**Backend Documentation**

Prepared for

**Intellicode**

VERSION 1.0

DATE: 18th Oct, 2021

Prepared by

HazelSoft

Contents

[1. Introduction 3](#_Toc85477715)

[2. Setting up development environment locally 3](#_Toc85477716)

[2.1 Pre-Requisites 3](#_Toc85477717)

[2.2 Download Source Code 3](#_Toc85477718)

[2.3 Setting up Database 3](#_Toc85477719)

[2.4 Setting Environment Variables 3](#_Toc85477720)

[2.4.1 The launchSettings.json file: 4](#_Toc85477721)

[2.4.2 appsettings.json 4](#_Toc85477722)

[2.4.3 Important Settings 4](#_Toc85477723)

[3. Used Frameworks and Libraries 5](#_Toc85477724)

[4. Code Structure 5](#_Toc85477725)

[4.1 API 5](#_Toc85477726)

[4.1.1 Intelli.AgentPortal.Api 5](#_Toc85477727)

[4.2 Domain 6](#_Toc85477728)

[4.3 Infrastructure 6](#_Toc85477729)

[4.4 Shared 7](#_Toc85477730)

[4.5 Solution Items 7](#_Toc85477731)

[4.6 Swagger Documentation 7](#_Toc85477732)

[4.6.1 Add and configure Swagger middleware 7](#_Toc85477733)

[4.6.2 Customize the UI 8](#_Toc85477734)

[5. Deployment 9](#_Toc85477735)

[5.1 Pre-Requisites 9](#_Toc85477736)

[5.1.1 Install Latest SQL Server Express on Server 9](#_Toc85477737)

[5.1.2 Install Dot Net 5 Runtime on windows server 9](#_Toc85477738)

[5.1.3 Install RabbitMQ 9](#_Toc85477739)

[5.1.4 Enable IIS on Server 9](#_Toc85477740)

[5.2 Update source code on server 9](#_Toc85477741)

[5.3 Deploy databases 10](#_Toc85477742)

[5.4 Create production configuration file 10](#_Toc85477743)

[5.5 Publish AgentPortal 10](#_Toc85477744)

[5.6 Create the IIS Site 10](#_Toc85477745)

[6. Technology Stack 11](#_Toc85477746)

[6.1 Entity Framework 11](#_Toc85477747)

[6.2 Database First Approach 12](#_Toc85477748)

[6.2.1 Creating database 12](#_Toc85477749)

[6.2.2 Requirements to use Repository Pattern for new models 12](#_Toc85477750)

[6.2.3 Seeding screen and its elements in database. 14](#_Toc85477751)

[7. Features 15](#_Toc85477752)

[Following are the main features of backend and explanation that how they are implemented: 15](#_Toc85477753)

[7.1 User & Role Management 15](#_Toc85477754)

[7.2 Users Per Company 16](#_Toc85477755)

[7.3 Password Management 17](#_Toc85477756)

[7.4 Audit 18](#_Toc85477757)

[7.5 Logging mechanism 19](#_Toc85477758)

# Introduction

This document outlines the architecture for the backend and tools and technologies used to support it. We would also cover code structure for each module in the project and also the explanation which help developers to setup the development environment locally.

# Setting up development environment locally

## Pre-Requisites

* Microsoft Visual Studio Community 2019 or higher with web development workload.
* SQL Server 2019 Express or higher.
* SQL Server Management Studio v18.8 or higher.
* RabbitMQ.
* RabbitMQ management portal.

## Download Source Code

Download source from git repository (<https://github.com/INTELLISOLUTION/AgentPortal.git>) and open solution file in Microsoft Visual Studio and wait for the required packages to be downloaded.

## Setting up Database

* Make sure your SQL Server is running and can be connected using SQL Server Management Studio with default Server Name: [PC Name]\SQLEXPRESS
* Run the scripts, in sequence, located inside Solution => Solution Items => Migrations Folder to create database.
* Run the script located inside Solution => Solution Items => Migrations => Audit Folder, “Audit DbSetup” to setup audit database.

