# System documentation – Self-evaluation, version 1.0

## Source code

The source code can be found at DIFI’s Github, via this URL: <https://github.com/difi/tilsyn-automat>

## Library/Components

The website is developed in ASP.NET Core 2.1

* .NET Core 2.1
* ASP.NET Razor pages.
* EntityFrameworkCore 2.1
* EPPlus (used to create Excel files)
* Swashbuckle (used for API documentation)
* WindowsAzure.Storage (used for image uploads)

## Description of the code

The system consists of six different code projects and two linkages to other systems (IDPorten and Azure) .

### Code projects: API

This is an API layer, which is used in order to transmit data between the website and the database. It can also be used as a stand-alone API. The API is documented by using ”Swagger/Swash” and the documentation can be accessed via <https://[URL]/swagger/index.html>. The API always returns an object of the ApiResult type, which contains Suceeded (yes/no), possible ID of the post in question, potential errors and finally Data, which is the requested object (for example List<UserItem>).

In order to access the API, an API key is needed. It is located in secrets.iso.

"Api": {   
 "Key": "{RandomString}"   
 "IpList": "{Comma-separated list}"  
}

### Code project: Database

This project contains all database linkages. The database linkage is made via EntityFrameworkCore.   
The project consists of to database layers, on for the entire application and one for logging purposes.

The project uses code first, and and all database tables are created and updated via classes in shared, which then are updated in the database via Powershell script. These scripts can be found in wwwroot/Migrations.

In order to create an update script, run the command:   
Add-Migration -context ApplicationDbContext Vx (where X is a serial number)

In order to run the update of the database with all new scripts, run the command:  
Update-Database -context ApplicationDbContext

In order to run the website (both locally and in Azure) a connection string in the application options is required. We refer you to the chapter “Options” in order for you to see what that database connection string looks like.

The tables in the database are decribed in the “Database” chapter.

### Code project: Cache

This project contains a cache layer for the more demanding database linkages (operations and self-evaluations). This cache lies between the API and the database linkage by “Decoration”, which is configured in ConfigureServices in Startup.cs.

### Code project: Log

This project contains a logging layerfor all database linkages. This cache lies between the API and the database coupling by “Decoration”, which is configured in ConfigureServices in Startup.cs. The contents of the log is configured in the application options (for example secrets.iso). The options consist of this:

"Log": {

"Active": true, (bestämmer om loggning ska vara påslaget eller inte)

"LogGetSucceeded": false, (bestämmer om hämtningar av data ska loggas)

"LogChangeSucceeded": true, (bestämmer om ändringar av databasen ska loggas)

"LogError": true, (bestämmer om fel ska loggas)

"LogLongTime": 500 (bestämmer om allt som tar längre tid än X ms ska loggas)

},

### Code project: Shared

All objects (classes) whicha are shared between the API, Database, Cache, Log och wwwroot, such as an user object (UserItem). These objects are described in the API documentation as well, including what is required and what is not. Shared also contains some shared code, attributes and lists. Finally, shared contains seven “Interfaces” which the database, log and cache implement.

### Code projectt: Wwwroot

This is the website itself. It consists of different parts, as described below.

* Wwwroot
  + Contains all static files, such as js, css and images.
* Business
  + Contains logic needed for the entire website, such as API requests, error management and Excel management.
* Migrations
  + Contains scripts used to create and update the database via Powershell.
* Pages
  + Contains all pages and parts of pages
  + Admin
    - This is the directory where all pages for editors and administrators are found.
  + Declaration
    - This is the directory where all pages for activities/self-evaluation are found.
  + Shared
    - Contains shared views, such as form objects, error management, etc.
* Resources
  + Contains all translations for buttons, headings, etc.

## Linkages

### ID Porten

API documentation: <https://difi.github.io/idporten-oidc-dokumentasjon/>  
Contact: [IDporten@difi.no](mailto:IDporten@difi.no)  
Options:

"IdPorten": {

"BaseUrl": "https://oidc-ver2.difi.no/idporten-oidc-provider",  
 (URL to ID portens environment)

"ClientId": "{ClientId}",  
 (Decided together with IDPorten)

"Secret": "{Secret}",  
 (Secret key from IDPorten)

"Nonce": "{RandomString}",

(A string which is nestled into the request in order to ensure a more secure control of requests)

"RedirectUrl": "https://localhost:44343/LoginHandler",

(URL used to handle log-in tokens)

"LogoutUrl": "https://localhost:44343/LogoutHandler"

(URL for log-out)

},

Token: handle via JWT (JSON Web Tokens) - <https://jwt.io/>

### Azure

Azure is used in order to store all uploaded images. This applies even if the website is run locally. In order for this to work, we use Fine Uploader 5.16.2 Azure version, but a number of custom adaptations are marked in the code with //FUNKA Custom code start and //FUNKA Custom code end.

In order to make the upload work, settings must be made in secrets.iso.

"Azure": {

"StorageAccountName": "{StorageAccountName}",

(This is a stoarage account created in Azure)

"StorageAccountKey": "{StorageAccountKey}",

(Key created by Azure)

"StorageContainer": " {StorageContainer}"

(Directory where the files are stored, should be a different one for each environment)

},

## Settings

This is a complete list over all configurations that are required in order for the system to work. They are described in their respective chapters.

{­­

"ConnectionStrings": { "DefaultConnection": "Server={Server};Database={Database};User ID={User};Password={Password};MultipleActiveResultSets=true" },

"IdPorten": {

"BaseUrl": "https://oidc-ver2.difi.no/idporten-oidc-provider",

"ClientId": "{ClientId}",

"Secret": "{Secret}",

"Nonce": "{RandomString}",

"RedirectUrl": "https://localhost:44343/LoginHandler",

"LogoutUrl": "https://localhost:44343/LogoutHandler"

},

"Azure": {

"StorageAccountName": "{StorageAccountName}",

"StorageAccountKey": "{StorageAccountKey}",

"StorageContainer": " {StorageContainer}"

},

"Log": {

"Active": true,

"LogGetSucceeded": false,

"LogChangeSucceeded": true,

"LogError": true,

"LogLongTime": 500

},

"Api": {  
 "Key": "{RandomString}"  
 "IpList": "{Comma-separated list }"  
 }

}

## Drive documentation

### Local development environment

URL: <https://localhost:44343/>

The code is located in DIFI’s Github and can be run locally against a shared SQL server or one of your own. Azure blog storage is used for image storing.

Webbserver: IIS Express  
Database server: MS SQL server 2016 (version 11)

### Development environment

URL: <https://sjalvdeklaration-test.azurewebsites.net/>

This environment can be found in Azure as a web application (app service), Azure DB and Azure blog storage.

### External acceptance test environment

URL: <https://egenkontroll-test.azurewebsites.net/>

This environment can be found in Azure as a web application (app service), Azure DB and Azure blog storage.

### Production environment

This environment can be found in Azure as a web application (app service), Azure DB and Azure blog storage.

### Deployment to production environment and testing environment

In order to deploy Azure, you use a publication file which is downloaded in the Azure portal under the application service you want to deploy to. You then import this to Visual Studio and select “Publish” at wwwroot projektet.