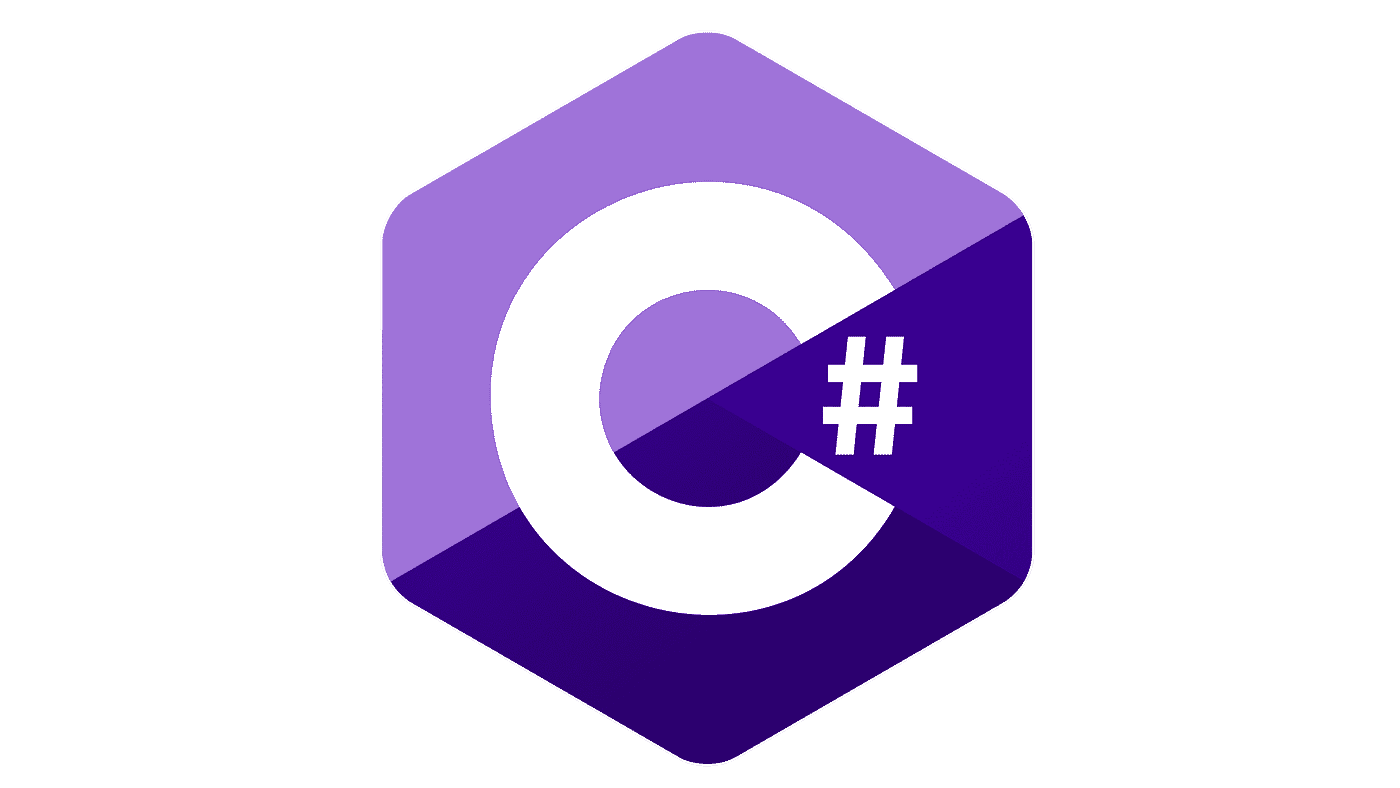


Technical

Documentation

DTM

Document Trace Manager

Made by: Serena Kenter

Date: 18-01-2024

[1. Introduction 3](#_Toc2133852686)

[2. Installation 4](#_Toc1511701651)

[2.1. Vue.js 5](#_Toc987030834)

[2.2. .NET 5](#_Toc618187092)

[2.3. Run project 5](#_Toc903272149)

[3. Code Structure and functions 6](#_Toc794281440)

[4. Tools 8](#_Toc942883975)

[4.1. Frameworks/Languages 9](#_Toc2070661336)

[4.2. Libraries 9](#_Toc1450187453)

[4.3. Packages 9](#_Toc1934316042)

[4.4. Database management tool 9](#_Toc1093453801)

[4.5. Comments 10](#_Toc1854099331)

# 1. Introduction

The original assignment arose in response to the organization's growth, revealing various issues. The primary concern was the inefficiencies stemming from manual processes, particularly in managing a substantial volume of documents such as passports, contracts, and user agreements. These processes consumed significant time and often led to errors and overlooked tasks, posing obstacles to smooth operations and potential legal and operational complications.

