Gear Documentation

Lupei Nicolae

Jun 16, 2020

Getting Started

Gear is a .NET framework for building your apps.

The primary goal of Gear BPMN is to provide a notation that is readily understandable by all business users, from the business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation.BPMN will also advance the capabilities of traditional business process notations by inherently handling B2B business process concepts, such as public and private processes , as well as advanced modeling concepts of entities and processes. A software system designed to fully automate content management in websites. The goal is to reduce or eliminate the intervention of programmers in editing and managing their sites. The system facilitates the organization, control and publication of documents or other content, such as multimedia images and resources, often facilitates document sharing. Allows translations from the website by adding languages and keys, and also has the ability to translate automatically using external providers. Allows you to create dynamic pages that include creating page blocks and managing them, adding themes, and adapting pages to different environments. It has functional interaction with the database, in the first phase with MSSql and PostGreSql, there is also the management of the entities in the system that are used to store the information in the site, this point is automated and often generated by the system depending on the user's options. The main purpose is to build on a framework a much faster site than if it started from 0. Have an installer that allows the initial setup. It has many working tools with content in many formats, being as close as possible to the user. Possess the cache module for accessing information faster and render in templates, which can also be custom. Possess the notification and messaging module, so the system alerts you about system changes or actions made by other users.

The following platforms are supported:

- .NET Core 2.0+
- .NET Standard 2.0+

CHAPTER 1

Installation

The app can be started using 3 environments:

- `Development` - is used for development purpose
- `Stage` - is used on pre- production
- `Production` - is used for clients, on server side

`NOTE`: For development use Development env `NOTE`: Each configuration corresponds to a configuration file, like this: `appsettings.{Env}.json`

Structure of appssetings file:

```
{
  "SystemConfig": {
    "MachineIdentifier": ".GR.Prod"
  },
  "ConnectionStrings": {
    "Provider": "Npgsql.EntityFrameworkCore.PostgreSQL",
    "ConnectionString": "Host=127.0.0.1;Port=5432;Username=postgres;Password=Gear2019;
↪Persist Security Info=true;Database=GEAR.PROD;MaxPoolSize=1000;"
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Error"
    }
  },
  "HealthCheck": {
    "Timeout": 3,
    "Path": "/health"
  },
  "LocalizationConfig": {
    "Languages": [
      {
        "IsDisabled": false,
```

(continues on next page)

(continued from previous page)


```
        "Identifier": "en",
        "Name": "English"
      },
      {
        "IsDisabled": false,
        "Identifier": "ro",
        "Name": "Romanian"
      },
      {
        "IsDisabled": true,
        "Identifier": "ru",
        "Name": "Russian"
      },
      {
        "IsDisabled": true,
        "Identifier": "it",
        "Name": "Italian"
      },
      {
        "IsDisabled": true,
        "Identifier": "fr",
        "Name": "French"
      },
      {
        "IsDisabled": true,
        "Identifier": "de",
        "Name": "German"
      },
      {
        "IsDisabled": true,
        "Identifier": "uk",
        "Name": "Ukrainian"
      },
      {
        "IsDisabled": true,
        "Identifier": "ja",
        "Name": "Japanese"
      },
      {
        "IsDisabled": true,
        "Identifier": "zh",
        "Name": "Chinese"
      },
      {
        "IsDisabled": true,
        "Identifier": "el",
        "Name": "Greek"
      },
      {
        "IsDisabled": true,
        "Identifier": "nl",
        "Name": "Dutch"
      },
      {
        "IsDisabled": true,
        "Identifier": "pl",
        "Name": "Polish"
```


(continues on next page)