## Setting Environment Variables

ASP.NET Core configures app behavior based on the runtime environment using an environment variable.

* DOTNET\_ENVIRONMENT
* ASPNETCORE\_ENVIRONMENT

The development environment can enable features that shouldn't be exposed in production. For example, the ASP.NET Core templates enable the [Developer Exception Page](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-5.0#developer-exception-page) in the development environment.

The environment for local machine development can be set in the Properties\launchSettings.json file of the project. Environment values set in launchSettings.json override values set in the system environment.

### The launchSettings.json file:

* Is only used on the local development machine.
* Is not deployed.
* contains profile settings

### appsettings.json

The default JsonConfigurationProvider loads configuration in the following order:

* appsettings.json
* appsettings.Environment.json : For example, the appsettings.Production.json and appsettings.Development.json files.
* The environment version of the file is loaded based on the IHostingEnvironment.EnvironmentName

appsettings.Environment.json values override keys in appsettings.json. For example, by default: In development, appsettings.Development.json configuration overwrites values found in appsettings.json.

In production, appsettings.Production.json configuration overwrites values found in appsettings.json.

### Important Settings

* The ConnectionStrings setting should be changed as per your sql server settings.
* The EmailConfirmationCallBackUrl setting should be changed to url of the host in order to build valid url link for email confirmation and password reset.
* The Jwt:SecretKey setting should be changed. Unique and valid random character string of any length is applicable for this setting value.
* The SendEmailSettings section of configuration should be changed to use your own Send grid provided account settings.

# Used Frameworks and Libraries

Following are the frameworks / libraries are used for Common Foundation Backend. All of these are easily available over “nuget” repository

|  |  |
| --- | --- |
| Framework/Library Name | Purpose |
| .NET Core | Main runtime for the whole project |
| EF Core | ORM for database. |
| AutoMapper | For object to object mapping (if needed) |
| NLog | Server side logging |
| SwashBukle (optional) | For documentation with swagger |
| RabbitMQ | For event Publishing and subscribing |

# Code Structure

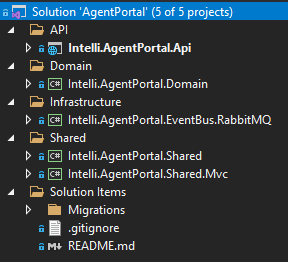


Figure 1

## API

API solution folder has following Api Project.

### Intelli.AgentPortal.Api

This is the startup project and responsible for providing all the services to the client.

This project is further divided into following sections:

#### Constants

Classes with constants can be defined here, e.g., IAuthConstants.cs here define constants to read JWT settings from configuration.

#### Controllers

Controllers are further divided into categories, Audit, Authentication, Authorization, Company, Configuration, Meta, Reports.

#### DTO

Dto contains all the classes used to encapsulate and transfer model data.

#### Events

Audit event handler and mapping factory classes which are used to handle audit events

#### Helpers

Helper classes are used to provide support for controller and services

#### Model

User login result and User registration result models are created here. User login result is essentialy same as user registration result.

#### Services

Services which are being used by controllers are implemented here. This folder contains Auth, Cache, Email, JWT, Privileges & Session services interfaces and their implementation in their corresponding folders.

## Domain

Domain contains domain library, which has database contexts, model classes and generic repository implementation. Database models are accessed via repository pattern by encapsulating each model inside a repository.

## Infrastructure

Infrastructure contains event bus implementation based on RabbitMQ. It contains classes and interfaces that are used by repositories to que messages in event bus along with event listeners and consumers that will read respective qued message from the que and handle requests by using implmented handlers.

## Shared

It contains two libraries that are being used by API and if more APIs are added these libraries are used to get unified requests from clients and return standard predefined response to clients. This ensures seamless integrity with clients.

