ПСС, Платформа

Система управления доступом

*13 стр.*

***Дата: 15.12.2016***

***Версия:*** [Ключевые слова]

***Статус: В разработке***

Оглавление

[Введение 3](#_Toc469595130)

[Назначение системы 4](#_Toc469595131)

[Компоненты системы 5](#_Toc469595132)

[Схема взаимодействия системы 6](#_Toc469595133)

[Описание работы ядра системы 7](#_Toc469595134)

[Принцип работы 7](#_Toc469595135)

[Алгоритм проверки доступа 8](#_Toc469595136)

[Для группы: 8](#_Toc469595137)

[Для пользователя: 8](#_Toc469595138)

[Что необходимо для подключения системы безопасности к приложению 8](#_Toc469595139)

[Описание настройки конфигурации системы 8](#_Toc469595140)

[Для конфигурации системы необходимо выполнить два действия: 8](#_Toc469595141)

[При подключении расширения Security.Web, необходимо: 9](#_Toc469595142)

[Схема базы данных 11](#_Toc469595143)

[Схема работы механизма проверки доступа 12](#_Toc469595144)

[Последовательность настройки системы доступа 13](#_Toc469595145)

# Введение

В данном документе представлено описание системы управления доступом для программных систем ПСС.

# Назначение системы

Система предназначена:

1. для настройки прав доступа пользователям и группам пользователей;
2. ограничения доступа к определенным ресурсам системы на основе политик безопасности;
3. аутентификации пользователя в информационной системе

# Компоненты системы

Система состоит из следующих компонентов:

1. База данных
2. Ядро системы (Security)
3. Компонент, описывающий интерфейсы взаимодействия системы (Security.Interfaces)
4. Компонент доступа к данным (Security.EntityDal)
5. Компонент описывающий модель данных (Security.Model)
6. Драйвер, для управления данными ядром системы (Security.EntityFramework)
7. Расширение для веб-приложений (Security.Web)
8. Система настройки прав доступа (Веб-приложение)

# Схема взаимодействия системы

Драйвер

DAL

Расширение для ядра

БД

Приложение

Система настройки

# Описание работы ядра системы

Ядро системы состоит из следующих компонентов:

1. Участники безопасности (пользователи и группы пользователей)
2. Роли
3. Типы доступа
4. Объект безопасности
5. Политика безопасности[[1]](#footnote-1)
6. Разрешения

Участник безопасности – это пользователь или группа пользователей, на которых накладываются роли (безопасности). Один пользователь может входить в несколько групп, одна группа может иметь несколько пользователей. Один участник безопасности может иметь несколько ролей, также как роль может предоставляться нескольким участникам безопасности.

Роль – это сопоставление, с одной стороны множества прав доступа (политик безопасности), необходимых для выполнения конкретных функций, а с другой стороны — подмножества [пользователей](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C), которые должны иметь эти права.

Тип доступа – это составляющая политик безопасности. Является статичной информацией, определяющейся во время проектирования ПО. Предназначена для настройки политик безопасности на низком уровне. Например, операции извлечения, добавления, редактирования, удаления данных из БД.

Объект безопасности – объект данных, процесс или набор процессов, к которым необходимо ограничить доступ.

Разрешение – это определение политики безопасности, т.е. назначение определенного типа доступа для объекта безопасности и предоставление этой политики безопасности для роли.

## Принцип работы

При входе в систему пользователю предлагается пройти идентификацию. После успешной идентификации пользователя, информация о нем сохраняется в куки клиента и передается при каждом новом запросе. При каждом запросе идентифицированный пользователь проходит аутентификацию в модуле аутентификации SecurityAuthenticateHttpModule. Доступ к пользователю текущего контекста можно осуществить с помощью HttpContext.User. Доступ к действиям контроллеров настраивается с помощью атрибутов, унаследованных от атрибута Security.Web.Mvc.AuthoriseAttribue (для приложений WebApi Security.Web.Http.AuthoriseAttribue), который объявлен в модуле Security.Web.AithorizeAttribute. Если запрос не является запросом Ajax, клиенту возвращается страница и ошибка с кодом 403, иначе сведения о об ошибки и код ошибки. Таким образом, в клиентском коде (javascript) необходимо учитывать эти ошибки в ответах от сервера.

## Алгоритм проверки доступа

Для проверки доступа используется метод CoreSecurity.CheckAccess(string login, string secObject, Enum accessType), как видно из описания к методу необходимо передать логин пользователя, наименование объекта безопасности и тип доступа. Проверка доступа происходит на уровне СУБД. Сначала проверяется доступ, если в качестве имени было передано имя группы (как участники безопасности: группы и пользователи хранятся в одной коллекции), далее если участник безопасности не является группой, то проверяется доступ для пользователя.

