# Gear Documentation

**Lupei Nicolae**

**Jun 20, 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"
  },
  "IsConfigured": true,
  "LdapSettings": {
    "ServerName": "",
    "ServerPort": 389,
```

(continues on next page)

(continued from previous page)

```
    "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": "AppName",
    "Interval": "24",
    "PostGreSqlBackupSettings": {
      "PgDumpPath": "C:\\Program Files\\PostgreSQL\\11\\bin\\pg_dump.exe",
      "Host": "localhost",
      "Port": "5432",
      "User": "postgres",
      "Password": "1111",
      "Database": "AppName.PROD",
      "FileExtension": "pgbackup"
    },
    "MsSqlBackupSettings": {
    }
  },
  "EmailSettings": {
    "Enabled": true,
    "Host": "smtp.office365.com",
    "Port": 587,
    "Timeout": 5000,
    "EnableSsl": true,
    "NetworkCredential": {
      "Email": "",
      "Password": ""
    }
  },
  "Sentry": {
    "Dsn": "https://a898fb5130514f2485704835f8109591@sentry.io/1547729",
    "IncludeRequestPayload": true,
    "SendDefaultPii": true,
    "MinimumBreadcrumbLevel": "Debug",
    "MinimumEventLevel": "Warning",
```

(continues on next page)

(continued from previous page)

```
    "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
```

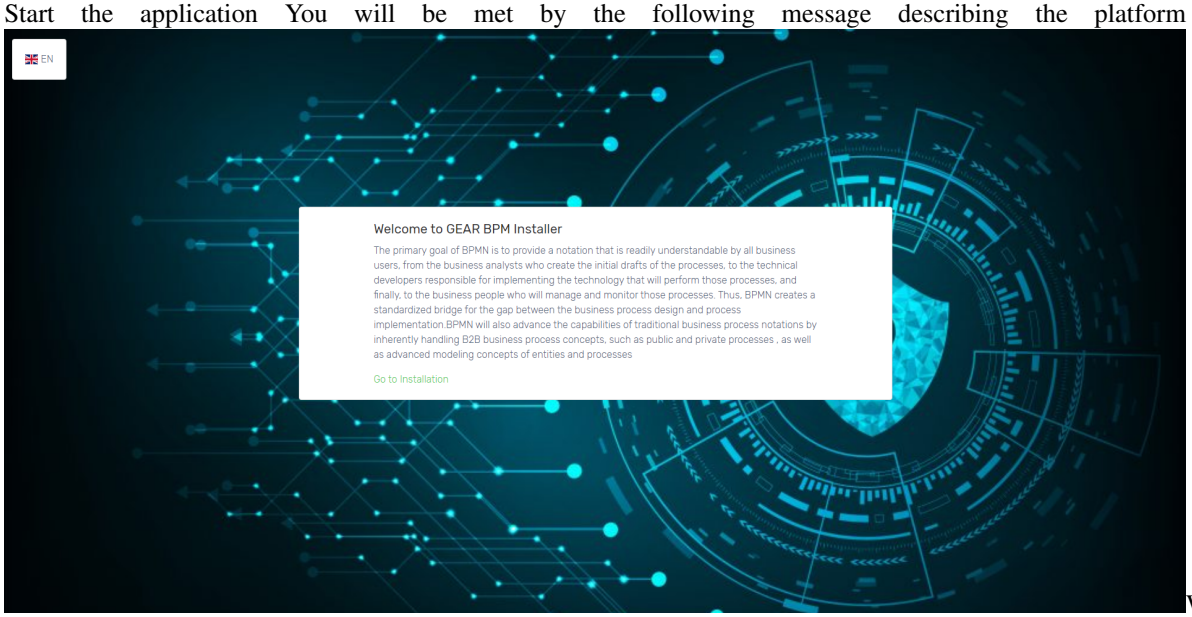
   `optional` for change exposed port

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

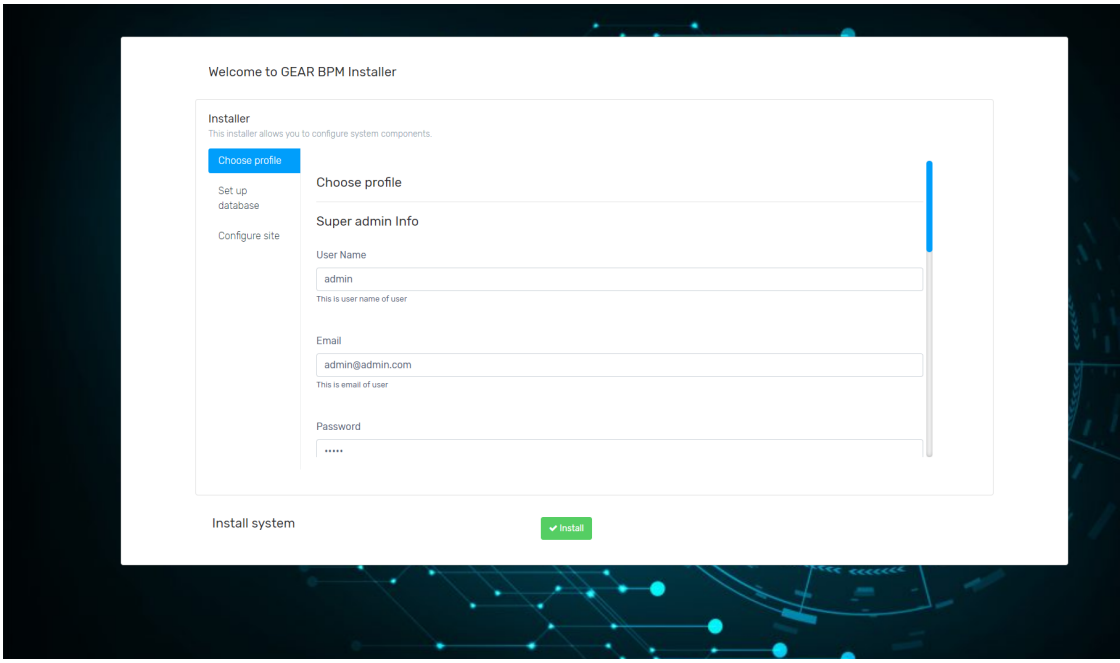
## 1.3 Install steps

`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

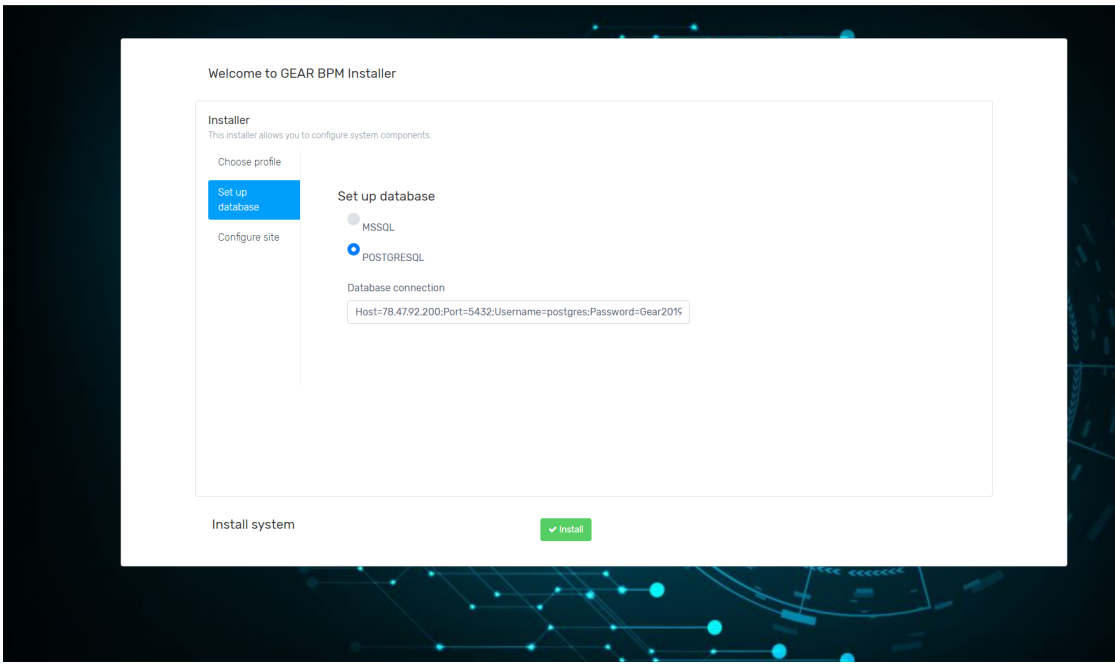



Welcome board Click on `Go to installation`




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, Sweet-Alert2, 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
- Entities management module
- Workflow manager and builder module
- Task Management Module
- Calendar module
- Synchronization component with external calendar
- Notifications module
- Email module
- Document Management Module (DMS)
- User Management Module
- Ldap module
- Role management module
- Permissions module
- Report and Statistics Module
- The chat module
- Content management module
- Page Management module
- Forms management module
- Menu management module
- Localization module
- Authentication and authorization module

- Group management module
- User account activity and tracking module
- User preferences module
- Phone verification module
- User profile module
- Multi tenant module
- Files module
- Subscriptions module
- Commerce module
  - Products module
  - Orders module
  - Payments module
    - Braintree payment method
    - Paypal payment method
    - Mobil pay payment method
    - Apple Pay
    - Google Pay
    - Authorize.net pay
- Dashboard module
- Database backup module (postgres)
- Audit module
- External clients module
- Installer module
- Logger module
- Modules module
- Webhooks module
- 2-Factor authentication module

CHAPTER 6

# GR.Core Module

## 6.1 Description

Core is a module that contains interfaces, abstract classes, extensions, validations, helpers, it is the module that the other modules should extend

## 6.2 Install

- Package manager