## Solution Items

Solution items contain migration scripts which are essentialy sql scripts. These scripts are added in a way that the queries in a script runs as transaction and on successfull run logs its entry in migration history to avoid duplication. Each subsequent change in database should be added in migration script on same pattern to provide proper versioning across database setups on staging, production and any related deployments.

## Swagger Documentation

Swagger (OpenAPI) is a language-agnostic specification for describing REST APIs. It allows both computers and humans to understand the capabilities of a REST API without direct access to the source code. Its main goals are to:

* Minimize the amount of work needed to connect decoupled services.
* Reduce the amount of time needed to accurately document a service.

We used Swashbackle OpenApi implementation for .Net in our Project.

Swagger UI offers a web-based UI (Tab in your browser) that provides information about the service, using the generated OpenAPI specification. Swashbuckle include an embedded version of Swagger UI, so that it can be hosted in your ASP.NET Core app using a middleware registration call. Each public action method in your controllers can be tested from the UI.

### Add and configure Swagger middleware

|  |  |
| --- | --- |
| \_ = services.AddSwaggerGen(c =>  {  // configure SwaggerDoc and others  c.SwaggerDoc("v1", new OpenApiInfo { Title = AssemblyName, Version = "v1" });  // add JWT Authentication  var securityScheme = new OpenApiSecurityScheme  {  Name = "JWT Authentication",  Description = "Enter JWT Bearer token \*\*\_only\_\*\*",  In = ParameterLocation.Header,  Type = SecuritySchemeType.Http,  Scheme = "bearer", // must be lower case  BearerFormat = "JWT",  Reference = new OpenApiReference  {  Id = JwtBearerDefaults.AuthenticationScheme,  Type = ReferenceType.SecurityScheme  }  };  c.AddSecurityDefinition(securityScheme.Reference.Id, securityScheme);  c.AddSecurityRequirement(new OpenApiSecurityRequirement  {  {securityScheme, Array.Empty<string>()}  });  }); | |
|  | |

### Customize the UI

Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.), specifying the Swagger JSON endpoint.

|  |
| --- |
| // Use Swagger  app.UseSwagger();  app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", AssemblyName)); |
|  |

# 

# Deployment

The Deployment Process of the backend is as follows:

## Pre-Requisites

### Install Latest SQL Server Express on Server

For database deployment install SQL Server express or higher on server machine, you can also install SQL Server Management studio to manager your databases.

<https://www.microsoft.com/en-us/sql-server/sql-server-downloads>

### Install Dot Net 5 Runtime on windows server

Download and install the Dot Net Runtime 5.0 (Hosting bundle installer for Windows) into your hosting windows server from following link:

<https://dotnet.microsoft.com/download/dotnet/5.0>

### Install RabbitMQ

Download and install RabbitMQ from the following link:

<https://www.rabbitmq.com/download.html>

To enable RabbitMQ management plugin (optional):

<https://www.rabbitmq.com/management.html>

### Enable IIS on Server

Another thing that have to be enabled in your server, is the Web IIS feature. You might already have this enabled in your hosting server, however, you have to verify that it has been installed there.

From your Windows Server , Open Server Manager, then IIS, then Manage and select ‘Add Roles and Features’, then go to features, then see if the Web IIS checkbox is enabled, if not, then proceed with installing it.

## Update source code on server

Create a folder on server and download source code into the directory for instance from git repository.

E.g. create folder C:\Backend\AgentPortal

## Deploy databases

Deploy database by running scripts in migration folder.

For Audit database run script in Migration / Audit folder.

## Create production configuration file

In Api folder copy appsettings.Development.json file as appsettings.Production.json and edit as per server requirements.

E.g. connection string on server might be different or you may want to use different encryption key and email sending settings on server etc.