### Для группы:

производится выборка всех разрешений, роли которых установлены для группы, и если хотя бы одно разрешение соответствует переданным параметрам: объекту безопасности и типу доступа, то выдается положительное решение о доступе.

### Для пользователя:

аналогично группам, за исключением того, что выбранные разрешения включают в себя разрешения всех групп, в которые входит пользователь. К примеру, если пользователь **user1** входит в группу **Group1**, а для группы **Group1** установлена роль **Role1**, то разрешения, установленные для **Role1,** будут одинаково действовать и для пользователя **user1**.

## Что необходимо для подключения системы безопасности к приложению

1. Подключение библиотек системы к приложению;
2. Настройка объектов безопасности с помощью атрибутов или любым иным способом, реализовав интерфейс ISecurityObjects;
3. При использовании расширения Security.Web
   1. выполнить конфигурацию
   2. Подключить модуль аутентификации идентифицированного пользователя для сессии
4. Реализовать форму ввода логина и пароля
5. С помощью поставляемого API реализовать проверку идентификации пользователя
6. Для ручной проверки прав доступа необходимо вставить соответствующий код[[2]](#endnote-1) проверки в необходимых местах разрабатываемого приложения.
7. Настроить роли и разрешения для ролей.
8. Определить необходимые группы и пользователей, которые являются участниками безопасности
9. Назначить группам и пользователям необходимые роли.

## Описание настройки конфигурации системы

### Для конфигурации системы необходимо выполнить два действия:

1. Зарегистрировать модуль Ninject, который отвечает за загрузку зависимостей для системы безопасности, с помощью метода Config.RegisterCommonModule<CommonModule>();

. Для текущей версии доступна реализация модуля CommonModule пакета Security.EntityFramework, работающего с ORM EntityFramework и который реализует фабрику SecurityFactory.

1. Создать перечислитель (Enum), прописать в нем типы доступа[[3]](#footnote-2), затем зарегистрировать тип этого перечислителя с помощью метода Config.RegisterAccessTypes(typeof(EAccessType));

### При подключении расширения Security.Web, необходимо:

1. Создать класс атрибута авторизации унаследованного от Security.Web.Mvc.AuthorizeAttribute или Security.Web.Http.AuthorizeAttribute для WebApi-приложений.
2. Для необходимых методов-действий контроллеров проставить атрибут авторизации, например:

[ManualAuthorize(ObjectName = "Главная страница")]   
public ActionResult Index) {    
    return View();    
}

1. Определить тип доступа, который будет использоваться для определения доступа к методам действий контроллеров. Например,
2. Security.Web.Config.ExecAccessType = EAccessType.Exec;
3. Зарегистрировать сборку, которая требуется в защите от несанкционированного доступа

Security.Web.Config.RegisterSecurityAssembly(typeof(SecurityConfig).Assembly);

1. Зарегистрировать объекты безопасности, которые были добавлены в код вручную и установить их. Например,

{

Security.Web.Config.RegisterSecurityObjects(MainController.CustomSecurityObjects);  
SetUpSecurityObjects();

