**PaymentTransaction API**

A lightweight ASP.NET Core Web API for managing payment transactions across multiple providers, supporting API key authentication, Swagger documentation, and EF Core migrations.

**Setup Instructions**

**Prerequisites**

* .NET SDK 9.0 (or 7.0/8.0)
* SQL Server or SQL Server Express
* Git
* (Optional) Visual Studio 2022+ or VS Code

**Configuration**

1. **Clone the repository**:

git clone https://github.com/your-username/payment-transaction-api.git

cd payment-transaction-api

1. **Edit appsettings.json**:

{

"ApiKey": "simple-test-api-key-12345",

"ConnectionStrings": {

"PaymentTransactionConnectionString": "Server=localhost;Database=PaymentDb;User Id=sa;Password=yourpassword;",

"PaymentAuthConnectionString": "Server=localhost;Database=PaymentAuthDb;User Id=sa;Password=yourpassword;"

}

}

1. **Install Dependencies & Tools**:

dotnet restore

dotnet tool install --global dotnet-ef

1. **Apply Migrations**:

dotnet ef database update

1. **Check Swagger in your browser**:

* Swagger UI: https://localhost:[YOURPORTNUMBER]/swagger

**Authentication**

All secured endpoints require an API key in the request headers. Use the test API key set in appsettings.json.

You can use the "Authorize" button in Swagger UI to enter your key and test endpoints.

**Unit Tests**

To run unit tests from the root folder:

dotnet test ./Tests/PaymentTransaction.Tests.csproj

**API Documentation**

**Status Endpoints**

GET: /api/status

GET: /api/status/{id} (Required: StatusId GUID)

POST: /api/status/{id} (Required: StatusId GUID)

PUT: /api/status/{id} (Required: StatusId GUID)

DELETE: /api/status/{id} (Required: StatusId GUID)

**Currency Endpoints**

GET: /api/Currency

GET: /api/Currency/{id}

POST: /api/Currency/{id}

PUT: /api/Currency/{id}

DELETE: /api/Currency/{id}

**Payment Methods Endpoints**

GET: /api/paymentmethod

GET: /api/paymentmethod/{id} (Required: PaymentMethodId GUID)

POST: /api/paymentmethod/{id} (Required: PaymentMethodId GUID)

PUT: /api/paymentmethod/{id} (Required: PaymentMethodId GUID)

DELETE: /api/paymentmethod/{id} (Required: PaymentMethodId GUID)

**Provider Endpoints**

GET: /api/providers

GET: /api/providers/{id} (Required: ProviderId GUID)

POST: /api/providers/{id} (Required: ProviderId GUID)

PUT: /api/providers/{id} (Required: ProviderId GUID)

DELETE: /api/providers/{id} (Required: ProviderId GUID)

**Transactions Endpoints**

GET: /ingest/{providerName} (Required: providerName, Idempotency-Key GUID)

POST: /api/providers/{id}

POST: /api/transaction

GET: /api/transaction/{id} (Required: Transaction ID GUID)

PUT: /api/transaction/{id} (Required: Transaction ID GUID)

DELETE: /api/transaction/{id} (Required: Transaction ID GUID)

**Summary Endpoint**

GET: /summary

Returns:

- Total number of transactions

- Total volume per provider

- Breakdown of statuses (Pending, Completed, Failed)

**Webhook Simulation**

Webhook listens for transaction-related events from providers (e.g., "payment completed", "order processed") and automatically ingests the data.

POST: /transactions/webhook/{providerName}

**Filters and Queries**

GET: /transactions

Query Parameters:

- filterOn / filterQuery (providerName or Status)

- fromDate / toDate (e.g., 2024-03-02)

- sortBy / isAscending (e.g., providerName or Status, true/false)

**Swagger & API Schema Annotations**

This project uses Swashbuckle annotations:

* [SwaggerSchema]
* [SwaggerSchemaExample]

These enhance the Swagger UI documentation by adding property descriptions and example values for all DTOs.

Example file: Models/DTO/ProviderDto.cs

**Notes on Design and Assumptions**

**Folder Structure**

PaymentTransaction/

├── Attributes/

├── Controllers/

├── CustomActionFilters/

├── Data/

├── Mappings/

├── Middleware/

├── Models/

│ ├── Configs/

│ ├── Domain/

│ ├── DTO/

├── Migrations/

├── Properties/ (launchSettings.json)

├── Repositories/

├── Validators/

├── appsettings.json

├── Program.cs

└── README.md

Tests/

├── Controllers/UnitTest.cs

**Middleware**

/Middleware/CombinedAuthMiddleware.cs: Simulates authentication by validating an API key from the request headers and assigning a claims-based identity to enable secure access without JWT or OAuth.

**Entity Models**

Defined in /Models/Domain/ and used as EF Core entities. These define table structures, relationships, and constraints. Managed via EF migrations:

* DbContext: PaymentTransactionDbContext
* Migration: dotnet ef migrations add <name>
* Apply: dotnet ef database update

**Migrations**

/Migrations/ folder contains EF Core migration files. These track changes to the schema over time.

**Mappings**

/Mappings/AutoMapperProfiles.cs defines mapping between Domain models and DTOs using AutoMapper. Ensures clean transformations between layers.

**Models**

Organized for separation of concerns:

* /Models/Domain/: Business entities (e.g., Provider)
* /Models/DTO/: Data Transfer Objects used in the API layer
* /Models/Configs/: Provider-specific configuration classes (e.g., PayPalConfig)

**Data Access**

/Data/PaymentTransactionDbContext.cs: EF Core DbContext class maps domain models to database tables and supports migrations.

**Repositories**

/Repositories/: Interfaces and implementations abstract database logic. Promotes testability and loose coupling.

Example:

* IProviderRepository
* SQLProviderRepository

**Validators**

/Validators/: FluentValidation classes for DTOs. (Note: DataAnnotations also used for basic field validation.)

**Custom Action Filters**

/CustomActionFilters/: Reusable logic for model validation and idempotency enforcement.

**Controllers**

/Controllers/: Define API endpoints and use dependency injection, AutoMapper, and repositories for request handling and responses.

**Attributes**

/Attributes/: Custom attributes and Swagger filters. Used to enhance documentation and enforce header requirements (e.g., Idempotency-Key).

**Idempotency-Key Implementation**

Implemented to prevent duplicate transaction insertions from repeated requests.

* **Header Enforcement**: [RequiresIdempotencyKeyHeader] on endpoints like /ingest/{providerName} and /webhook/{providerName}
* **Action Filter**: IdempotencyFilter:
  + Checks if transaction with key exists
  + Returns existing or inserts new
* **Swagger Enhancement**: AddIdempotencyKeyHeaderParameter adds header info to Swagger docs automatically

**Dependency Injection (DI)**

Uses ASP.NET Core DI framework to inject:

* Repositories (ITransactionRepository, IProviderRepository)
* DbContext (PaymentTransactionDbContext)
* AutoMapper

Services are registered in Program.cs and injected via constructor.

**Notes on Design and Assumptions**

* Modular structure for scalability and clarity
* Repository pattern for business logic separation
* AutoMapper for DTO-model decoupling
* EF Core handles schema evolution with migrations
* Custom filters handle validation and idempotency
* Swagger for complete API discovery and testing
* [DataAnnotations] for model validation

**Assumptions:**

* Safe retries are needed for ingestion endpoints (especially with webhooks)
* API will expand to support more providers
* Swagger docs are a key consumer-facing requirement