(continued from previous page)


```
      },
      {
        "IsDisabled": true,
        "Identifier": "es",
        "Name": "Spanish"
      }
    ],
    "Path": "Localization",
    "SessionStoreKeyName": "lang",
    "DefaultLanguage": "en"
  },
  "IsConfigured": true,
  "LdapSettings": {
    "ServerName": "",
    "ServerPort": 389,
    "UseSSL": false,
    "Credentials": {
      "DomainUserName": "",
      "Password": ""
    },
    "SearchBase": "",
    "ContainerName": "",
    "DomainName": "",
    "DomainDistinguishedName": ""
  },
  "WebClients": {
    "CORE": {
      "uri": "http://159.69.195.160:6969"
    },
    "BPMApi": {
      "uri": "http://159.69.195.160:6969"
    }
  },
  "RedisConnection": {
    "Host": "127.0.0.1",
    "Port": "6379"
  },
  "BackupSettings": {
    "Enabled": false,
    "UsePostGreSql": false,
    "UseMsSql": false,
    "BackupFolder": "ISODMS",
    "Interval": "24",
    "PostGreSqlBackupSettings": {
      "PgDumpPath": "C:\\Program Files\\PostgreSQL\\11\\bin\\pg_dump.exe",
      "Host": "localhost",
      "Port": "5432",
      "User": "postgres",
      "Password": "1111",
      "Database": "ISODMS.PROD",
      "FileExtension": "pgbackup"
    },
    "MsSqlBackupSettings": {
    }
  },
  "EmailSettings": {
    "Enabled": true,
```


(continues on next page)

(continued from previous page)

```
    "Host": "smtp.office365.com",
    "Port": 587,
    "Timeout": 5000,
    "EnableSsl": true,
    "NetworkCredential": {
      "Email": "iso_dms.mail@indrivo.com",
      "Password": "I50_dm5.M@!1"
    }
  },
  "Sentry": {
    "Dsn": "https://a898fb5130514f2485704835f8109591@sentry.io/1547729",
    "IncludeRequestPayload": true,
    "SendDefaultPii": true,
    "MinimumBreadcrumbLevel": "Debug",
    "MinimumEventLevel": "Warning",
    "AttachStackTrace": true,
    "Debug": true,
    "DiagnosticsLevel": "Error"
  }
}
```

1.1 Explanation of appsettings blocks:

- `SystemConfig` - represent a general section that provide some global info:
  - MachineIdentifier - is used for identify the app id, if is installed multiple GEAR apps, after app installation it is ovverided by generated string
- `ConnectionStrings` - represent databases providers configuration
  - Provider - default is postgres
    - Npgsql.EntityFrameworkCore.PostgreSQL - postgres ,enabled and default
    - Microsoft.EntityFrameworkCore.SqlServer - enabled
    - Microsoft.EntityFrameworkCore.Sqlite - for the future
    - Microsoft.EntityFrameworkCore.InMemory - for the future
    - Microsoft.EntityFrameworkCore.Cosmos - for the future
    - Pomelo.EntityFrameworkCore.MySql - for the future
    - Pomelo.EntityFrameworkCore.MyCat - for the future
    - EntityFrameworkCore.SqlServerCompact40 - for the future
    - EntityFrameworkCore.SqlServerCompact35 - for the future
    - EntityFrameworkCore.Jet - for the future
    - MySql.Data.EntityFrameworkCore - for the future
    - FirebirdSql.EntityFrameworkCore.Firebird - for the future
    - EntityFrameworkCore.FirebirdSql - for the future
    - IBM.EntityFrameworkCore - for the future
    - EntityFrameworkCore.OpenEdge - for the future

- [Logging] - see microsoft docs
- `LocalizationConfig` - language configurations
- `IsConfigured` - This property determines whether the app has been installed or not, if set to true then the configurations set in the database are taken, otherwise when accessing any page, it will be redirected to the installer
- `LdapSettings` - This involves configuring the AD mode
- `RedisConnection` - configurations for distributed cache
  - Host - represent the ip address of redis connection
  - Port - represent the port where is bind redis service, default: 6379
- `BackupSettings` - this section is used for backup module, now is developed only for postgres provider
- `EmailSettings` - this section is used for email client
  - Enabled - set active or inactivity of service
  - Host - the smptp host
  - Port - the port of smtp
  - Timeout - represents the time allowed for the service to wait for the message to be successfully sent
  - EnableSsl - represent usage of smtp with ssl
  - NetworkCredential
    - Email - existent smtp email
    - Password - the password of smtp email
- `Sentry` - consult sentry documentation for .net core

1.2 App run

To start the app, you need:

1. Restore ui packages on all razor projects (is optional step because, they are restored on build)

```
libman restore
```