```
Install-Package GR.Core -Version 1.9.3
```

- .net cli

```
dotnet add package GR.Core --version 1.9.3
```

- Package reference

```
<PackageReference Include="GR.Core" Version="1.9.3" />
```

- Packet cli

```
paket add GR.Core --version 1.9.3
```

## 6.3 Module usage

```
graph LR
A[GR.Core] -- extend --> B(Module abstraction 1)
A -- extend --> C(Module abstraction 2)
```

(continues on next page)

(continued from previous page)

```
A -- extend --> R(Module abstraction 3)
B --> D{Application}
C --> D
R --> D
```

## 6.4 Abstractions

- `IBackgroundTaskQueue` - interface that is used for push tasks to be executed in background
- `IBaseModel` - interface that has all BaseModel fields, can be used on classes that use inheritance from another classes that BaseModel
- `IBase<T>` - represent IBaseModel with genric type of entry id
- `IDbContext` - this interface is the database context of module, each DbContext must implement this interface
- `ISender` - represent a generic sender of messages, it can send anything, it depend on registered providers
- `IWritableOptions` - custom implementation of IOptionsSnapshot , has a method to update app settings configurations

## 6.5 BaseModel class

BaseModel is the base class for entities that use EF Core Structure of this class can be found below

```
    public abstract class BaseModel : IBase<Guid>
    {
        /// <summary>
        /// Constructor. Initialize object with default values. A unique Id, Creation␣
↪time and set IsDeleted to false
        /// </summary>
        protected BaseModel()
        {
            Id = Guid.NewGuid();
            Created = DateTime.UtcNow;
            Changed = DateTime.UtcNow;
        }

