Payroll Engine Backend

👉 This application is part of the Payroll Engine.

Open API

The Payroll Engine API supports the Open API specification and describes the interface to the Swagger tool. The document REST Service Endpoints document describes the available endpoints.

> Payroll Engine swagger.json

API Versioning

In the first 1.0 release of the REST API, no version header is required in the HTTP request. For future version changes, the HTTP header **X-Version** with the version number must be present.

API Content Type

The Payroll REST API supports HTTP requests in `JSON` format.

Backend Server

In order to run the backend server, the web host must support the execution of .NET Core applications. Follow these steps to start the IIS Express service for local development:

- Dotnet using the binary file:

```
dotnet <PathToBin>/PayrollEngine.Backend.Server.dll --
urls=https://localhost:44354/
```

- Dotnet using the project file, using the working path `Backend.Server/`:

```
dotnet run --urls=https://localhost:44354/
```

- Visual Studio solution `PayrollEngine.Backend.sln` using the debugger.

Application Settings

The server configuration file `appsetings.json` contains the following settings:

Setting	Description	Type	Default
`StartupCulture`	The culture of the backend process	string	System culture
`AuditTrailDisabled`	Disable the audit trail for regulation objects	bool	false

Setting	Description	Type	Default
`LogHttpRequests`	Log http requested to log file	bool	false
`InitializeScriptCompiler`	Initialize the script compiler to reduce startup time	bool	false
`DumpCompilerSources`	Store compiler source files [1]	bool	false
`DbTransactionTimeout`	Database transaction timeout	timespan	10 minutes
`DbCommandTimeout`	Database command timeout	seconds	2 minutes
`WebhookTimeout`	Webhook timeout	timespan	1 minute
`FunctionLogTimeout`	Timeout for tracking long function executions	timespan	off
`AssemblyCacheTimeout`	Timeout for cached assemblies	timespan	30 minutes
`VisibleControllers`	Name of visible API controllers [2] [3]	string[]	all
`HiddenControllers`	Name of hidden API controllers [2] [3]	string[]	none
`DarkTheme`	Use swagger dark theme	bool	false
`ApiKey`	Enable api key protection, dev-secret only!	string	none
`Serilog`	Logger settings	Serilog	file and console log

[1] Store compilation scripts the disk. Analyses only feature.

[2] Wildcard support for `*` and `?`.

[3] `HiddenControllers` setting cannot be combined with `VisibleControllers` setting.

> It is recommended that you save the application settings within your local User Secrets.

Database connection string

The backed database connection string is determined by the following priority:

1. Environment variable `PayrollDatabaseConnection`.
2. Program configuration file `appsettings.json`.

Application Logs

The backend server stores its logs in the application folder `logs`.

Api Key

Once set, the API key is the only way to access the API endpoints. The API client must send it in the `Api-Key` request header.

The API key is defined in the following places (in order of priority):

1. System environment variable `PayrollApiKey`
2. Value `ApiKey` in the application settings file `appsettings.json`

When an endpoint request is made, the API key must be included in the `Api-Key` HTTP header.

> When the API key is active, Swagger requires authorization from it.

C# Script Compiler

The business logic defined by the business in C# is compiled into binary files (assemblies) by the backend using Roslyn. This procedure has a positive effect on the runtime performance, so that even extensive calculations can be performed sufficiently quickly. At runtime, the backend keeps the assemblies in a cache. To optimize memory usage, unused assemblies are periodically deleted (application setting `AssemblyCacheTimeout`).

You can use the 'InitializeScriptCompiler' application setting to start the Roslyn engine when the application starts, thereby eliminating the runtime delay.

To perform a more in-depth analysis, set the `DumpCompilerSources` application setting to force the C# script compiler to save the source scripts of the compilation as disk files. These files are stored in the `ScriptDump` folder within the application folder, ordered by function type and dump date.

Solution projects

The.NET Core application consists of the following projects:

Name	**Type**	**Description**
`PayrollEngine.Domain.Model`	Library	Domain objects and repositories
`PayrollEngine.Domain.Scripting`	Library	Scripting services
`PayrollEngine.Domain.Application`	Library	Application service
`PayrollEngine.Persistence`	Library	Repository implementations
`PayrollEngine.Persistence.SqlServer`	Library	SQL Server implementation
`PayrollEngine.Api.Model`	Library	Rest objects
`PayrollEngine.Api.Core`	Library	Rest core services
`PayrollEngine.Api.Map`	Library	Mapping between rest and domain objects
`PayrollEngine.Api.Controller`	Library	Rest controllers
`PayrollEngine.Backend.Controller`	Library	Routing controllers
`PayrollEngine.Backend.Server`	Exe	Web application server with rest api

Docker Support

Build the Docker image:

```
docker build -t payroll-backend .
```

Run with database connection:

```
docker run -p 5000:5000 \
  -e
ConnectionStrings__DefaultConnection="Server=localhost;Database=PayrollEngine;User
Id=sa;Password=PayrollStrongPass789;TrustServerCertificate=True;" \
  payroll-backend
```

Verify API is accessible at http://localhost:5000

Further documents

- OData queries
- Database Management
- Developer Guidelines

Third party components

- Object mapping with Mapperly - license `Apache 2.0`
- OpenAPI with Swashbuggle - license `MIT`
- Database query builder with SqlKata - license `MIT`
- Database object mapping with Dapper - license `Apache 2.0`
- Logging with Serilog - license `Apache 2.0`
- Tests with xunit - license `Apache 2.0`