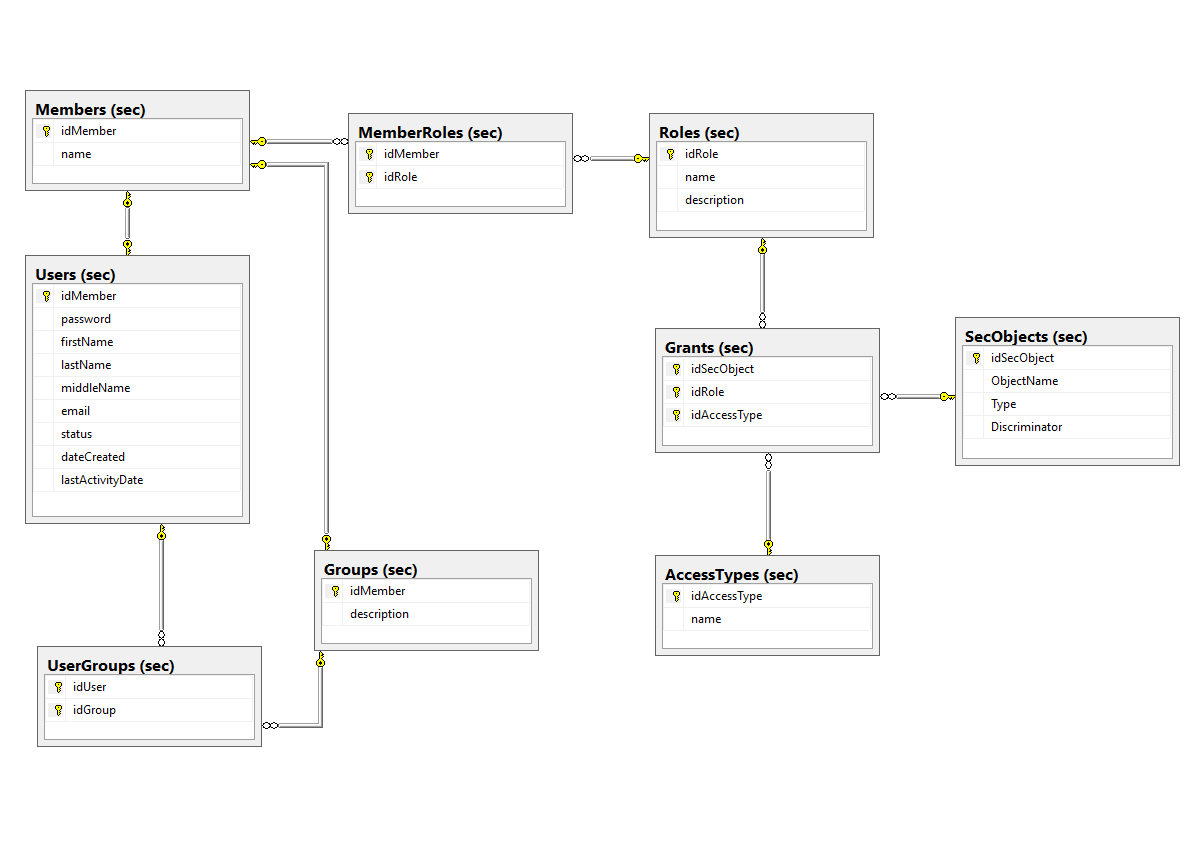
}

private static void SetUpSecurityObjects()  
{  
   using (var security = new BaseSecurity())  
      {  
          var securityObjects = Security.Web.Config.GetSecurityObjects().Select(s => s.ObjectName).ToList();  
          var sameInstalledObjects =  
             security.SecObjectCollection.Where(e => securityObjects.Contains(e.ObjectName))  
                    .Select(e => e.ObjectName)  
                    .ToList();  
          var newSecObjects = securityObjects.Except(sameInstalledObjects, StringComparer.OrdinalIgnoreCase);  
  
          foreach (var secObject in newSecObjects.Select(s => new SecObject() { ObjectName = s }))  
         {  
             security.SecObjectCollection.Add(secObject);  
         }  
  
         security.SaveChanges();  
    }  
}

    public class MainController : Controller  
    {  
 …  
        public ActionResult Test()  
        {  
            using (var security = new BaseSecurity())  
            {  
                var login = ((IUser)HttpContext.User.Identity).Login;

//ручная проверка доступа к объекту «Объект1»  
                if (security.CheckAccess(login, "Объект1", EAccessType.Exec))  
                {  
                    //some definition  
                    return new HttpStatusCodeResult(HttpStatusCode.OK);  
                }  
  
                return new HttpStatusCodeResult(HttpStatusCode.Forbidden);  
            }  
        }  
  
        public class SecurityObject : ISecurityObject  
        {  
            public string ObjectName { get; set; }  
        }  
  
        public class SecurityObjects : ISecurityObjects  
        {  
            private readonly List<ISecurityObject> \_objects = new List<ISecurityObject>();  
  
            public void Add(ISecurityObject @object)  
            {  
                \_objects.Add(@object);  
            }  
  
            public IEnumerator<ISecurityObject> GetEnumerator()  
            {  
                return \_objects.GetEnumerator();  
            }  
  
            IEnumerator IEnumerable.GetEnumerator()  
            {  
                return GetEnumerator();  
            }  
        }  
  
        public static ISecurityObjects CustomSecurityObjects()  
        {  
            return new SecurityObjects { new SecurityObject { ObjectName = "Объект1" } };  
        }  
    }

# Схема базы данных



# Схема работы механизма проверки доступа



# Последовательность настройки системы доступа



1. Совокупность объекта безопасности и типа доступа к нему является политикой безопасности. [↑](#footnote-ref-1)
2. [↑](#endnote-ref-1)
3. В простейшем случае (как и в большинстве) достаточного одного [↑](#footnote-ref-2)