        /// <summary>Stores Id of the Object</summary>
        public Guid Id { get; set; }

        /// <inheritdoc />
        /// <summary>Stores Id of the User that created the object</summary>
        public virtual string Author { get; set; }

        /// <inheritdoc />
        /// <summary>Stores the time when object was created</summary>
        public DateTime Created { get; set; }

        /// <inheritdoc />
        /// <summary>
        /// Stores the Id of the User that modified the object. Nullable
        /// </summary>
```

(continues on next page)

(continued from previous page)

```
        public virtual string ModifiedBy { get; set; }

        /// <inheritdoc />
        /// <summary>Stores the time when object was modified. Nullable</summary>
        public DateTime Changed { get; set; }

        /// <inheritdoc />
        /// <summary>
        /// Stores state of the Object. True if object is deleted and false otherwise
        /// </summary>
        public virtual bool IsDeleted { get; set; }

        /// <inheritdoc />
        /// <summary>
        /// Version of data
        /// </summary>
        public virtual int Version { get; set; }

        /// <inheritdoc />
        /// <summary>
        /// Tenant id
        /// </summary>
        public virtual Guid? TenantId { get; set; }

        /// <summary>
        /// Disable audit tracking
        /// This only work on current instance,
        /// The true value ignore save a new version on audit
        /// </summary>
        [JsonIgnore]
        [NotMapped]
        public virtual bool DisableAuditTracking { get; set; }

        /// <summary>
        /// Get props name
        /// </summary>
        /// <returns></returns>
        public static IEnumerable<string> GetPropsName()
        {
            return typeof(BaseModel).GetProperties().Select(x => x.Name).ToList();
        }
    }
```

## 6.6 Attributes

## 6.7 Events

## 6.8 Exceptions

## 6.9 Extensions

- AssemblyExtensions

  - `GetTypeFromAssembliesByClassName` - get type by name from all assemblies
  - `GetAutoMapperProfilesFromAllAssemblies` - get all AutoMapper mappers from all assemblies
- BoolExtensions
  - `Negate` - negate boolean value
- CollectionExtensions
  - `AddRange` - add range items in HashSet, ICollection
  - `Replace` - replace item in generic list
  - `DistinctBy` - get items distinct by a object propriety
  - `Join` - join a list of strings
  - `IsLast` - return true if item is the last in collection
  - `IsFirst` - check if is first item
  - `GetDifferences` - get differencesfrom 2 lists
  - `ContainsAny` - check if list is contained in another
  - `AnyStartWith`- check if list has any items that start with a string
  - `ToKeyValuePair`- transform to KeyValuePair dictionary
  - `ToObservableCollection` - transform collection to an observable
  - `Randomize` - randomize items in a collection
  - `Transpose` - transpose collection
  - `ToCollection` - IEnumerable to Collection
  - `Combinations` - Returns all combinations of a chosen amount of selected elements in the sequence.
- CryptoExtensions
  - `EncryptWithRSA` - Encrypt a string using the supplied key. Encoding is done using RSA encryption.
  - `DecryptWithRSA` - Decrypt a string using the supplied key. Decoding is done using RSA encryption.
- DateTimeExtensions
  - `EndOfDay` - Get end of day
  - `StartOfDay` - Start of day
  - `DayIndex` - Day of week
  - `Intersects` - Intersects dates
  - `IsWeekend` - Check if is weekend
  - `Age` - Get the age of person
  - `IsLeapYear` - Returns whether or not a DateTime is during a leap year.
  - `DisplayTextDate` - Display text date
  - `CreatedThisWeek` - query to get items created this week
  - `CreatedThisMonth` - query to get items created this month
  - `CreatedToday` - query to get items created today

- `CreatedOneHour` - query to get items created one hour
- `CreatedThisYear` - query to get items created this year
- `ToTimeStamp` - Date to time stamp

## 6.10 Helpers

1. `ZipHelper` ZipHelper is a helper that use ZipArchive and create an archive from a Dictionary<string, MemoryStream> , first arg is name of file and the second is stream of file, the result is a stream. Example:

```
var items = new Dictionary<string, MemoryStream>();
MemoryStream archive = ZipHelper.CreateZipArchive(items);
```

Example of return File in controller:

```
return File(archive, ContentType.ApplicationZip, "archive_name");
```

2. `EncryptHelper` This helper is created for encrypt and decrypt anything , encryption is due grace a pass phrase. Example:

```
//Encryption
 var serializedCard = cards.SerializeAsJson();
 var key = GenerateKey(user);
 var encrypted = EncryptHelper.Encrypt(serializedCard, key);

 //Decryption
  var serializedString = EncryptHelper.Decrypt(cardHash, key);
  var cards = serializedString.Deserialize<IEnumerable<CreditCardPayViewModel>>();
```

1. `MimeTypes` Contains all popular mime types, example of a record:

```
{".ez", "application/andrew-inset"}
```

1. `MimeMapping` This module maps document extensions to Content Mime Type. Example:

```
var getType = MimeMapping.GetMimeMapping(fileName);
```

1. `ModelBinders` ModelBinders is a .net core feature that allow us to map property on HttpRequest serialization, for more info check microsoft docs

- `GearBinder<TValue>` - Custom generic binder Example:

```
public class ProductsFilterRequest
    {
        public virtual int Page { get; set; } = 1;
        public virtual int PerPage { get; set; } = 10;

