## **📌 Available Tables**

1. **TrusteeAMLPlans**
2. **TrusteeAMLCategory**
3. **TrusteeAMLUploadData**
4. **PLAN\_INFO**

## **🔹 Table Relationships (from queries)**

* PLAN\_INFO has columns like PRODUCT\_ID, FILE\_SET, etc.
* TrusteeAMLPlans and TrusteeAMLCategory seem related to Trustee plans and AML categories.
* TrusteeAMLUploadData stores the **uploaded AML documents or data**.

## **🔹 What We Can Build with These Tables**

### **1️⃣ Trustee Dashboard (AML & Plan Info)**

* Show **Trustee Plans** (from TrusteeAMLPlans).
* Show **AML Status by Category** (from TrusteeAMLCategory).
* Show **AML Upload Data** (from TrusteeAMLUploadData).
* Show **Plan Info & Status** (from PLAN\_INFO).  
  + Filter by PRODUCT\_ID = '0020' or
  + Filter where FILE\_SET NOT IN ('FP', 'FZ').

### **2️⃣ Example Queries for Dashboard**

* **Total Plans:** sql  
  CopyEdit  
  SELECT COUNT(\*) FROM PLAN\_INFO WITH (NOLOCK)
* **Plans with PRODUCT\_ID = '0020':** sql  
  CopyEdit  
  SELECT COUNT(\*) FROM PLAN\_INFO WHERE PRODUCT\_ID = '0020'
* **AML Categories Summary:** sql  
  CopyEdit  
  SELECT CategoryName, COUNT(\*) AS Total
* FROM TrusteeAMLCategory
* GROUP BY CategoryName
* **AML Uploads (latest):** sql  
  CopyEdit  
  SELECT TrusteeId, FileName, UploadedDate
* FROM TrusteeAMLUploadData
* ORDER BY UploadedDate DESC

## **🔥 Next Step for Project**

Now we can **restructure your project** to use these real table names and create APIs:

### **Backend (Dapper)**

* GET /api/plans → data from PLAN\_INFO
* GET /api/plans/aml → data from TrusteeAMLPlans
* GET /api/plans/aml/category → data from TrusteeAMLCategory
* GET /api/plans/aml/uploads → data from TrusteeAMLUploadData

### **Frontend (React Dashboard)**

* Show cards:  
  + Total Plans
  + AML Categories Count
  + Total AML Uploads
* Add tables & filters (like FILE\_SET NOT IN ('FP','FZ'))

👉 **Do you want me to rewrite the API (Trustee/AML/SP) using these real tables now?** Or should I show you the **Dapper repository + controller code for each table** first?

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

Yes

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

Great! Let's now **rewrite the backend (API) using your real PES tables** and Dapper.

## **1️⃣ API Structure with Real Tables**

We'll create 4 main controllers:

1. **PlanInfoController** → PLAN\_INFO
2. **TrusteeAMLPlansController** → TrusteeAMLPlans
3. **TrusteeAMLCategoryController** → TrusteeAMLCategory
4. **TrusteeAMLUploadDataController** → TrusteeAMLUploadData

## **2️⃣ Repository Pattern (Dapper)**

### **Step 1: Create DapperContext**

Data/DapperContext.cs

csharp

CopyEdit

* using System.Data;
* using Microsoft.Data.SqlClient;
* namespace PES.DataPortal.API.Data
* {
* public class DapperContext
* {
* private readonly IConfiguration \_config;
* public DapperContext(IConfiguration config)
* {
* \_config = config;
* }
* public IDbConnection CreateConnection()
* => new SqlConnection(\_config.GetConnectionString("DefaultConnection"));
* }
* }