2. Restore C# nuget packages by typing

```
dotnet restore
```

3. Build. To build, you must navigate the explorer to the path: `./src/GR.WebHosts/GR.Cms` or

```
cd ./src/GR.WebHosts/GR.Cms
```

   after it execute the following command:

```
dotnet build
```

4. If build has run successfully, it is the green wave to start the project

```
dotnet run
```

   `optional` for change exposed port

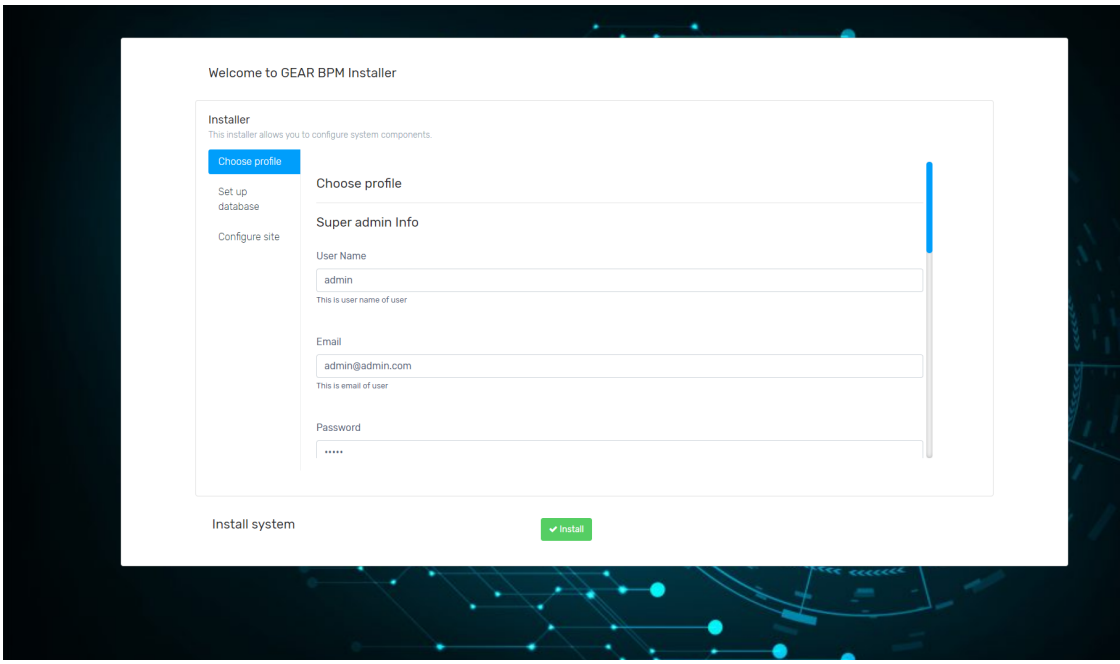
```
dotnet run --urls=http://localhost:5001/
```

1.3 Install steps

`Note`: Be sure that in appsettings{Env}.json, the IsConfigured property is set to false

1. Start the application You will be met by the following message describing the platform

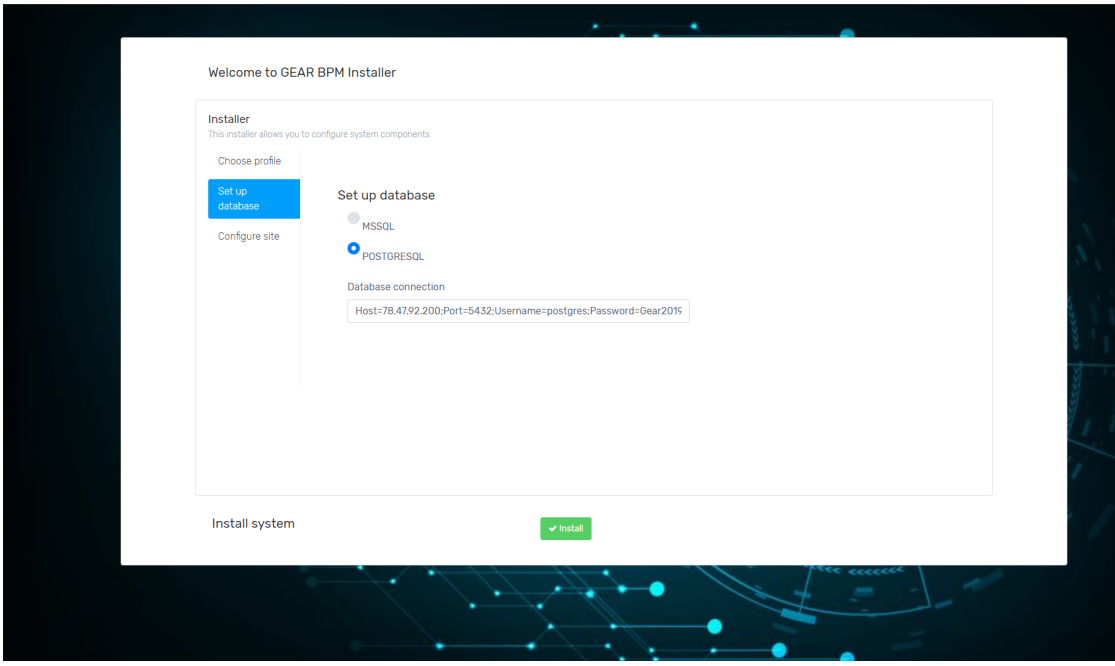
Welcome board Click on `Go to installation`