The impetus for the assignment arose from the urgent need to address these issues and enhance operational efficiency. The ultimate goal was to develop a customized application specifically focused on generating alerts as the expiration dates of critical documents approached. The aim was to timely inform relevant employees about upcoming expiration dates, enabling them to take appropriate measures to maintain the validity of these documents.

The initial scope of the assignment included developing the document management application and the alert generation feature. However, beyond the original scope of the assignment, additional functionalities related to employee overviews, loan overviews, and linking employees to borrowed items, including the history of these transactions, were added.

The additional requirements for employee and loan management expanded the scope of the assignment. Now, the application had to support not only document management but also provide complex functionalities for tracking employees, their borrowed items, and the history of these interactions.

The intended outcome is a comprehensive web application that meets the original needs of document management and alert generation while also fulfilling the new requirements for employee and loan management. This is expected to significantly enhance operational efficiency and provide a comprehensive solution to the challenges arising from the organization's growth. The choice of a web application ensures broad accessibility, flexible scalability, and sustainability, meeting the expectations of all stakeholders within the organization.

For this program, I used .NET 7 and Vue.js. These tech tools make our app solid and responsive. .NET 7 manages data and business logic well. Vue.js crafts dynamic and user-friendly interfaces, ensuring a smooth experience. This tech combo makes our app modern, effective, and user-friendly.

# 2. Installation

2.1. Vue.js  
**Node.js and npm**  
Vue.js relies on Node.js and npm (Node Package Manager). Install Node.js from https://nodejs.org/, which includes npm.  
  
**Vue CLI**  
Install Vue CLI globally using npm: npm install -g @vue/cli  
  
**Project Dependencies**  
Install axios by inserting the following command [npm i axios].  
Install moment by inserting the following command [npm i vue-moment].  
Install JWT decode by inserting the following command [npm i vue-jwt-decode].

2.2. .NET  
**Visual Studio (or Visual Studio Code)**  
For C# development, you typically use Visual Studio. Install Visual Studio from https://visualstudio.microsoft.com/. Alternatively, you can use rider from <https://www.jetbrains.com/rider/download/#section=windows>**.NET SDK**  
Install the .NET 7 x64 SDK from <https://dotnet.microsoft.com/en-us/download/visual-studio-sdks>.  
  
**C# Language Support**  
Ensure that your development environment has proper C# language support. Visual Studio usually includes this by default  
  
**Database SQL server**  
Install SQL server management studio from <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>.  
  
**CORS Configuration**  
This application uses CORS (Cross origin resource sharing). Make sure to check that in the back-end file Program.cs in the function AddCors() you set WithOrigins to the right url -> .WithOrigins("URL of hosted front-end")  
  
Then also do this for the front-end file axios-auth.js. Change the baseURL to the one you use.  
baseURL: URL of the hosted back end'

2.3. Run project  
**Run the Back end**  
Start your C# back-end application. If you're using Visual Studio, build and run your project or if you are using the terminal use the command [dotnet run].

**Run the Front end**  
Start your Vue.js Front end application by using the 2 commands:  
1. npm install  
2. npm run dev

# 3. Code Structure and functions

**Load employees from GAL**  
Let’s start with the functionality to load the employees from the global adress list. You can find this at BLL -> Services -> Background ->LoadEmployeeService.cs. The function GetEmployeesFromGAL() retreives al information of the gal so I can use it in my app. This is done through a ms azure graph api. In the following steps I will explain how I set this up so you can do the same.  
  
1. Go to <https://portal.azure.com/#home> and search for Microsoft entra ID.

2. In the left sidebar click on App registrations. And click on your app or create a new one.

3. Now copy the client ID. In the Back-end file appsettings.json you can past this value at “AzureAd”, also past this value at Service -> Authentication -> AzureAuthProvider.cs. Now do the same for the tenant ID, but it should look like this in the AzureAuthProvider:   
 Authority = "https://login.microsoftonline.com/[tenantID]"

4. Then go to Microsoft entra ID -> App registrations -> your app -> certificates and secrets. Now create a new client secret. Set a description and expirationdate. Press add, copy the value and past it in the AzureAuthProvider.cs.