## **3️⃣ PlanInfoController**

Controllers/PlanInfoController.cs

csharp

CopyEdit

* using Dapper;
* using Microsoft.AspNetCore.Mvc;
* using PES.DataPortal.API.Data;
* namespace PES.DataPortal.API.Controllers
* {
* [ApiController]
* [Route("api/[controller]")]
* public class PlanInfoController : ControllerBase
* {
* private readonly DapperContext \_context;
* public PlanInfoController(DapperContext context)
* {
* \_context = context;
* }
* [HttpGet]
* public async Task<IActionResult> GetAllPlans()
* {
* var sql = "SELECT \* FROM PLAN\_INFO WITH (NOLOCK)";
* using var connection = \_context.CreateConnection();
* var plans = await connection.QueryAsync(sql);
* return Ok(plans);
* }
* [HttpGet("byProduct/{productId}")]
* public async Task<IActionResult> GetPlansByProduct(string productId)
* {
* var sql = "SELECT \* FROM PLAN\_INFO WHERE PRODUCT\_ID = @ProductId";
* using var connection = \_context.CreateConnection();
* var plans = await connection.QueryAsync(sql, new { ProductId = productId });
* return Ok(plans);
* }
* [HttpGet("excludeFileset")]
* public async Task<IActionResult> GetPlansExcludingFileset()
* {
* var sql = "SELECT \* FROM PLAN\_INFO WHERE FILE\_SET NOT IN ('FP', 'FZ')";
* using var connection = \_context.CreateConnection();
* var plans = await connection.QueryAsync(sql);
* return Ok(plans);
* }
* }
* }

## **4️⃣ TrusteeAMLPlansController**

Controllers/TrusteeAMLPlansController.cs

csharp

CopyEdit

* using Dapper;
* using Microsoft.AspNetCore.Mvc;
* using PES.DataPortal.API.Data;
* namespace PES.DataPortal.API.Controllers
* {
* [ApiController]
* [Route("api/[controller]")]
* public class TrusteeAMLPlansController : ControllerBase
* {
* private readonly DapperContext \_context;
* public TrusteeAMLPlansController(DapperContext context)
* {
* \_context = context;
* }
* [HttpGet]
* public async Task<IActionResult> GetAll()
* {
* var sql = "SELECT \* FROM TrusteeAMLPlans WITH (NOLOCK)";
* using var connection = \_context.CreateConnection();
* var data = await connection.QueryAsync(sql);
* return Ok(data);
* }
* }
* }

## **5️⃣ TrusteeAMLCategoryController**

Controllers/TrusteeAMLCategoryController.cs

csharp

CopyEdit

* using Dapper;
* using Microsoft.AspNetCore.Mvc;
* using PES.DataPortal.API.Data;
* namespace PES.DataPortal.API.Controllers
* {
* [ApiController]
* [Route("api/[controller]")]
* public class TrusteeAMLCategoryController : ControllerBase
* {
* private readonly DapperContext \_context;
* public TrusteeAMLCategoryController(DapperContext context)
* {
* \_context = context;
* }
* [HttpGet]
* public async Task<IActionResult> GetAllCategories()
* {
* var sql = "SELECT \* FROM TrusteeAMLCategory WITH (NOLOCK)";
* using var connection = \_context.CreateConnection();
* var categories = await connection.QueryAsync(sql);
* return Ok(categories);
* }
* }
* }

## **6️⃣ TrusteeAMLUploadDataController**

Controllers/TrusteeAMLUploadDataController.cs

csharp

CopyEdit

* using Dapper;
* using Microsoft.AspNetCore.Mvc;
* using PES.DataPortal.API.Data;
* namespace PES.DataPortal.API.Controllers
* {
* [ApiController]
* [Route("api/[controller]")]
* public class TrusteeAMLUploadDataController : ControllerBase
* {
* private readonly DapperContext \_context;
* public TrusteeAMLUploadDataController(DapperContext context)
* {
* \_context = context;
* }
* [HttpGet]
* public async Task<IActionResult> GetAllUploads()
* {
* var sql = "SELECT \* FROM TrusteeAMLUploadData WITH (NOLOCK)";
* using var connection = \_context.CreateConnection();
* var uploads = await connection.QueryAsync(sql);
* return Ok(uploads);
* }
* }
* }

## **7️⃣ Register DapperContext in Program.cs**

csharp

CopyEdit

* builder.Services.AddSingleton<DapperContext>();

## **8️⃣ Test Swagger**

Once you run the project:

bash

CopyEdit

* https://localhost:5041/swagger

You will see:

* /api/planinfo
* /api/planinfo/byProduct/{productId}
* /api/planinfo/excludeFileset
* /api/trusteeamlplans
* /api/trusteeamlcategory
* /api/trusteeamluploaddata

## **Next Step**

👉 Do you want me to now **connect these new APIs to the React Dashboard** (cards + tables) so it shows **real AML/Plan data**?  
 Or should I **add roles (Admin/Trustee) and secure these APIs** before frontend?