2. Configure admin profile tab Settings:

- `User Name` - adminstrator user name
- `Email` - your email address to receive emails on system events

- `Password` and `Confirm Password` - the administrator password
- `First Name` - admin first name
- `Last Name` - admin last name
- `Organization Name` - represent the default organization name



1. Set up database provider of database provider `Note:` Use postgres default, because MsSql has not been tested for a long time, we plan support for other providers `Connection String example:` Host=127.0.0.1;Port=5432;Username=postgres;Password=Gear2020;Persist Security Info=true;Database=Gear.PROD;MaxPoolSize=1000;
2. Press `Install` button and wait until the system is installed

CHAPTER 2

Tehnologies

.NET CORE 2.2, C#, MVC, JQuery, PostgreSql database, PgAdmin, javascript native, javascript Prototype, SweetAlert2, Libman, Razor View Libraries, Modular Arhitecture, Entity Framework, Bootstrap 4, .Net Standard, Nginx, IIS, Kestrel

CHAPTER 3

Requirements

3.1 Compatibility

The framework is cross platform, thanks .net core, is compatible with Windows, Linux distributive and Mac OS.

3.1.1 Windows OS Requirements

To start the app you need the following packages installed:

- .Net Core SDK 2.2.402 - C# cross platform framework
- Redis - Distributed cache soft
- Pg Admin 4 - Postgres manager for interact with database
- PostgreSql - Database provider
- Libman - UI packages provider

3.1.2 Ubuntu 18.04 or another OS Requirements

To start the app you need the following packages installed:

- .Net Core SDK 2.2.402 - C# cross platform framework, optional link
- Redis - Distributed cache soft
- Pg Admin 4 - Postgres manager for interact with database
- PostgreSql - Database provider
- Libman - UI packages provider

`Note`: For other OS supported by .net core, consult internet sources

CHAPTER 4

Hardware Requirements

4.1 OS

- Windows
- MAC OS
- Red Hat Enterprise Linux 7, 6 - 64-bit (x86_64 or amd64)
- CentOS 7 - 64-bit (x86_64 or amd64)
- Oracle Linux 7 - 64-bit (x86_64 or amd64)
- Fedora 28, 27 - 64-bit (x86_64 or amd64)
- Debian 9 (64-bit, arm32), 8.7 or later versions - 64-bit (x86_64 or amd64)
- Ubuntu 18.04 (64-bit, arm32), 16.04, 14.04 - 64-bit (x86_64 or amd64)
- Linux Mint 18, 17 - 64-bit (x86_64 or amd64)
- openSUSE 42.3 or later versions - 64-bit (x86_64 or amd64)
- SUSE Enterprise Linux (SLES) 12 Service Pack 2 or later - 64-bit (x86_64 or amd64)
- Alpine Linux 3.7 or later versions - 64-bit (x86_64 or amd64)

4.2 Hardware

To run the app you should have the following minimum requirements:

- `RAM`: 8+ GB (ECC/NonECC) > 1333 MHZ
- `CPU`: (4+) x 3 GHZ, XEON or I7 , new generations of cpu to be desired
- `Storage` HDD/SSD/SSHD : > 50 GB