        [ModelBinder(BinderType = typeof(GearBinder<IEnumerable<CommerceFilter>>))]
        public virtual IEnumerable<CommerceFilter> Filters { get; set; }
    }
```

- `GearDictionaryBinder<TValue>` - custom generic binder for dictionary

1. `Singleton<T, TResolver>` - is an implementation of singleton pattern that allow to use it in a short mode, if value is null, it create automatically a default constructor instance. Example:

```
public static IWindsorContainer Container => Singleton<IWindsorContainer,␣
↪WindsorContainer>.Instance;
```

1. `Arg` validator is a helper that throw an exception if argument not correspond to criteria, list of methods:

- NotNull
- NotNullOrEmpty
- InRange
- LessThan
- LessThanOrEqualTo
- GreaterThan
- GreaterThanOrEqualTo Example:

```
        public static void Register(string name, Type type)
        {
            Arg.NotNullOrEmpty(name, nameof(Register));
            Arg.NotNull(type, nameof(Register));
            Storage.TryAdd(name, type);
        }
```

1. `EnumHelper` It has a method to get the description value from attribute of enum: Example:

```
var result = await _service.DeleteReportAsync(id);
result.Result = ResultMessagesEnum.DeleteSuccess.GetEnumDescription();
```

1. `ExceptionHandler` it contains extensions for enumto transform to ResultModel Example:

```
ResultMessagesEnum.FolderNotSaved.ToErrorModel<Guid>();
```

1. `GearPolicy` - is a helper that execute and retry task in case of an error, it use Policy from Poly library Example:

```
//if the first execution return an error, it retry 3 times with time delay
var depositRequest = await GearPolicy.ExecuteAndRetry(async ()
                => await _coinbaseProClient.DepositsService.
↪DepositFundsAsync(paymentMethod, amount, Currency.USD));
```

1. `IoC` - is an inversion of control helper that extend Castle Windsor Container Available methods:

- `void RegisterService<TAbstraction>(string providerName, Type provider) where TAbstraction : class`
- `void RegisterSingletonService<TAbstraction>(string providerName, Type provider) where TAbstraction : class`
- `void RegisterTransientService<TAbstraction>(string providerName, Type provider) where TAbstraction : class`
- `void RegisterTransientService<TAbstraction, TImplementation>() where TImplementation : class, TAbstraction where TAbstraction : class`
- `void RegisterTransientService<TAbstraction, TImplementation>(TImplementation instance) where TImplementation : class, TAbstraction where TAbstraction : class`

- `void RegisterTransientService<TAbstraction, TImplementation>(string providerName) where TImplementation : class, TAbstraction where TAbstraction : class`
- `void RegisterSingletonService<TAbstraction, TImplementation>() where TImplementation : class, TAbstraction where TAbstraction : class`
- `void RegisterSingletonService<TService>() where TService : class`
- `void RegisterServiceCollection(Dictionary<Type, Type> toMapCollection)`
- `void RegisterScopedService<TAbstraction, TImplementation>(TImplementation instance) where TImplementation : class, TAbstraction where TAbstraction : class`
- `void RegisterScopedService<TAbstraction, TImplementation>() where TImplementation : class, TAbstraction where TAbstraction : class`
- `void RegisterService<TAbstraction, TImplementation>(Func<ComponentRegistration<TAbstra ComponentRegistration<TAbstraction>> configuration)`
- `bool IsServiceRegistered<TService>()`
- `bool IsServiceRegistered(string provider)`
- `TService Resolve<TService>()`
- `TService ResolveNonRequired<TService>()`
- `TService ResolveNonRequired<TService>(string key)`
- `T Resolve<T>(string key)`
- `object Resolve(Type type)` Example:

```
// register IAuditManager  audit manager implementation
 IoC.RegisterTransientService<IAuditManager, TManager>();
// Resolve IConfiguration  instance
 var configurator = IoC.Resolve<IConfiguration>();
```

1. `JsonParser` - is a helper for read json files Available methods:

- `T ReadArrayDataFromJsonFile<T>(string filePath) where T : class`
- `IEnumerable<dynamic> ReadDataListFromJsonWithTypeParameter(string filePath, Type entity)`
- `T ReadObjectDataFromJsonFile<T>(string filePath) where T : class` Example:

```
var path = Path.Combine(AppContext.BaseDirectory, "Configuration/WidgetGroups.json");
var items = JsonParser.ReadArrayDataFromJsonFile<ICollection<WidgetGroup>>(path);
```

1. `ObjectIdentificationHelper` - is a helper for check object type Available methods:

- `bool IsList(this object o)`
- `bool IsDictionary(this object o)`
- `bool IsInt(this string sVal)`
- `bool IsNumeric(this string sVal)` Example:

```
if (!jObject.ContainsKey(_section))
            {
                jObject.Add(sectionObject.IsList()
                    ? new JProperty(_section, new JArray(JArray.Parse(JsonConvert.
↪SerializeObject(sectionObject))))
                    : new JProperty(_section, new JObject(JObject.Parse(JsonConvert.
↪SerializeObject(sectionObject)))));
            }
            else
            {
                if (sectionObject.IsList())
                    jObject[_section] = JArray.Parse(JsonConvert.
↪SerializeObject(sectionObject));
                else
                    jObject[_section] = JObject.Parse(JsonConvert.
↪SerializeObject(sectionObject));
            }
```

1. `ResourceProvider` - get app settings file path, in dependency of app environment: [Development, Production, Staging] Example:

```
//The result can be appsettings.json
ResourceProvider.AppSettingsFilepath(hostingEnvironment))
```

1. `ResultModel<T>` - is a common return type for api and methods, it allow easy to check status of response and read the errors, it also contains helpers that allow easy understand the failure. Examples:

```
 public static async Task<ResultModel> UploadAsync(this IFormFile file, string␣
↪filePath)
        {
            var result = new ResultModel();
            try
            {
                using (var stream = File.Create(filePath))
                {
                    await file.CopyToAsync(stream);
                }

                result.IsSuccess = true;
            }
            catch (Exception e)
            {
                result.AddError(e.Message);
            }

            return result;
        }
```

## 6.11 Services

1. `WritableOptions<T>` - default implementation of IWritableOptions

## 6.12 Global settings

## 6.13 GearApplication instance

## 6.14 Mappers

### 6.14.1 AutoMapper

AutoMapper is supported directy from Core, created profiles are automatically scanned and registered, only that is need is to create your profile class. Example of profile:

```
public class OrganizationProfile : Profile
{
    public OrganizationProfile()
    {
        CreateMap<Foo, FooDto>();
        // Use CreateMap... Etc.. here (Profile methods are the same as configuration␣
↪methods)
    }
}
```

### 6.14.2 Mapster

Mapster is a library that support mapping type to another type, it has extensions for object type, sample example:

```
var user= new User();
var ldapUser = user.Adapt<LdapUser>();
```

More docs coming soon

CHAPTER 7

# 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 8

# Audit Module

## 8.1 Description

This module is used to log any data changes of database records. It is custom used for each entity: entities can be registered, can chose that fields to track and that to ignore. `Note`: Audit is compatible only with models that inherit from `BaseModel` class or implement `IBase<T>` This module has 2 tables that store your data changes:

1. `public DbSet<TrackAudit> TrackAudits { get; set; }`
2. `public DbSet<TrackAuditDetails> TrackAuditDetails { get; set; }`

## 8.2 Installation

In your startup class, register the service:

```
using GR.Audit;
using GR.Audit.Abstractions.Extensions;
...
//----------------------------------------Audit Module---------------------------------
↪-----
            config.GearServices.AddAuditModule<AuditManager>();
...
```

## 8.3 Usage

For use this module is need to install GR.Audit nuget package or download the library from github. In a few steps we can log our data changes.

1. Your module DbContext must inherit from `TrackerDbContext` Example of usage:

```
using GR.Audit.Contexts;
...
 public class GroupsDbContext : TrackerDbContext, IGroupContext
 {
 }
```

1. Your model must have attached the `TrackEntity` attribute like in an examples below: Track all fields:

```
using GR.Audit.Abstractions.Attributes;
using GR.Audit.Abstractions.Enums;
...

[TrackEntity(Option = TrackEntityOption.AllFields)]
    public class GearUser : IdentityUser<Guid>, IBase<Guid>
```

Track only selected fields:

```
TrackEntity(Option = TrackEntityOption.SelectedFields)]
    public class GearUser : IdentityUser<Guid>, IBase<Guid>
    {
        /// <summary>
        /// Stores user first name
        /// </summary>
        [MaxLength(50)]
        [TrackField(Option = TrackFieldOption.Allow)]
        public virtual string FirstName { get; set; }

        /// <summary>
        /// Stores user last name
        /// </summary>
        [MaxLength(50)]
        [TrackField(Option = TrackFieldOption.Allow)]
        public virtual string LastName { get; set; }
```

1. Register the context in your `IServiceCollection` extensions Example:

```
using GR.Audit.Abstractions.Extensions;
...
services.RegisterAuditFor<IIdentityContext>("Identity module");
```

If your context has early created migrations, after use the audit module, is needed to create new migrations. For docs on how to create migrations with EF Core, you can find here

For extend `IdentityContext` you can use the `TrackerIdentityDbContext` context Example:

```
 public class GearIdentityDbContext : TrackerIdentityDbContext<GearUser, GearRole,␣
↪Guid>, IPermissionsContext
 {}
```

Thanks of `BaseModel` we can update fields of record:

- `Created` - DateTime - the date of record creation
- `Changed` - DateTime - the date of record modification
- `Author` - user name of user that has created this record
- `ModifiedBy` - string - the user name of user
- `Version` - int - the version of record, is incremented on each change of record

Example on how to use the `BaseModel` class on your entity

```
  [DebuggerDisplay(@"\{{" + nameof(Name) + @",nq}\}")]
    [TrackEntity(Option = TrackEntityOption.SelectedFields)]
    public class Group : BaseModel
    {
    }
```

The audit is saved only after called SaveChanges(), SaveChangesAsync or Push(), PushAsync() context methods.

To ignore save audit of record, set to true the `DisableAuditTracking` field, this field is not saved and not serialized. Example:

```
var user = await _userManager.UserManager.Users.FirstOrDefaultAsync(x => x.UserName␣
↪== model.UserName);
user.LastLogin = DateTime.Now;
user.DisableAuditTracking = true;
await _userManager.UserManager.UpdateAsync(user);
```

In showed example save audit on change last login date for user is disabled.

CHAPTER 9

# Cache module

## 9.1 Description

This module is created for store some data in cache by key, now it has 2 implementations:

- `InMemory`
- `Distributed Cache (Redis)`

## 9.2 Installation

This module is registered in core of GR,WebApplication, it has configurations, for details on configuration, consult WebApplication module docs

But it can easily installed in Services configuration:

1. `InMemory` provider

```
services.AddCacheModule<InMemoryCacheService>();
```

1. `Distributed` cache

```
services.AddDistributedMemoryCache()
                .AddCacheModule<DistributedCacheService>()
                .AddRedisCacheConfiguration<RedisConnection>(configuration.
↪HostingEnvironment, configuration.Configuration);
```

## 9.3 Usage

For use this module, you need to inject the service, injection can be made using `IoC` helper or standard .net core injection: Example:

```
    [Authorize(Roles = GlobalResources.Roles.ADMINISTRATOR)]
    public class CacheController : Controller
    {
        /// <summary>
        /// Inject cache service
        /// </summary>
        private readonly ICacheService _cacheService;