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

Yes

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

Perfect! Now let's **connect these new APIs to the React Dashboard** so your frontend shows **real data from PES tables**.

## **1️⃣ Dashboard Cards with Live Data**

We will connect to the new APIs:

* /api/planinfo → Total Plans
* /api/trusteeamlplans → AML Plans count
* /api/trusteeamluploaddata → Total Uploads

### **Update DashboardPage.js**

javascript

CopyEdit

* import React, { useEffect, useState } from 'react';
* import axios from 'axios';
* import { Container, Grid, Card, CardContent, Typography, Button } from '@mui/material';
* import { useDispatch } from 'react-redux';
* import { logout } from '../redux/slices/authSlice';
* import TrusteeList from '../components/TrusteeList';
* const DashboardPage = () => {
* const [counts, setCounts] = useState({ plans: 0, amlPlans: 0, uploads: 0 });
* const dispatch = useDispatch();
* useEffect(() => {
* const fetchCounts = async () => {
* try {
* const plans = await axios.get('https://localhost:5041/api/planinfo');
* const amlPlans = await axios.get('https://localhost:5041/api/trusteeamlplans');
* const uploads = await axios.get('https://localhost:5041/api/trusteeamluploaddata');
* setCounts({
* plans: plans.data.length,
* amlPlans: amlPlans.data.length,
* uploads: uploads.data.length
* });
* } catch (err) {
* console.error('Error fetching dashboard data:', err);
* }
* };
* fetchCounts();
* }, []);
* return (
* <Container>
* {/\* Header \*/}
* <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>
* {/\* Cards \*/}
* <Grid container spacing={2} sx={{ mb: 3 }}>
* <Grid item xs={12} md={4}>
* <Card>
* <CardContent>
* <Typography variant="h6">Total Plans</Typography>
* <Typography variant="h5" color="primary">{counts.plans}</Typography>
* </CardContent>
* </Card>
* </Grid>
* <Grid item xs={12} md={4}>
* <Card>
* <CardContent>
* <Typography variant="h6">AML Plans</Typography>
* <Typography variant="h5" color="secondary">{counts.amlPlans}</Typography>
* </CardContent>
* </Card>
* </Grid>
* <Grid item xs={12} md={4}>
* <Card>
* <CardContent>
* <Typography variant="h6">Total Uploads</Typography>
* <Typography variant="h5" color="error">{counts.uploads}</Typography>
* </CardContent>
* </Card>
* </Grid>
* </Grid>
* {/\* Trustee Table \*/}
* <TrusteeList />
* </Container>
* );
* };
* export default DashboardPage;

## **2️⃣ Update Trustee Table (Plan Info)**

We can display PLAN\_INFO data in TrusteeList.js:

javascript

CopyEdit

* import React, { useEffect, useState } from 'react';
* import axios from 'axios';
* import { Table, TableBody, TableCell, TableHead, TableRow, Paper } from '@mui/material';
* const TrusteeList = () => {
* const [plans, setPlans] = useState([]);
* useEffect(() => {
* axios.get('https://localhost:5041/api/planinfo')
* .then(res => setPlans(res.data))
* .catch(err => console.error(err));
* }, []);
* return (
* <Paper>
* <Table>
* <TableHead>
* <TableRow>
* <TableCell>Plan Number</TableCell>
* <TableCell>Plan Name</TableCell>
* <TableCell>Product ID</TableCell>
* <TableCell>File Set</TableCell>
* </TableRow>
* </TableHead>
* <TableBody>
* {plans.map((p, index) => (
* <TableRow key={index}>
* <TableCell>{p.PLAN\_NUM}</TableCell>
* <TableCell>{p.PLAN\_NAME1}</TableCell>
* <TableCell>{p.PRODUCT\_ID}</TableCell>
* <TableCell>{p.FILE\_SET}</TableCell>
* </TableRow>
* ))}
* </TableBody>
* </Table>
* </Paper>
* );
* };
* export default TrusteeList;

## **3️⃣ Result**

* Dashboard now shows **real counts** of plans, AML plans, and uploaded AML data.
* Trustee table displays all records from PLAN\_INFO.