CHAPTER 5

Modules

The framework has developed the following modules:

- Module for managing dynamic entities
- Workflow manager and builder module
- Task Management Module
- Calendar module
- Synchronization component with external calendar
- Notifications module
- Document Management Module (DMS)
- User Management Module
- Role and permissions management module
- Report and Statistics Module
- The chat module
- Content management module
- Page Management module
- Forms management module
- Menu management module
- The localization module
- Authentication and authorization module

CHAPTER 6

Core Razor Module

`Note`: This library contains `System.Drawing` namespace, for usage in Linux, you must install the following packages:

```
sudo apt install libc6-dev
sudo apt install libgdiplus
```

CHAPTER 7

Calendar Module

Calendar is a module for the GEAR framework that allows the creation of events, inviting people to events within an organization. There is a nice interface like the Outlook application. It is possible to synchronize with providers such as Google and Outlook

7.1 Calendar Abstractions

7.1.1 Command line

```
dotnet add package GR.Calendar.Abstractions --version 1.0.3
```

7.1.2 Package Manager

```
PM> Install-Package GR.Calendar.Abstractions -Version 1.0.3
```

#Add calendar extension to GEAR

```
//-----------------------------------Calendar Module--------------------------------
↪----
                  config.GearServices.AddCalendarModule<CalendarManager>()
                          .AddCalendarModuleStorage<CalendarDbContext>(options =>
                          {
                                  options.GetDefaultOptions(Configuration);
                                  options.EnableSensitiveDataLogging();
                          })
                          .AddCalendarRazorUIModule()
                          .SetSerializationFormatSettings(settings =>
                          {
                                  settings.ReferenceLoopHandling =␣
↪ReferenceLoopHandling.Ignore;
```

(continues on next page)

(continued from previous page)

```
                        })
                        .AddCalendarRuntimeEvents()
                        .RegisterSyncOnExternalCalendars()
                        .RegisterTokenProvider<CalendarExternalTokenProvider>()
                        .RegisterCalendarUserPreferencesProvider
↪<CalendarUserSettingsService>()
                        .RegisterGoogleCalendarProvider()
                        .RegisterOutlookCalendarProvider(options =>
                        {
                                options.ClientId = "d883c965-781c-4520-b7e7-
↪83543eb92b4a";
                                options.ClientSecretId = "./7v5Ns0cT@K?BdD85J/
↪r1MkE1rlPran";
                                options.TenantId = "f24a7cfa-3648-4303-b392-
↪37bb02d09d28";
                        })
                        .AddCalendarGraphQlApi();
```

CHAPTER 8

Workflow Builder Module

8.1 Description

Workflow builder is a module that belongs to Gear and aims to manage states for an object.

A workflow consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information It can be depicted as a sequence of operations, declared as work of a person or group, an organization of staff, or one or more simple or complex mechanisms.

The module is made on 3 layers:

- GR.WorkFlows
- GR.WorkFlows.Abstractions
- GR.Workflows.Razor

GR.WorkFlows.Abstractions contains interfaces:

- `IWorkflowContext` - contains the contracts with the entities, in order to use this interface is injected
- `IWorkFlowCreatorService` - contains the description of the workflow creation methods
- `IWorkflowExecutorService` - contains the description of the methods of changing transitions, states and executing actions for a particular object

8.2 Installation

To install this module you need to refer to GR.WorkFLows.Abtractions or to the library on Nuget with the same name
`Example`

```
services.RegisterGearWebApp(config =>
        {
//--------- some configuration ------------
```

(continues on next page)

(continued from previous page)

```
//-------------------------------------Workflow module--------------------------------
↪-----
            config.GearServices.AddWorkFlowModule<WorkFlow, WorkFlowCreatorService,␣
↪WorkFlowExecutorService>()
                .AddWorkflowModuleStorage<WorkFlowsDbContext>(options =>
                {
                    options.GetDefaultOptions(Configuration);
                    options.EnableSensitiveDataLogging();
                })
                .AddWorkflowRazorModule();

//--------- another modules ------------
```