        /// <summary>
        /// Cache options
        /// </summary>
        private readonly IWritableOptions<CacheConfiguration> _writableOptions;

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="cacheService"></param>
        /// <param name="writableOptions"></param>
        public CacheController(ICacheService cacheService, IWritableOptions
↪<CacheConfiguration> writableOptions)
        {
            _cacheService = cacheService;
            _writableOptions = writableOptions;
        }
```

Available methods:

- `Task<bool> SetAsync<TObject>(string key, TObject obj) where TObject : class;` - set a value in cache with a key Example:

```
string cacheKey = GenerateKey(s, key);
await _cacheService.SetAsync(cacheKey, new {});
```

- `Task<TObject> GetAsync<TObject>(string key) where TObject : class;` - get a value by saved previously in cache Example:

```
 var cacheData = await _cacheService.GetAsync<List<CountryInfoViewModel>>(Key);
```

- `Task RemoveAsync(string key);` - remove a value from cache
- `IEnumerable<CacheEntry> GetAllKeys();` - get all registered keys
- `IEnumerable<CacheEntry> GetAllByPatternFilter(string pattern);` - find keys by filter pattern
- `bool IsConnected();` - check if provider is available
- `void FlushAll();` - remove all from cache
- `string GetImplementationProviderName();`
- `Task<ResultModel<T>> GetOrSetResponseAsync<T>(string key, Func<Task<ResultModel<T>>> func) where T : class;` - get value, if value is null then the resolver is called, result is saved in cache and returned. Note: the result is saved only if the result state is OK Example:

```
var request = await _cacheService.GetOrSetResponseAsync("coinbasePro_products",␣
↪async ()
                => await SendServiceCall<IEnumerable<Product>>(HttpMethod.Get, "/
↪products"));
```

- `Task<T> GetOrSetResponseAsync<T>(string key, Func<Task<T>> func) where T : class;` - this method get or set the value, if value is null, then is resolver is called
- `Task<T> GetOrSetWithExpireTimeAsync<T>(string key, TimeSpan life, Func<Task<T>> func) where T : class;` - this method get and set values in cache, then the life argument is set for a specific time, the value from cache is available until the timer is not expired, if time expire, the value is renewed , renew process is not due automatically, is due in moment of call of method Example:

```
            var profit = _cacheService.GetOrSetWithExpireTimeAsync($"wallet_profit_
↪{currency.Code}_{source.Id}", TimeSpan.FromMinutes(15), async () =>
                await CalculateWalletProfitAsync(source.Id, source.WalletType.Symbol,␣
↪currency.Code)).GetAwaiter().GetResult();
```

# CHAPTER 10

# 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

## 10.1 Calendar Abstractions

### 10.1.1 Command line

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

### 10.1.2 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 11

# Workflow Builder Module

## 11.1 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

## 11.2 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

## 11.3 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);
```

## 11.4 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);
            }
        }
    }
```

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

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

Note: TActionClass need to inherit BaseWorkFlowAction

### 11.4.2 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

## 11.5 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

## 11.6 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

## 11.7 License

MIT

CHAPTER 12

# User Preferences module

## 12.1 Module description

Module name - `GR.UserPreferences` This module is created for store and get user preferences values from another modules with a custom configuration for each key, there is a web API that is responsible for all keys, there is validation at values and each key can be registered by several types such as list or text or something else, only the behavior in the UI remains and the configuration at values and validation

## 12.2 Abstractions

- `IUserPreferencesContext` - it contains abstractions to the entities with which the given module works
- `IUserPreferencesService` - represent the abstraction with that another modules or service can work
  - `GetValueByKeyAsync` - get the value of user preference by key name, the value is provided for currently logged user

```
Task<ResultModel<string>> GetValueByKeyAsync(string key);
```

  - `AddOrUpdatePreferenceSettingAsync` - set value for some key on user preference

```
 Task<ResultModel> AddOrUpdatePreferenceSettingAsync(string key, string␣
↪value);
```

  - `GetPreferenceConfigurationAsync` - get configuration of preference item with user selected option

```
    Task<BaseBuildResponse<object>> GetPreferenceConfigurationAsync(string␣
↪key);
```

  - `GetAvailableKeys` - get registered keys

## 12.3 API

The api with docs can be found on swagger, for that, you can write in url: http://you_gear_app_url/swagger , the category is `UserPreferencesApi`

## 12.4 Integration

### 12.4.1 Registration of module

Here we have a standard recording of the module with default configurations

```
//--------------------------------User preferences Module ---------------------------
↪-----
                        config.GearServices
                                .AddUserPreferencesModule<UserPreferencesService>()
                                .RegisterPreferencesProvider
↪<DefaultUserPreferenceProvider>()
                                .AddUserPreferencesModuleStorage
↪<UserPreferencesDbContext>(options =>
                                {
                                        options.GetDefaultOptions(Configuration);
                                        options.EnableSensitiveDataLogging();
                                });
```

### 12.4.2 Register of key

For do this is need to bind module extension on startup configuration with interface IServiceCollection

- key need to be unique and not empty
- TPreferenceItem can be anything class that inherit the PreferenceItem class and implement all parent class methods
- The Func is used for set and customize key configurations

```
/// <summary>
        /// Register user preferences key
        /// </summary>
        /// <typeparam name="TPreferenceItem"></typeparam>
        /// <param name="services"></param>
        /// <param name="key"></param>
        /// <param name="options"></param>
        /// <returns></returns>
        public static IServiceCollection RegisterUserPreferenceKey<TPreferenceItem>
↪(this IServiceCollection services, string key, Func<TPreferenceItem,␣
↪TPreferenceItem> options)
            where TPreferenceItem : PreferenceItem, new()
        {
            var provider = IoC.Resolve<DefaultUserPreferenceProvider>(
↪"PreferencesProvider_Instance");
            provider.RegisterPreferenceItem(key, options(new TPreferenceItem
            {
                Key = key
            }));
```

(continues on next page)

(continued from previous page)