## Publish AgentPortal

For next step open command prompt and run following command:

dotnet publish -c Release --output C:\inetpub\wwwroot\Intellicode\Intelli.AgentPortal.Api

## Create the IIS Site

* In IIS Manager, open the server's node in the Connections panel. Right-click the Sites folder. Select Add Website from the contextual menu.
* Provide a Site name and set the Physical path to the app's deployment folder that you created in previous step e.g.

C:\inetpub\wwwroot\Intellicode\Intelli.AgentPortal.Api.

* Provide the Binding configuration and create the website by selecting OK.
* For subsequent deployments IIS needs to be stopped and then started again after deployment.

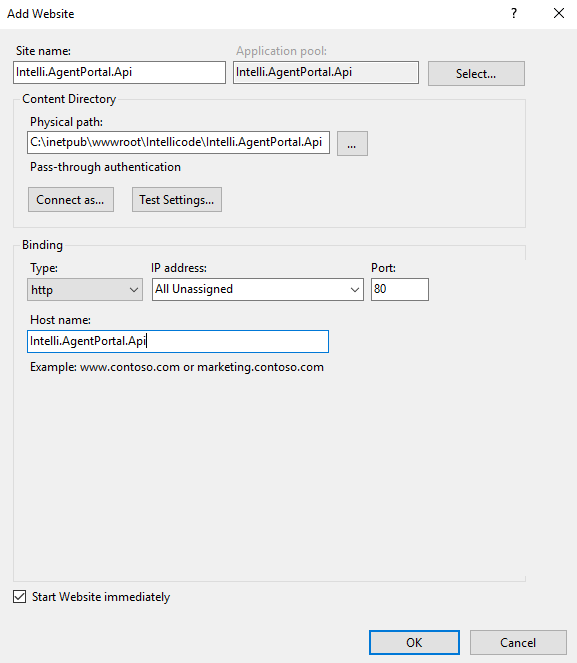


Figure 2

# Technology Stack

Following technologies are used to develop backend api:

* ASP.NET CORE Web API
* Entity Framework (Database First)
* SQL Server

## Entity Framework

Entity Framework is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.

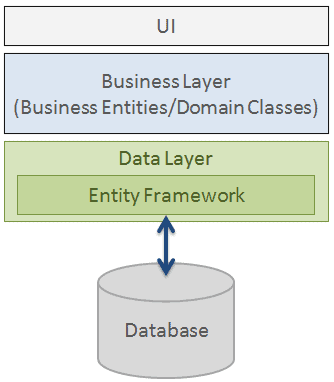


Figure 3

## Database First Approach

### Creating database

Entity Framework is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.

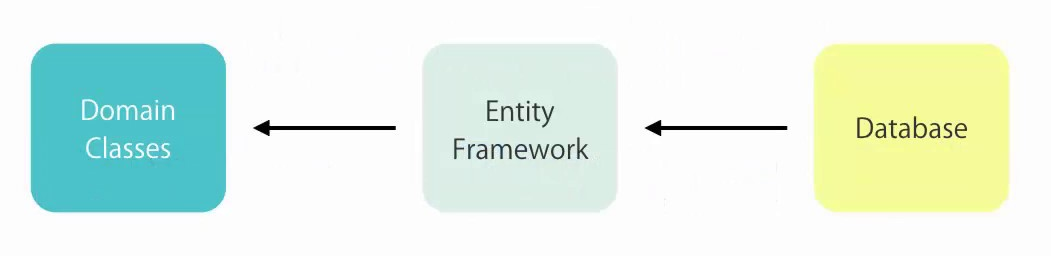


Figure 4

* To create database, run sql scripts which are in Solution Items / Migration folder.