5. Finally we need to add the api permissions to have access to the GAL information. Go to Microsoft entra ID -> App registrations -> your app -> Api permissions. Then click on add a permission -> microsoft graph then add Directory.Read.All and User.Read.All for both Delegated permissions and Application permissions. You might need consent from the azure administrator. This can be done through the button Grant admin consent for 4-rest.

The client secret has an expiration date, so step 4 must be done again when it expires, or the application won’t stop working, but the functionality will.

**Send notification**  
The main assignment was to send emails to the administration to know when documents like contracts and passports are expiring. I have done this in BLL -> Services -> Background -> ExpirationCheckService.cs in the function ProcessExpiringDocumentsAsync(). It checks every day if there are documents 5 or 6 weeks before expiring. If there is a document about to expire it sends an email with the information of that document and the employee which is connected to it. Further code for sending that email is covered in the MailService.cs.

**Delete Product**  
In the productOverview you can delete products, but to make sure you don’t accidentally delete it I add an archive which deletes the products after 90 days. This functionality is also written in the ExpirationCheckService.cs. The function ProcessDeletedProductsAsync() checks if the field TimeDeleted is equal to 90 days and then it will be deleted permanently.

**ValidationHelper**  
BLL -> Services -> Background -> ValidationService.cs checks annotation errors. In the models I have set the [stringLength] and [required] which can cause an error. This function catches this mistake and makes sure to throw a clear exception.

**Error handling**  
I have chosen for an ExceptionHandlingMalware which consists of all exceptions I used. This class is called in the program.cs while building, so that if an exception is thrown, this class catches the exception and returns the right message.

**JWT token**  
For security I have inserted a jwt token which you can find at BLL -> Services -> Authentication -> JwtValidationService.cs. This token is retreived in the front-end by axios.defaults.headers.common['Authorization'] = "Bearer " + res.data;  
And is then set into the localstorage so you can retreive it from any file.

The purpose of this token is to not let unauthorized access into your application. To do so you must add this token to the header in every single API call. Then in the back end we can check if the jwt token is correct or not. I did this by using the [authorize] annotation. This AutorizeAttribute class makes sure to check every API call on a valid token. If it doesn’t have a valid token in the header, it returns an exception, and no access is allowed.

**Migrations**  
Changes made to your models, such as adding a field or removing a model, need to be migrated. Migrations are the .NET way of implementing these changes in your database schema, and they are typically generated automatically. However, it's essential to understand when to create a migration.

For a more in-depth understanding of migrations, I recommend reading the documentation on this site:  
<https://learn.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>

**CORS**  
I enabled cors for some extra security in the application. This code is mostly located in the program.cs. The method AddCors() makes sure that only the front-end application with that specific URL can use this back end. At the bottom you can see I added app.UseCors("ApiCorsPolicy"); Which means you can use cors by adding [EnableCors("ApiCorsPolicy")] above every single API Controller.

# 4. Tools

## 4.1. Frameworks/Languages

* .NET 7
* Vue.js 10.3.0
* HTML
* CSS

## 4.2. Libraries

* XUnit test (Tests)
* Class library (BLL)
* Class library (DAL)
* ASP.NET Core Web API (PL)

## 4.3. Packages

* Microsoft.AspNetCore.Authentication.JwtBearer
* Microsoft.AspNetCore.Mvc.NewtonsoftJson
* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.Design
* Microsoft.EntityFrameworkCore.InMemory
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.Extensions.Hosting.Abstractions
* Microsoft.Graph
* Microsoft.NET.Test.Sdk
* Microsoft.OpenApi
* Coverlet.collector
* Xunit
* Xunit.runner.visualstudio
* MailKit
* MimeKit
* Moq
* Swashbuckle.AspNetCore.SwaggerGen
* Swashbuckle.AspNetCore.SwaggerUI

4.4. Database management tool  
The application uses a SQL server management studio database. To connect to the database, you need a ConnectionString. The ConnectionString currently used in the appsettings.json is connected to an azure database called DTM. This ConnectionString is added in the program.cs in the function AddDbconnection(). This function creates the connection by retreiving the connectionString from the appsettings.json.

If you choose to create a new database, a new ConnectionString is required.

4.5. Comments  
In the application itself, there are comments that provide explanations about a piece of code. Comments in .NET are indicated with 2 or 3 /. For example: ///this is a comment.