```
            return services;
        }
```

### 12.4.3 Examples

#### List Type

Here is an example of register the user time zone preference with key: UserPreferencesResources.UserTimeZoneKey

```
 //Register timeZone
            services.RegisterUserPreferenceKey<ListPreferenceItem>
↪(UserPreferencesResources.UserTimeZoneKey, options =>
            {
                options.IsRequired = true;
                options.IsValidValue = value =>
                {
                    var response = new ResultModel();
                    if (value.IsNullOrEmpty()) return response;
                    try
                    {
                        TimeZoneInfo.FindSystemTimeZoneById(value);
                        response.IsSuccess = true;
                        return response;
                    }
                    catch (Exception e)
                    {
                        response.AddError(e.Message);
                    }

                    return response;
                };

                options.ResolveListItems = selectedZone =>
                {
                    //$"({zone.StandardName}) {zone.Id}"
                    var data = TimeZoneInfo.GetSystemTimeZones()
                        .Select(zone =>
                            new DisplayItem
                            {
                                Id = zone.Id,
                                Label = zone.DisplayName,
                                Selected = selectedZone == zone.Id
                            }).ToList();

                    return Task.FromResult<IEnumerable<DisplayItem>>(data);
                };
                return options;
            });
```

#### Text Type

here a simple key named `someKey` that the value of is required and on save the value must be equal with 5

```
services.RegisterUserPreferenceKey<TextPreferenceItem>("someKey", options =>
            {
                options.IsRequired = true;
                options.IsValidValue = value =>
                {
                    var response = new ResultModel();
                    if (value == "5")
                    {
                        response.IsSuccess = true;
                    }
                    else
                    {
                        response.AddError("Value is not equal at 5");
                    }

                    return response;
                };
                return options;
            });
```

### 12.4.4 Extend

To add new preference class, you must inherit from abstract class PreferenceItem and implement the methods, also you can extend current types as ListPreferenceItem and TextPreferenceItem

The implementation of list type

```
    public class ListPreferenceItem : PreferenceItem
    {
        /// <summary>
        /// Type
        /// </summary>
        public override string Type => "List";

        /// <summary>
        /// Get configuration
        /// </summary>
        /// <param name="currentValue"></param>
        /// <returns></returns>
        public override async Task<BaseBuildResponse<object>>␣
↪GetConfigurationAsync(string currentValue)
        {
            var data = await GetListValuesAsync(currentValue);

            return new BaseBuildResponse<object>
            {
                IsSuccess = true,
                Result = data,
                Type = Type
            };
        }

        /// <summary>
        /// Check if is valid
        /// </summary>
        /// <returns></returns>
```

(continues on next page)

(continued from previous page)

```
        public override bool Validate()
        {
            var isValid = ResolveListItems != null;
            return isValid;
        }

        /// <summary>
        /// Task for resolve list of items
        /// </summary>
        public Func<string, Task<IEnumerable<DisplayItem>>> ResolveListItems = null;

        /// <summary>
        /// Get list of items
        /// </summary>
        /// <param name="selected"></param>
        /// <returns></returns>
        public virtual async Task<IEnumerable<DisplayItem>> GetListValuesAsync(string␣
↪selected)
        {
            var items = await ResolveListItems(selected);
            return items;
        }
    }
```

CHAPTER 13

# Localization module

## 13.1 Description

Localization module is used for translate text from UI, auto translations, add/edit text, add new languages

## 13.2 Install

### 13.2.1 Register services

At the moment, are implement 2 methods:

- With json files

Add in `startup.cs` in ConfigureServices method

```
                        //---------------------------------Localization Module--------
↪-----------------------------
                        config.GearServices
                                .AddLocalizationModule<JsonFileLocalizationService,␣
↪JsonStringLocalizer>()
                                .BindLanguagesFromJsonFile(Configuration.
↪GetSection(nameof(LocalizationConfig)))
                                .RegisterTranslationService<YandexTranslationProvider>
↪(
                                        Configuration.
↪GetSection(nameof(LocalizationProviderSettings)))
                                .AddLocalizationRazorModule()
```

Need to add in appsettings this section:

```
  "LocalizationConfig": {
    "Languages": [
```

(continues on next page)

(continued from previous page)

```
      {
        "IsDisabled": false,
        "Identifier": "en",
        "Name": "English"
      },
      {
        "IsDisabled": false,
        "Identifier": "ro",
        "Name": "Romanian"
      },
      {
        "IsDisabled": true,
        "Identifier": "ru",
        "Name": "Russian"
      },
      //Here you can add another languages
    ],
    "Path": "Localization",
    "SessionStoreKeyName": "lang",
    "DefaultLanguage": "en"
  }
```

For all added languages in this json file, is needed to add a localization json file in Localization folder, of root folder or you can customize with another folder in your appsettings file

The name of file must be in this format: {LanguageIdentifier}.json For current configuration:

- ./Localization/en.json
- ./Localization/ro.json
- ./Localization/ru.json

`Note` : for all your environments `appsettings.{EnvName}.json` , is needed to add configuration of languages

For change the default language you must change the property `DefaultLanguage` from appsettings file. To disable a language, you can set `true` to `IsDisabled` property, and the language will be not showed in list for change language, also this can be done with UI module or api call (view this with swagger)

- With database