### Requirements to use Repository Pattern for new models

In order to use repository pattern for new models, following points needs to be considered:

#### Before Scaffolding Models

* Database table should have Id of type int as its primary key
* IsActive column of type boolean
* CreatedAt column of type bigint
* UpdatedAt column of type bigint

#### Scaffolding Models

* To scaffold models when changes are made in database, use following command:

scaffold-dbcontext "Data Source=.\SQLEXPRESS;Initial Catalog=AgentPortal;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Model -ContextDir Database -Project Intelli.AgentPortal.Domain -NoOnConfiguring -Force -t AspNetUsers, Batches, ColumnPreferences, Companies, configurations, Countries, DocumentClasses, DocumentGroupNames, DocumentsPerCompanies, PasswordHistory, RoleScreens, RoleScreenColumns, RoleScreenElements, Screens, ScreenColumns, ScreenElements, SystemRoles, SystemUsers, SystemUserCountries, SystemUserRoles, UserPreferences, UserSessions, UsersPerCompanies

* If new tables are added in database it should be appended in above mentioned command at the end e.g. ,table\_name1, table\_name2

After running above mentioned command you have to do following steps to remove errors:

* Remove AspNetUsers model class as it is already added in custom folder to be inherited from IdentityUser base class.
* Generated AgentPortalContext class should not be inherited from DbContext because it is being inherited by IdentityDbContext in its partial implementation which is required to use identity framework. So remove DbContext inheritence from generated AgentPortalContext class.

#### After Scaffolding Models

Now inside Intelli.AgentPortal.Domain => Model => Custom folder of domain library, add partial class with the same name as of model name. This partial class should implement IEntity interface.

AgentPortal\Intelli.AgentPortal.Domain\Model\Core\IEntity.cs

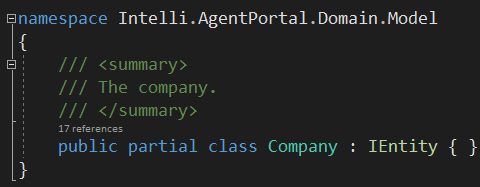


Figure 5

### Seeding screen and its elements in database.

In order to seed screens and its elements migration is created and can be run on databases.

Following is an example migration to seed data

USE [AgentPortal]

GO

IF NOT EXISTS (SELECT \*

FROM [dbo].[\_\_EFMigrationsHistory]

WHERE [MigrationId] = '20211008221300\_Seed\_Screen\_Data')

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

SET IDENTITY\_INSERT [dbo].[Screens] ON

INSERT INTO [dbo].[Screens] ([Id], [ScreenName], [IsActive], [CreatedAt], [UpdatedAt])

VALUES (11, 'UserPerCompany', 1, 0, 0)

SET IDENTITY\_INSERT [dbo].[Screens] OFF

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20211008221300\_Seed\_Screen\_Data', N'5.0.9');

COMMIT;

PRINT 'Data Seeded Successfully!'

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION

PRINT 'Error: ' + ERROR\_MESSAGE()

PRINT 'Transactions Rolled Back'

END CATCH

END

# Features

# Following are the main features of backend and explanation that how they are implemented:

## User & Role Management

Authorization is managed by associating users with roles. Users and roles have NxN relation.

Following database diagram shows relation between users and roles and other entities associated to implement authorization.

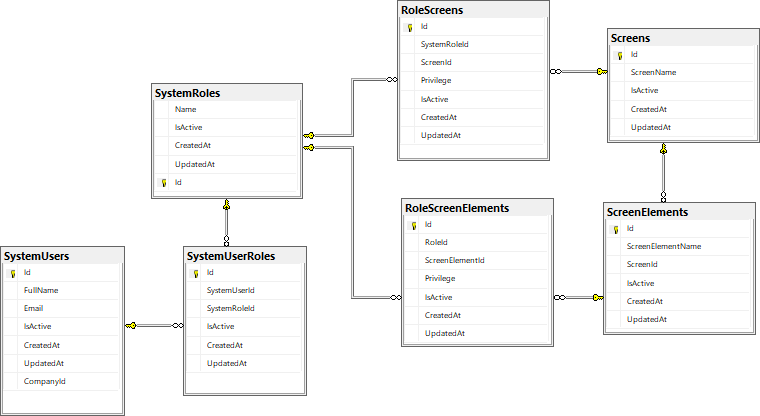


Figure 6

# 

* Screens and screen elements have many to many relation with system roles.
* System Roles have many to many relation with System Users.
* Screens are associated with roles with either admin rights or custom rights.
* In case of custom rights each screen element which is associated with screen can be further configured and can either be hidden, read only or full control.
* When multiple roles are associted with a user, and two or more roles have different privileges regarding same screen, then privileges are merged in a way that higher privilege have precedence over lower privilege.
* Controllers in AgentPortal\Intelli.AgentPortal.Api\Controllers\Authorization\

