and enterprise-level:

## **✅ What’s Already Done**

1. **Backend**
   * Dapper-based API with **Trustee, AML, SP** endpoints (read-only).
   * **Authentication with ASP.NET Identity** (Register/Login).
   * Swagger, CORS enabled.
   * Repository pattern implemented (clean code).
2. **Frontend**
   * React + Material UI setup.
   * Login & Protected Routes (Redux integrated).
   * Dashboard Page with Cards + Trustee Table.
   * Trustee Details Page (AML & SP data tables).

## **🔹 What’s Remaining (To Make it Enterprise-Ready)**

### **1️⃣ Dashboard Cards – Live Data**

* Right now, Total Trustees / AML Pending / SP Expiring Soon are **hardcoded**.
* We need APIs:  
  + GET /api/trustee/count
  + GET /api/aml/pendingCount
  + GET /api/sp/expiringSoon
* Connect these APIs to **update the numbers dynamically**.

### **2️⃣ Charts (Material UI + Chart.js/Recharts)**

* Show AML compliance % as a **pie/donut chart**.
* Show SP contract expiry timeline as a **bar/line chart**.
* Library: npm install recharts (or Chart.js).

### **3️⃣ Error Handling & Loading State**

* Add try/catch in frontend API calls.
* Show **loading spinners** using MUI CircularProgress when data is fetching.
* Show error alerts if API fails.

### **4️⃣ Roles & Permissions**

* Assign roles (Admin, Trustee) during registration.
* Trustees can only see **their own data**:  
  + Modify TrusteeController → filter by User.Identity.Name.
* Admin sees **all data**.

### **5️⃣ Token-Based Auth (JWT) *(Optional for now)***

* Currently using cookie/session login.
* For better **API security**, we can implement JWT tokens.

### **6️⃣ UI Polishing**

* Add **Material UI layout** (Header, Sidebar, Footer).
* Add **logout redirect**.
* Dark Mode toggle *(optional)*.

### **7️⃣ Deployment**

* Backend: Host on **Azure App Service** / IIS.
* Frontend: Host on **Vercel/Netlify** or **Azure Static Web App**.
* Use **Azure SQL Database** for production DB.

## **🔥 Next Action**

Do you want me to:

1. **Finish the Dashboard cards with live API data (counts)** first?
2. Or **add charts (AML %, SP expiry)** with Recharts?
3. Or **integrate roles & permissions** so Admin/Trustee see different data?

👉 Which one should we do now? (We can complete all remaining points one by one.)

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

Ok

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

Great! Let's wrap this up **step by step** so the project becomes production-ready.

## **1️⃣ Finish Dashboard Cards with Live API Data (Counts)**

### **Backend: Add Count APIs**

#### **TrusteeController.cs**

csharp

CopyEdit

[HttpGet("count")]

public async Task<IActionResult> GetCount()

{

var query = "SELECT COUNT(\*) FROM Trustee";

using var connection = \_trusteeRepository.GetConnection(); // Or use DapperContext directly

var count = await connection.ExecuteScalarAsync<int>(query);

return Ok(new { total = count });

}

#### **AMLController.cs**

csharp

CopyEdit

[HttpGet("pendingCount")]

public async Task<IActionResult> GetPendingCount()

{

var query = "SELECT COUNT(\*) FROM AML WHERE Status = 'Pending'";

using var connection = \_amlRepository.GetConnection();

var count = await connection.ExecuteScalarAsync<int>(query);

return Ok(new { total = count });

}

#### **SPController.cs**

csharp

CopyEdit

[HttpGet("expiringSoon")]

public async Task<IActionResult> GetExpiringSoonCount()

{

var query = "SELECT COUNT(\*) FROM SP WHERE ContractEnd < DATEADD(DAY,30,GETDATE())";

using var connection = \_spRepository.GetConnection();

var count = await connection.ExecuteScalarAsync<int>(query);

return Ok(new { total = count });

}

### **Frontend: Connect Dashboard Cards**

**DashboardPage.js**

javascript

CopyEdit

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

import axios from 'axios';

import { Container, Grid, Card, CardContent, Typography } from '@mui/material';

import TrusteeList from '../components/TrusteeList';

const DashboardPage = () => {

const [counts, setCounts] = useState({ trustees: 0, aml: 0, sp: 0 });

useEffect(() => {

async function fetchCounts() {

const t = await axios.get('https://localhost:5001/api/trustee/count');

const a = await axios.get('https://localhost:5001/api/aml/pendingCount');

const s = await axios.get('https://localhost:5001/api/sp/expiringSoon');

setCounts({

trustees: t.data.total,

aml: a.data.total,

sp: s.data.total

});

}

fetchCounts();

}, []);

return (

<Container>

<Grid container spacing={2} sx={{ mb: 3 }}>

<Grid item xs={12} md={4}>

<Card>

<CardContent>

<Typography variant="h6">Total Trustees</Typography>

<Typography variant="h5" color="primary">{counts.trustees}</Typography>

</CardContent>

</Card>

</Grid>

<Grid item xs={12} md={4}>

<Card>

<CardContent>

<Typography variant="h6">AML Pending</Typography>

<Typography variant="h5" color="secondary">{counts.aml}</Typography>

</CardContent>

</Card>

</Grid>

<Grid item xs={12} md={4}>

<Card>

<CardContent>

<Typography variant="h6">SP Expiring Soon</Typography>

<Typography variant="h5" color="error">{counts.sp}</Typography>

</CardContent>

</Card>

</Grid>

</Grid>

<TrusteeList />

</Container>

);

};

export default DashboardPage;

## **2️⃣ Next: Charts & Visual Insights**

We can now **add Recharts** (AML compliance % & SP expiry trends) to Dashboard:

bash

CopyEdit

npm install recharts

Then, add **PieChart** and **BarChart** components to DashboardPage.js using live data.

## **3️⃣ Roles & Permissions (Optional)**

* Add **Admin & Trustee roles** using Identity.
* Trustees see only their data (filter on backend using User.Identity.Name).
* Admin can see all data.

## **4️⃣ Error Handling & UI Polish**

* Add **loading spinners** while API calls are in progress.
* Add **Snackbar alerts** for errors.
* Finalize header/footer & layout.