```
                        //---------------------------------Localization Module--------
↪------------------------------
                        config.GearServices
                                .AddLocalizationModule<DbLocalizationService,␣
↪DbStringLocalizer>()
                                .BindLanguagesFromDatabase()
                                .AddLocalizationModuleStorage<TranslationsDbContext>
↪(options =>
                                {
                                        options.GetDefaultOptions(Configuration);
                                        options.EnableSensitiveDataLogging();
                                })
                                .RegisterTranslationService<YandexTranslationProvider>
↪(
                                        Configuration.
↪GetSection(nameof(LocalizationProviderSettings)))
                                .AddLocalizationRazorModule()
```

## 13.3 Usage

This module can be used in multiple modes:

- with javascript

```
window.translate("key"); // the result will be the translated text with current␣
↪language from session
```

- in Razor

```
@Inject IStringLocalizer Localizer

var value = Localizer["key"]; //with C#
@Localizer["key"] //with razor
```

- with html attributes

```
<span translate="text_in_english_key">Text in english</span>
```

To translate a html block after document load, you can call `window.forceTranslate` with selector of block, as in example

```
window.forceTranslate("#MySelectorId");
window.forceTranslate(".MySelectorClass");
window.forceTranslate("span");
window.forceTranslate("*"); // to translate all tags
```

`Note` : if call of this method is done whitout selector, all document will be translated where is present `translate` attribute `Note` : The response of method is a promise, if you want to do some actions after translations, you must to apply `then` as in example

```
window.forceTranslate().then(() => {
    alert("Text in page was translated");
});
```

## 13.4 Available language packets

To find the available default language packs click Languages

CHAPTER 14

# Notification Module

## 14.1 Description

This module is intended for sending notifications to users, notifications can be sent to one or more users simultaneously, notifications are received real time

## 14.2 Install

### 14.2.1 Hub registration

```
        public override void Configure(IApplicationBuilder app)
                {
                        ...
                        app.UseNotificationsHub<GearNotificationHub>();
                        ...
                }
```

### 14.2.2 Module registration

```
                        //------------------------------Notification Module----------
↪--------------------------
                        config.GearServices.AddNotificationModule
↪<NotifyWithDynamicEntities<GearIdentityDbContext, GearRole, GearUser>, GearRole>()
                                .AddNotificationSubscriptionModule
↪<NotificationSubscriptionService>()
                                .AddNotificationModuleEvents()
                                .AddNotificationSubscriptionModuleStorage
↪<NotificationDbContext>(options =>
                                {
```

(continues on next page)

(continued from previous page)

```
                                        options.GetDefaultOptions(Configuration);
                                        options.EnableSensitiveDataLogging();
                                })
                                .RegisterNotificationsHubModule<CommunicationHub>()
                                .AddNotificationRazorUIModule();
```

## 14.3 Usage

In order to send notifications, it is necessary to inject the service:

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

## 14.4 Examples

1. An example of sending a notification to the user with id {userId} with the message “Hello from admin”

```
await _notify.SendNotificationAsync(new List<Guid> { userId }, new Notification
                {
                    Subject = $"Notification subject",
                    SendLocal = true,
                    SendEmail = true,
                    Content = "Hello from admin"
                });
```

CHAPTER 15

# Email Sender module

## 15.1 Description

Email sender is an implementation of SMTP sender, it has a base `IEmailSender` interface that implement `Microsoft.AspNetCore.Identity.UI.Services.IEmailSender` It also extend `ISender` that allow to send messages without inject `IEmailSender`

## 15.2 Installation

For us this module, it must be registered in your Startup class

```
//----------------------------------------Email Module--------------------------------
↪-----
                        config.GearServices.AddEmailModule<EmailSender>()
                                .AddEmailRazorUIModule()
                                .BindEmailSettings(Configuration);
```

In app settings is need to add the following settings:

```
  "EmailSettings": {
    "Enabled": true,
    "Host": "smtp.gmail.com",
    "Port": 587,
    "Timeout": 5000,
    "EnableSsl": true,
    "NetworkCredential": {
      "Email": "",
      "Password": ""
    }
  },
```

If register the Razor module, these settings can be edited from UI page , see in admin configurations of GEAR

## 15.3 Usage

This module ca be used in 2 ways:

1. Inject `ISender` Example:

```
using GR.Core.Services;
...
private readonly AppSender _sender;
...
var result = await _sender.SendAsync("email", $"Invite to {GearApplication.
↪ApplicationName}", message, model.Email);
```

1. Inject `IEmailSender` Example:

```
using GR.Email.Abstractions;
...
private readonly IEmailSender _emailSender;

await _emailSender.SendEmailAsync("user@mail.com", "Subject", "text");
```

CHAPTER 16

# 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
- add database localization provider
- refactoring on notifications api
- full refactoring on localization module
- add support for module appSettings files
- clean language files
- add support for import language pack
- add support for download laguage pack/paks(zip)
- add google pay configuration page
- finish google pay payment flow
- configuration of google play console for merchant id
- refactoring and improvements on subscription module, add new api
- add shiping and billing addresses on order details
- add apple pay module
- add polyfill for apple pay
- add apple pay configuration page
- add hot reloader for razor modules (only ui assets)
- add new extensions
- code refactoring
- add hot reloader for Views (Razor Class Library modules)
- separe sms phone 2-factor verification
- add abstraction module for 2-factor
- add email 2-factor verification
- order countries and cities by name
- fix time zones name
- add api for import countries from json
- add admin route: /admin
- add and refactoring paypal module
- add google pay payment module
- improvements on subscription module
- refactoring and fix bugs on product variations
- refactoring on commerce modules

- add seed product type on install
- add products store with filters
- move settings from appsettings file to custom files
- add validations for user preverences values

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