are responsible for providing APIs to associate screen and screen elements with roles and then roles with users.

* In Auth controller when user is authenticated, the PrivilegesService is responsible for providing privilege related data of the user with the help of PrivilegesMergeFactory & PrivilegesDTOFactory.
* User’s privileges dto is built in PrivilegesDTOFactory and multiple privileges are merged in PrivilegesMergeFactory.

## Users Per Company

* Agent portal backend can restrict total number of users that can be logged into the system per company. This number can be set by calling api in following controller:

AgentPortal\Intelli.AgentPortal.Api\Controllers\Company\UsersPerCompanyController.cs

* In order to implement users per company restriction it is necessary to implement that each user can have only one session against its credentials.

Restriction against multiple sessions of a user is implemented in the following way:

* Whenever a user signs in with its valid credentials a json web token is returned in successfull response along with user’s privileges.
* This JWT has a claim known as Jti which uniquely identifies the jwt token. This claim is added in token in following service:

AgentPortal\Intelli.AgentPortal.Api\Services\JWT\Impl\JWTService.cs

* A generated GUID (Globally Unique Identifier) is used as Jti claim in JWTService.
* Auth controller saves Jti claim along with user’s id using session manager which is implemented in following service:

AgentPortal\Intelli.AgentPortal.Api\Services\Session\Impl\SessionManager.cs

* Session manager treats Jti as session id and stores it in database and in memory.
* Database store is used as a backup to in mermory storage, in case, if web service process restarts for any reason.
* In memory session storage is used to quickly access active sessions for ping requests.
* Front end actively pings backend api to check if current session is still alive or not.
* Ping request extracts Jti from token and passes it to session manager along with userid to check if session is still active.

AgentPortal\Intelli.AgentPortal.Api\Controllers\Authentication\PingController.cs

If session is active user can continue to perform its operations and if it is not active user is logged out forcefully.

* User can propery ends its session by calling logout end point from front end:

AgentPortal\Intelli.AgentPortal.Api\Controllers\Authentication\LogoutController.cs

* If user does not ends its session properly and closes the browser and tries to log in again or during an active session user tries to login from another browser, in both cases user is prompted by a screen showing that its previous session is still active and if he / she wants to close that session, if user selects yes its previous session is closed.

AgentPortal\Intelli.AgentPortal.Api\Services\Auth\Impl\AgentPortalSignInManager.cs

* Sign in manager also implements Users per company restriction, before checking active session of the user, and compares total allowed users per company with currenty active session of users of the company, and if that limit exceeds user cannot sign in.

## Password Management

Following are the main features of password management and how they are implemented:

* If user forgots its password, it can click on the forgot password link on front end UI to request password reset link.

AgentPortal\Intelli.AgentPortal.Api\Controllers\Authentication\AuthController.cs

AgentPortal\Intelli.AgentPortal.Api\Services\Auth\Impl\AuthService.cs

* On clicking on the link in its registered email address user is prompted to update its password.
* User’s password selection is bound by the password strength policy created by the administrator.

AgentPortal\Intelli.AgentPortal.Api\Controllers\Configuration\ConfigurationsController.cs

* In addition to password strength an admin can also force users to change their passwords after x number of days and restrict them to not use last x number of passwords.
* To implement this, system maintains a log of user’s old passwords by calculating password’s hash and storing it in database along with user’s id and date and time, whenever the password is changed.
* Auth service is responsible for maintaining this log when user updates its credentials.