## **4️⃣ Next Enhancements**

👉 Which one do you want next?

1. Add **filters** (e.g., PRODUCT\_ID = '0020' or FILE\_SET NOT IN ('FP', 'FZ')) in frontend?
2. Add **AML Categories & Upload details page (drill-down)**?
3. Add **charts (AML compliance %)** in the dashboard?
4. Or implement **role-based access (Admin vs Trustee)**?

AUTHENTICATION DONE ? JWT IDENITYT

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

**Not yet – we are using ASP.NET Core Identity but haven’t fully switched to JWT authentication.** Let me set up **Identity + JWT Token Authentication** so that your APIs are secured properly.

## **1️⃣ Why JWT with Identity?**

* **ASP.NET Identity** handles users, roles, and passwords.
* **JWT Tokens** allow your React app to authenticate API calls securely (instead of cookies).

## **2️⃣ Install JWT & Identity Packages**

In your backend project, run:

bash

CopyEdit

dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

## **3️⃣ Setup Identity with EF Core**

### **a) Create AppUser.cs**

csharp

CopyEdit

using Microsoft.AspNetCore.Identity;

public class AppUser : IdentityUser

{

// You can add custom fields if needed

}

### **b) Add DbContext**

csharp

CopyEdit

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

public class AppDbContext : IdentityDbContext<AppUser>

{

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

}

### **c) Register in Program.cs**

csharp

CopyEdit

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

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

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

.AddEntityFrameworkStores<AppDbContext>()

.AddDefaultTokenProviders();

## **4️⃣ Configure JWT Authentication**

In Program.cs:

csharp

CopyEdit

using Microsoft.IdentityModel.Tokens;

using System.Text;

var jwtKey = builder.Configuration["Jwt:Key"];

var jwtIssuer = builder.Configuration["Jwt:Issuer"];

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = "JwtBearer";

options.DefaultChallengeScheme = "JwtBearer";

})

.AddJwtBearer("JwtBearer", options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtIssuer,

ValidAudience = jwtIssuer,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey))

};

});

### **Add JWT settings to appsettings.json**

json

CopyEdit

"Jwt": {

"Key": "SuperSecretJwtKey123!",

"Issuer": "https://localhost:5041"

}

## **5️⃣ Create AuthController.cs**

This will issue tokens after login.

csharp

CopyEdit

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Identity;

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<AppUser> \_userManager;

private readonly IConfiguration \_config;

public AuthController(UserManager<AppUser> userManager, IConfiguration config)

{

\_userManager = userManager;

\_config = config;

}

[HttpPost("register")]

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

{

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

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

if (!result.Succeeded) return BadRequest(result.Errors);

return Ok("User registered");

}

[HttpPost("login")]

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

{

var user = await \_userManager.FindByEmailAsync(email);

if (user == null || !await \_userManager.CheckPasswordAsync(user, password))

return Unauthorized("Invalid credentials");

// Create JWT Token

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, user.Email),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: \_config["Jwt:Issuer"],

audience: \_config["Jwt:Issuer"],

claims: claims,

expires: DateTime.Now.AddHours(2),

signingCredentials: creds

);

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

}

}

## **6️⃣ Secure Your APIs**

In each controller (PlanInfo, TrusteeAML, etc.), add:

csharp

CopyEdit

[Authorize]

[ApiController]

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

public class PlanInfoController : ControllerBase

{

...

}

## **7️⃣ React Frontend – Store Token in Redux**

When login API returns a token, save it in Redux and attach it to each request:

javascript

CopyEdit

axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;

## **Next Step**

Do you want me to now **update your React login flow** to work with this JWT system and automatically attach tokens to API calls?  
 Or should we **add roles (Admin/Trustee) in Identity** so each user only sees their own data?