`WorkFlowCreatorService`, `WorkFlowExecutorService` are the classes that have the basic implementation for the behavior of a workflow, they implement `IWorkFlowCreatorService` and `IWorkFlowExecutorService`, so if you want the basic implementation you need to use GR.WorkFlows. At your choice you can inherit these classes and override the methods

8.3 Register entity contract

In order to be able to use workflow manger for an object it is necessary to create a contract for an entity. For this we use the interface `IWorkflowExecutorService` Here we use the method:

```
Task<ResultModel<Guid>> RegisterEntityContractToWorkFlowAsync([Required] string␣
↪entityName, Guid? workFlowId);
```

Parameters:

- `entityName` -> represents the name of the entity, in this version there is no close connection with the entity from the database and this one from the workflow, we chose a more abstract way so that we do not have dependents and can be more generic
- `workFlowId` -> represents the id of a workflow already created using the ui builder (id belongs to the WorkFlow entity)

Also a contract can be registered using `IServiceCollection` `Example`:

```
config.GearServices.RegisterWorkFlowContract(nameof(DocumentVersion), Guid.Empty);
```

8.4 Workflow Actions

Workflow actions are post actions that are executed when changing a state for an object To create an action we must create a class that inherits from `BaseWorkFlowAction`, it contains an abstract method `InvokeExecuteAsync` that receives as a parameter a Dictionary parameter `Dictionary<string, string>`. This method will be called when the action is invoked to change the state of an object, of course this action must be attached to the appropriate transition.

`Base action`

```
public abstract class BaseWorkFlowAction
    {
        #region Injectable
```

(continues on next page)

(continued from previous page)

```
        /// <summary>
        /// Executor
        /// </summary>
        protected readonly IWorkFlowExecutorService Executor;

        #endregion

        /// <summary>
        /// Entry state
        /// </summary>
        protected EntryState EntryState { get; set; }

        /// <summary>
        /// Current transition
        /// </summary>
        protected Transition CurrentTransition { get; set; }

        /// <summary>
        /// Next transitions
        /// </summary>
        protected IEnumerable<Transition> NextTransitions { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="entry"></param>
        /// <param name="currentTransition"></param>
        /// <param name="nextTransitions"></param>
        protected BaseWorkFlowAction(EntryState entry, Transition currentTransition,␣
↪IEnumerable<Transition> nextTransitions)
        {
            EntryState = entry;
            CurrentTransition = currentTransition;
            NextTransitions = nextTransitions;
            Executor = IoC.Resolve<IWorkFlowExecutorService>();
        }

        /// <summary>
        /// Execute
        /// </summary>
        /// <returns></returns>
        public abstract Task InvokeExecuteAsync(Dictionary<string, string> data);
    }
```

Example of send notifications actions

```
    public class SendNotificationAction : BaseWorkFlowAction
    {
        #region Injectable

        /// <summary>
        /// Inject notifier
        /// </summary>
        private readonly INotify<GearRole> _notify;

        #endregion
```

(continues on next page)

(continued from previous page)

```
        public SendNotificationAction(EntryState entry, Transition transition,␣
↪IEnumerable<Transition> nextTransitions) : base(entry, transition, nextTransitions)
        {
            _notify = IoC.Resolve<INotify<GearRole>>();
        }

        /// <summary>
        /// Execute data
        /// </summary>
        /// <param name="data"></param>
        /// <returns></returns>
        public override async Task InvokeExecuteAsync(Dictionary<string, string> data)
        {
            var rolesForPrevTransition = await Executor.
↪GetAllowedRolesToTransitionAsync(CurrentTransition);
            var subject = "Entry x";
            if (data.ContainsKey("Name")) subject = data["Name"];
            await _notify.SendNotificationAsync(rolesForPrevTransition, new␣
↪Notification
            {
                Subject = $"{subject} state changed",
                Content = $"{subject} has changed its status from {CurrentTransition?.
↪FromState?.Name} to {CurrentTransition?.ToState?.Name}",
                SendLocal = true,
                SendEmail = true,
                NotificationTypeId = NotificationType.Info
            }, EntryState.TenantId);

            foreach (var nextTransition in NextTransitions)
            {
                var rolesForNextTransition = await Executor.
↪GetAllowedRolesToTransitionAsync(nextTransition);

                await _notify.SendNotificationAsync(rolesForNextTransition, new␣
↪Notification
                {
                    Subject = "You have new actions",
                    Content = $"{subject} can be switched to {nextTransition?.ToState.
↪Name} state",
                    SendLocal = true,
                    SendEmail = true,
                    NotificationTypeId = NotificationType.Info
                }, EntryState.TenantId);
            }
        }
    }
```