AgentPortal\Intelli.AgentPortal.Api\Services\Auth\Impl\AuthService.cs

* Password validator is responsible for checking if user is using its old password to update its credentials, by comparing old password hashes with current password’s hash.

AgentPortal\Intelli.AgentPortal.Api\Services\Auth\Impl\AgentPortalPasswordValidator.cs

* To implement, user must change its password after x number of days, system checks when user last changed its password by reading same log, in auth controller, when user is signing in and after its credentials are validated.

AgentPortal\Intelli.AgentPortal.Api\Controllers\Authentication\AuthController.cs

* A flag is retruned in login response that user must change password or not. And if true, user cannot perform any other task unless it changes its password. (Implemented in front end)

## Audit

Audit mechanism is used to keep track of all transactional operations performed on entities used in project and save them in audit database along with the user name who performed the operation, date on which the operation is performed and log of data added, updated or deleted. It is implemented as follows:

* Audit service uses RabbitMQ service as event bus to perform audit log operations asynchronously.
* Repository pattern raises AfterSave event after each save changes call.
* When save changes function is called in controller it creates audit logs and if AfterSave event is handled it passes that audit logs to event after invoking the event.
* If a transaction is active these audit logs are added in transaction and forwarded when transaction commit is successful.

AgentPortal\Intelli.AgentPortal.Domain\Repository\Impl\GenericRepository.cs

* Each controller that perform a database update and wants to log changes, can forward audit logs to event sender by implementing after save event.

AgentPortal\Intelli.AgentPortal.EventBus.RabbitMQ\Sender\Impl\EventSender.cs

* Event sender is responsible for sending audit logs updates to event bus and ques the messages in event bus.
* An event receiver runs as a background service and reads those messages from event bus.

AgentPortal\Intelli.AgentPortal.EventBus.RabbitMQ\Receiver\Impl\EventListener.cs

* Event receiver passes the messages to proper event consumer to process those messages, which essentialy contains audit logs.
* Audit event handler is responsible for processing the audit logs and saves them in audit database.

AgentPortal\Intelli.AgentPortal.Api\Events\Handlers\AuditEventHandler.cs

* Audit logs can be read using Audit Controller.

AgentPortal\Intelli.AgentPortal.Api\Controllers\Audit\AuditController.cs

## Logging mechanism

In addition to audit logs, system also implements exception logging using Nlogs.

NLog is a flexible and free logging platform for various .NET platforms, including .NET standard. NLog makes it easy to write to several targets, (database, file, console) and change the logging configuration on-the-fly

Nlog is configured in program file while configuring the web host defaults as follows:

|  |
| --- |
| public static IHostBuilder CreateHostBuilder(string[] args) =>  Host.CreateDefaultBuilder(args)  .ConfigureWebHostDefaults(webBuilder =>  {  webBuilder.UseStartup<Startup>()  .ConfigureLogging((hostingContext, logging) =>  {  logging.AddNLog(hostingContext.Configuration.GetSection("Logging"));  });  }); |
| A Nlog config file is added in project’s root directory to further configure Nlog behavior.   |  | | --- | | <?xml version="1.0" encoding="utf-8" ?>  <nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  <!-- the targets to write -->  <targets>  <!-- write to file -->  <target name="applog" xsi:type="File"  fileName="C:\Log\Intelli.AgentPortal.Api-${shortdate}.log"  layout="${longdate} - ${message} -  ${exception:format=StackTrace}${newline}" />  </targets>  <!-- rules to map from logger name to target -->  <rules>  <!-- all logs including Microsoft -->  <logger name="\*" minlevel="Trace" writeTo="applog" />  </rules>  </nlog> | |  | |

To add more configuration settings use nlog documentation:

[**https://nlog-project.org/config/**](https://nlog-project.org/config/)

* + This document ends here