8.4.1 The registration of an action is done in the following way:

```
services.RegisterWorkflowAction<TActionClass>();
```

Note: TActionClass need to inherit BaseWorkFlowAction

8.4.2 Injecting services into action handlers

The injection of services can only be done through Castle Windsor, an example of injection

```
IoC.Resolve<TService>();
```

Note: This service must first be registered using `IoC` service registration methods

8.5 The structure of a workflow

A workflow has the following structure :

- `Name` - the name of your workflow
- `Description` - some description about it
- `Enabled` - represent if it is active for usage
- `States` - represent a list of states that caracterize it
- `Transitions` - represent transitions between states

A state has the following structure:

- `Name` - unique name for some workflow that will be displayed on usage
- `Description` - something descriptive
- `IsStartState` - represents the initial state that will be set for an object, note: only one start state can exist
- `IsEndState` - represents the last state of an object
- `AdditionalSettings` - they are used for store some settings as a dictionary (`ex`: we store here the position x and y on the builder canvas)

A transition has the following structure:

- `Name` - abstract name that identify the transition
- `FromState` - is the start point for a transition, ex: the first transition has FromState value of the first state of a workflow
- `ToState` - is the end point for entry transition
- `Actions` - actions are the handlers that will be executed after the state of object will be changed to another, regulary in this system actions are classes that are use to execute some actions for state change
- `AllowedRoles` - here we store the user roles that can do this change of transition

8.6 The structure of entry that use workflows

For store the state of an object we use an `Entry State` entity that have the following structure:

- `Contract` - represents the id of contract of entity and workflow
- `EntryId` - represents the id of object
- `State` - represents the current state of object
- `Message` - represents the message that will be changed on state change

We store history of object states in `EntryStateHistory`, this entity has the following structure:

- `EntryState` - store the id of entry state
- `FromState` - store the precedent state of entry
- `ToState` - store the current state that is set in EntryState
- `Message` - represents the message that is set when the state of the object changes

8.7 License

MIT

CHAPTER 9

Change logs

v.1.9.4 - Lupei Nicolae May 2020

- add user activity module
- add phone verification module
- bug fixing
- add new extensions
- fix pagination extension
- refact core
- add new events for modules
- refact identity modules
- add model validator
- add new validation attributes
- refact naming
- add auth for notification hub
- add support for external devices to hub
- add error prevent with custom message on json api
- add support for multiple authentification types simultaneous
- add database for ip tracking
- add helper for parse and validate phone numbers
- add 2factor auth service with phone number
- fix user address service
- use and create mappers for dto
- add api for user

- remove districts from localization module
- add structure for read docs reading
- refact docs files
- generate xml comments for all projects and bind on swagger
- add configuration to generate xml comments

v.1.9.3 - Lupei Nicolae April 2020

- separe groups module from identity
- separe user profile module from identity
- separe permissions from identity
- separe country module from identity
- create custom permissions registrator
- create permissions configurations for some modules
- create identity.clients module from identityServer4
- create custom contexts for identityServer4 and custom registration
- add menu initializers

v.1.9.2 - Lupei Nicolae 16 March 2020

- develop ui module for database backup
- remove unused features
- add database provider for localization (beta)
- fix warning (too long file names)
- refactoring cache module
- identity from string to guid
- add seq logging provider
- refactoring logging provider
- add menu initialiazers for dashboard, notification, report modules
- refactoring controllers

v1.9.1 - Lupei Nicolae 13 March 2020

- clean iso infrastructure

v1.9.0 - Lupei Nicolae 13 March 2020

- files module
- documents module
- commerce module
- bug fixing

v1.8.0 - Lupei Nicolae 01 February 2020

- bug fixing

v1.7.0 - Lupei Nicolae 01 January 2020

- bug fixing

v1.0.0 - Lupei Nicolae 13 February 2019

- identity module
- entities module
- forms module